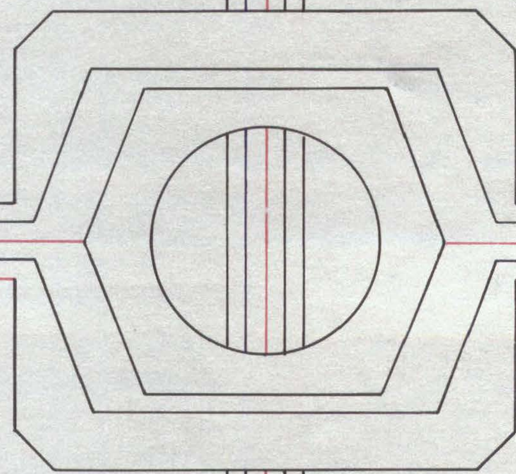
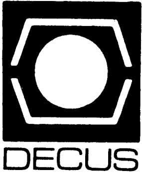
USA FALL 1986

# PROCEEDINGS OF THE DIGITAL EQUIPMENT COMPUTER USERS SOCIETY

D
E
C
U
S



25th SILVER ANNIVERSARY




# PROCEEDINGS OF THE DIGITAL EQUIPMENT COMPUTER USERS SOCIETY

## USA FALL 1986

Printed in the U. S. A.

"The following are trademarks of Digital Equipment Corporation"

ALL-IN-1
BASEWAY
DATATRIEVE
DEC
DEClab
DECmate
DECnet
DECpage
DECSYSTEM-10/20
DECUS
DECwriter
DIBOL
Digital logo
FALCON
IAS
LA100
MASSBUS
MicroPDP-11
MicroPower/Pascal
Micro/RSX
MicroVAX
MicroVMS
PDP (et al.)
PDT
P/OS
Professional
Q-bus
Rainbow
RSTS
RSX
RT
UNIBUS
VAX
VAXcluster
VMS
VT100 (et al.)
Work Processor
WPS-PLUS

Copyright © DECUS and Digital Equipment Corporation 1987 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.

*POLICY NOTICE TO ALL ATTENDEES OR CONTRIBUTORS "DECUS PRESENTATIONS, PUBLICATIONS, PROGRAMS, OR ANY OTHER PRODUCT WILL NOT CONTAIN TECHNICAL DATA/INFORMATION THAT IS PROPRIETARY, CLASSIFIED UNDER U. S. GOVERNED BY THE U. S. DEPARTMENT OF STATE'S INTERNATIONAL TRAFFIC IN ARMS REGULATIONS (ITAR)."*

DECUS and Digital Equipment Corporation make no representation that in the interconnection of products in the manner described herein will not infringe on any existing or future patent rights nor do the descriptions contained herein imply the granting of licenses to utilize any software so described or to make, use or sell equipment constructed in accordance with these descriptions.

Ada is a trademark of the U. S. Government, XEROX, and XNS are trademarks of Xerox Corporation, IBM, PROFS, PC-XT, and BITNET are trademarks of International Business Machines Corporation. UNIX is a trademark of AT&T Bell Laboratories, CP/M, PL/I are trademarks of Digital Research, Inc., MS-DOS is a trademark of Microsoft Corporation, TSX-PLUS is a trademark of S&H Computer Systems Inc, R: BASE 4000 is a trademark of Microrim, Intel 8088 is a trademark of Intel Corporation, LOTUS 1–2–3 is a trademark of Lotus Development Corporation, MULTIPLAN is a trademark of Microsoft Corporation, Mylar is a trademark of E. I. DuPont deNemours & Co., PLOTLN is a trademark of Image Research and Compugraphic Corporation, MUMPS is a trademark of Massachusetts General Hospital, Macintosh is a trademark and licensed to Apple Computer, Inc., Multibus is a registered mark of Intel Corporation, 8086 is a trademark Intel Corporation, VENIX is a trademark of VenturCom., Inc, Appletalk, and Apple II are a trademarks of Apple Computers, Inc., INGRES is a trademark of Relational Technology, Inc, Scribe is a trademark of Unilogic Ltd, UniLINK is a trademark of Applitek, HYPERchannel is a trademark of Network Systems Corporation, TIway is a trademark of Texas Instruments, Inc, 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, 68000 is a trademark of Motorola, Inc.

The articles are the responsibility of the authors and therefore, DECUS and Digital Equipment Corporations, assume no responsibility or liability for articles or information appearing in the document.

The views herein expressed are those of the authors and do not necessaily express the views of DECUS or Digital Equipment Corporation.

# TABLE OF CONTENTS


ARTICLE PAGE

**ARTIFICIAL INTELLIGENCE SIG**

Performance in the Evolution of Lisp as a General-purpose Language
Richard P. Gabriel ..... 1

A Distributed Knowledge Network Model
Steven Hughes ..... 7

Knowledge Representation Issues
Rebecca Wise ..... 13

**BUSINESS APPLICATIONS SIG**

Applications Software Design for the Multi-lingual Environment
Paul Mistretta, Phil Racine ..... 21

**COMMERCIAL LANGUAGES SIG**

Guidelines For Writing VAX COBOL Program Generators
Ray Davis ..... 29

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

MicroVAX Ancillary Control Process for Realtime Human-Machine Interface
Thomas Kane ..... 35

A Distributed Data Base For Real Time Data Acquisition and Process Control
David McGuigan, Robert Carey ..... 39

MicroVAX II Image Processing Tutorial
John Molinari ..... 47

Real Time Throughput of MicroVAX II and MicroVMS
Richard K. Somes ..... 51

Speakeasy: An Interactive Data Analysis Tool for the Research Scientist
David H. Saxe ..... 83

Real Time Acquisition Using the C Language
J-F Vibert ..... 95

An Investigation into the use of ELN in a Multiprocessor Compute Engine
Thomas Turano ..... 103

**DATA MANAGEMENT SIG**

Data Normalization
Pamela A. Valentine ..... 113

Data Engineering
James R. Yoder ..... 123

**DATATRIEVE/4GL SIG**

Advanced DATATRIEVE Record Definitions
Bart Z. Lederman ..... 129

Managing ALL-IN-1 with DATATRIEVE
Bart Z. Lederman ..... 139

Uses of Accent R
Winston Tellis ..... 149

ARTICLE PAGE

**EDUSIG**

MAKACT: An Account Maintenance Program for Large VAX/VMS Environments
Pat Feldner, George Stefanek ..... 161

Printing Across the Network
Ray Peterson, Mark Draughn, George Stefanek ..... 165

Faculty/Student Communications by Computer
Claude M. Watson ..... 167

Planning, Implementing and Managing a Comprehensive Campus-Wide Network
Don Shehi ..... 175

CMU Tutor
Bruce A. Sherwood ..... 179

Microcomputers: Support And Other Issues
David V. Cossey ..... 181

State of Washington School Network
Al Huff ..... 187

Enhancing Campus and Community Communications Through Voice, and Video Telecommunications
J. D. Thomas ..... 191

Data Management System For Academic Instructional Planning
Lisa M. Rotunni, Edward C. Hohmann, James A. Rounds ..... 195

**GRAPHICS APPLICATIONS SIG**

What is PostScript?
Ann Robinson ..... 207

Halftone - A Program For Converting Grey-Scale Images to Halftones
Robert Goldstein, Eli Peli, Karl Wooledge ..... 209

A Graphics Editor for 3-D CT-Scan Data For Musculo-Skeletal Modeling
L. M. Myers, W. L. Buford, Jr, D. E. Thompson ..... 213

Simplified User Interface for Technical Systems
Charles S. Janik ..... 219

Portable Graphics Packages for the C Language
J-F Vibert, J-N Albert, M. Rousseaux ..... 225

**HARDWARE AND MICROS SIG**

Giving the PDP-11/73 a Better Image
Clyde L. Tyndale and Richard B. Waltz ..... 229

A Simple Bootstrap Prom Programmer
Frank R. Borger ..... 249

**IAS SIG**

REESE BASIC (The Other BASIC)
Frank R. Borger ..... 257

A Multi-Terminal Task
Ted Smith ..... 261

| ARTICLE | PAGE |
|---|---|

## LANGUAGES & TOOLS SIG


Filling Some Holes in the VAX C Run-Time Library
Wayne E. Baisley .................................. 275

Choosing Automated Structured Analysis Tools
June Baker........................................ 281

A Generalized Coding Standard and Some Associated Tools
E. J. Straub, A. L. Slavich, C. Winter.................. 287

Systems Programming in a High-Level Language
E. W. Sewell....................................... 293

Choosing a Document Formatting System
Richard K. Wallace................................. 313

Use of the DEC Test Manager in an ANSI Standard Maintenance Test Strategy
James Tibbetts.................................... 317

Guided Tour of an Emacs Extension: dired
Peter Kaiser....................................... 327

Using the CMS Callable Interface
Glen Del Merritt .................................. 331


## NETWORKS SIG


Developing a Message Bus for Integrating VMS High Speed Task to Task Communications
Glen Macko........................................ 339

Network Print Servers
Robert E. McGee, W. V. Dixon ....................... 349

Using the KXT11-CA as an Intelligent Communications Controller
Arthur Hartwig.................................... 357

Development of a Packet Switch Exchange
Arthur Hartwig, Danny Smith........................ 365

DMI Tutorial and Design Approaches for a VAX-DMI Front-End
Roger Russ........................................ 369

TransLAN Technical Product Overview and Network Configuration Guidelines
Michael R Coker .................................. 375

MAP/OSI Protocol Package for VAX Computers
Stan Froyd........................................ 393

Utilizing the VAXcluster as a Network Hub
John Dennis....................................... 399


## OFFICE AUTOMATION SIG


Development of an In-house Training Program
Jennifer L. Rieck.................................. 405

From User Documentation to Sharing Information: Problems and Solutions in User Communications
Daniel Barrett..................................... 407


## PERSONAL COMPUTERS SIG


Remote Operation of the DEC Rainbow using MS-DOS
Larry D. Scott.................................... 415

Advanced PRO Toolkit
Robert Uleski..................................... 427


| ARTICLE | PAGE |
|---|---|

## RSX SIG


Lotus Blossoms Under RSX-11M-PLUS
Art Hurst......................................... 435

How to Get that Upgrade
Denny Walthers.................................... 449

Oceanographic Data Quality Control And Distribution System
Lloyd K. Thomas................................... 453

Archiving System for RSX
James B. Jackson .................................. 457


## SITE MANAGEMENT SIG


Organizing, Maintaining, and Distributing Software Products
Peter Heinicke, Tom Nicinski......................... 463

Developing a Computer Training Program for a DEC/IBM Environment at DuPont
Marlys Denison.................................... 469

Computer Room Design and Construction: A Case History
Brent Teeter P. E.................................. 473


## VAX SYSTEMS SIG


Effective use of VAX/VMS Autogen
Dennis L. W. Thury ................................ 479

Heterogeneous VAXclusters
Frank J. Nagy ..................................... 489

Primarily ULTRIX and a Little VMS on MicroVAXes
Wendy Rannenberg .................................. 493

XDELTA/DELTA Command Strings
B. C. Leahy........................................ 501

Trojan Horses, Worms, Viruses, and Robin
Steven Szep ....................................... 505

A VMS Response Logger- What the Users Think of Response Time
Robert B. Goldstein, Daniel P. B. Smith, Rivkah Stabiner.................................. 511

MOBIUS: New Directions in Micro and Host Integration
E. William Merriam ................................ 517

VMS File I/O Via QIO to an ACP
Al Tyrrill......................................... 521

The Overseer: An Activity Based Resource Management System for VAX/VMS
Steven G. Duff, Joseph W. Fiedeldey.................. 531

Ins and Outs of VMS Shareable Images
Ted A. Marshall.................................... 535

A New Technique for "System Performance Evaluation"
Schumann Rafizadeh ................................ 541

Programming with The VAX/VMS Screen Management Routines
Michael D. Orosz................................... 545

A VMS Facility for Data Encryption to the Data Encryption Standard
John Yardley ...................................... 553

VAX Network Backups
D. G. Darkangelo................................... 557


## POSTER PAPER


Creating Common Spooled Resources in a VAXcluster
Bob Rasmussen, Bob Nestor.......................... 565

# 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 50,000 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 Fall and Winter of 1986. It includes submissions from the Fall National Symposium.

The Fall 1986 Symposium was held at the Moscone Convention Center in San Francisco, California, from October 6 through 10, 1986. 6425 DECUS members attended the Fall Symposium in San Francisco, just 33 shy of a new record. They took part in birds-of-a-feather sessions, pre-symposium seminars, and over 1000 presentations made by both DECUS members and Digital. All of the papers within this volume were presented at that symposium.

The Mark Twain adage, "the coldest winter I ever saw was the summer I spent in San Francisco," didn't hold true for DECUS; San Francisco was in an Indian Summer the week we visited. The city (known to its natives as "The City") welcomed DECUS heartily; the only problem with the great success of this meeting was the lack of anticipation of our own success! Moscone Convention Center made for quite cramped quarters, and the huge number of attendees stretched the boundaries of each meeting room to their utmost.

Of the more than 75 papers in this volume, one-third were produced using the TeX document preparation system with Barbara Beeton's DeProc package. This provides a consistency and quality of document that has not been seen since the Proceedings were re-typed by hand on an IBM Magnetic Card Composer twenty years ago. As these DECUS Proceedings continue to mature, it is clear that machine-readable copies will be available in the near future. The new DECUServe project, aimed at bringing DECUS members together through electronic mail, database, and conferencing systems, would be a logical candidate as a distribution medium for future electronic editions of the Proceedings.

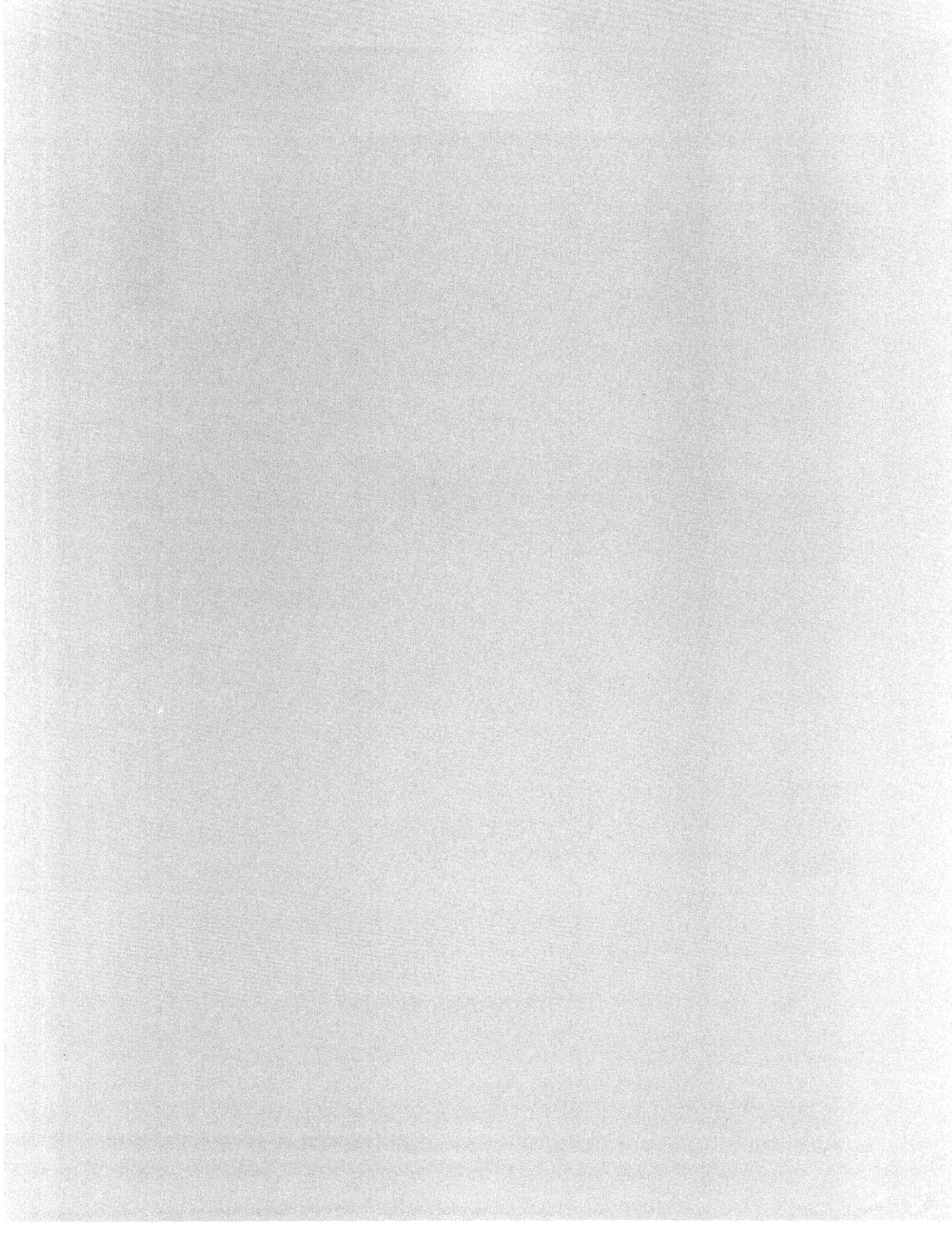
My thanks on behalf of the attendees of the Fall National Symposium go out to Mr. Joe Angelico and Dr. Jeffrey Jalbert, the DECUS volunteers who led the Symposium Committee. They worked together with DECUS staff members Ms. Nancy Wilga, Ms. Joanie Mann, and Ms. Gloria Caputo to put together the San Francisco 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. Judy Arsenault and Mr. Mark Grundler for their continuing support of this work.

Joel M. Snyder
Proceedings Editor
DECUS U. S. Chapter Publications Committee

# Papers Presented at

# Fall, 1986 National Symposium San Francisco, California October 6-10, 1986

# ARTIFICIAL INTELLIGENCE SIG



# Performance in the Evolution of Lisp as a General-purpose Language

by
R. P. Gabriel
Lucid, Inc.
Menlo Park, California

## Abstract

Lisp is the premier Artificial Intelligence programming language. It has not been used as a general-purpose programming language because its performance has been poor. Modern compilers and modern computers are improving the performance of Lisp so that it can be effectively used as a general-purpose programming language.

## 1. Introduction

Lisp was invented about 30 years ago to serve as the programming language for artificial intelligence reseach and for reasoning about programs. Lisp is a symbolic language in that the primary objects manipulated by Lisp programs are symbols or composites of symbols. Lisp has gained a reputation for being a slow language because many Lisps have never been extended to support a wide range of data types—programmers have had to emulate them—and because few Lisp implementors have concentrated on performance.

However, Lisp is now becoming a language with performance within the range expected of traditional programming languages. This paper presents some of the performance issues for Lisp along with how those issues have been addressed over the 30-year history of Lisp. We will see Lisp as a language becoming suitable as a general-purpose programming language.

## 2. Weak Typing

One of the most striking aspects of Lisp is that it supports weak typing. Weak typing requires that objects at runtime, rather than variables at compile time, have types associated with them. This usually requires that the format of objects as stored in memory be fixed, perhaps with some variation in representation among the types.

Everything in a Lisp system is a pointer, which is an address along with a tag. The tag specifies the type of the object and the tag plus the address specify the location of the object.

There are a number of techniques for implementing pointers on special-purpose machines (Lisp machines) and on general-purpose machines. One is to have the tag and the address be separate parts of the pointer. The Symbolics 3600 family uses this method—32 bits are set aside for the address and 4 bits for the tag.

On stock hardware there are a variety of methods for implementing pointers, depending on the underlying hardware. A pointer can be split across the address of the object and the object itself: A Lisp pointer can be simply an address that points to the object, in which the tag is stored—usually it is stored in the first (header) word. To determine the type of an object the header must be fetched, which involves a possibly additional memory fetch. A memory fetch might also incur the overhead of a page fault, and if the program is to simply examine the type of the object and not access any subparts of the object, this memory fetch and the possible page fault are unnecessary overhead.

Another method is to place the tag in the high-order byte of the address word. This way there are 8 bits for tag and 24 bits for address. If the addressing hardware

ignores the high-order byte, accessing the object is fast. However, objects of various types are scattered through the address space, and there are only at most $2^{24}$ objects of each type. Determining the type of the object involves a byte extraction and comparison.

A third method—probably the most commonly used technique for 32-bit, byte-addressable architectures—is called 'lowtag.' In an example lowtag system objects are stored on double-word boundaries and pointers are 32-bit words with the bottom 3 bits taken as the tag. The pointer is used as a byte address that points at some byte offset from the beginning of the object by the tag. To access a subpart of the object requires that an offset whose magnitude is equal to the tag be added to the pointer. Because the addition of a constant offset is frequently supported by the addressing hardware of the target computer, this operation is 'free.' If, in addition, the computer supports word-aligned loads with a hardware exception upon misalignment, type-checking can be nearly free.

Even in the case that type checking must be accomplished with inline code, the relative infrequency of type-checked operations with respect to the code emitted to implement the dynamic logic of a program leads to a minimal slowdown in the fully type-checked case.

Type-checking at runtime can nearly be eliminated with the addition of a layer of strong typing on top of a Lisp. Several Lisp dialects—MacLisp and Common Lisp—support such strong typing.

With the virtual elimination of type-checking at runtime, Lisp performance can be nearly as good as any other programming language, assuming that a similar programming style is adopted for Lisp as for the other programming language. That is, programming languages with static or stack-like memory management have an advantage over Lisp in that Lisp supports a relatively expensive form of dynamic memory management. If the Lisp programmer writes programs that use only static memory management, the performance differential between Lisp and procedural programming language will be minimal.

The problem of an inadequate set of data types has been met over the years. Lisp 1.5, the first Lisp dialect, had only 5 data types: atoms, fixed precision numbers, floating point numbers, arrays, and CONS cells. Common Lisp has approximately 20 data types, including a user-extensible record type.

The key to improved performance over the course evolution of Lisp has been the increasingly clever uses of data representations and the increased use of strong typing by compilers.

## 3. Garbage Collection

Lisp has provisions for dynamic storage management, which enables programmers to write programs that create new objects dynamically and that abandon those objects; the storage occupied by objects that have been abandoned can be reclaimed by a technique called 'garbage collection.' In the early literature the problem of how to reclaim storage allocated but no longer needed is called the 'erasure problem.'

Two basic techniques have been employed with many variations on each; one is true garbage collection in which storage no longer needed is placed into a free list of storage, and the other is garbage abandonment in which storage in use is moved from a current allocation area to a new allocation area.

Garbage abandonment is frequently referred to as 'stop and copy,' and it has some additional beneficial features. Because only storage in use is moved or

copied, the storage not in use is left behind. Copying to a new area compacts the storage in use—unused storage is typically found amidst used storage—and the paging behavior of the program is possibly improved. Because storage is allocated linearly within the storage allocation areas, faster allocation of storage for objects is accomplished. New objects allocated adjacently in time are allocated adjacently in storage, so that paging performance is also improved by this technique.

The essential procedure for both of these techniques is to stop normal Lisp processing when free storage is exhausted and to perform the action of finding free storage either by reclaiming now-unused storage or by abandoning it.

Variations on these techniques generally involve amortizing the cost of garbage collection or abandonment over the lifetime of the running program by performing some small percentage of the task while normal Lisp processing takes place. These techniques are called 'incremental' garbage collection, and when true garbage collection is the base technique reference counting is usually the incremental variation, and when garbage abandonment is the base technique Baker incremental garbage collection is usually the incremental variation.

The Xerox D machines implement reference counting while the Symbolics 3600 family implements Baker. Because the 3600 family now utilizes a composite technique called 'ephemeral' garbage collection, and because very few users of Symbolics machines enabled the Baker garbage collector when it was the only collection technique available on the machine, one can conclude that incremental garbage abandonment is not a workable solution. Furthermore, because the Xerox machines are used regularly with incremental reference counting enabled, one can conclude that it is a workable solution.

Ephemeral garbage collection is a composite technique; it comprises incremental Baker collection and incremental generation scavenging. Generation scavenging is a technique built, again, on top of garbage abandonment. The key observation for generation scavenging is that Lisp objects that will be eventualy garbage-collected have very short lifetimes. That is, if some object is to be abandoned it will be abandoned soon after it is created. Generation scavenging performs garbage abandonment in a context that takes into consideration only the most recently created objects. There are a series of 'generations,' and recently created objects are copied from one generation to the next as in garbage abandonment. If an object is to be eventually abandoned, it will probably be abandoned while in one of the generations. Objects that make it past the last generation are placed in the permanent heap.

Because generation scavenging is simply a variation on garbage abandonment, it can be performed incrementally.

Reference counting requires a reference count—a small number—to be associated with each Lisp object, that count being the number of other Lisp objects that point to the one in question. If the reference count is not zero, then the object is in use; if it is zero, it is not in used and the storage for that object can be collected. In practice it is not possible to keep accurate reference counts—the values can become larger than most practical storage sizes assigned to hold such counts—and circular structures can lead to locally correct but globally incorrect counts.

No Lisp implementation on stock hardware implements reference counting, but other garbage-collection-based languages on stock hardware have used reference counting.

As it happens, incremental garbage abandonment is not best suited to stock

hardware, because the techniques require a change of representation to the Lisp objects at some point in the collection process, and the change of representation will not be made at the same time to all Lisp objects; the Lisp process will be required to perform different operations depending on the state of the garbage collection. Moreover, the nature of the representation change also can result in much worse paging behavior, so that the conditionalization along with the increased paging activity might slow down the Lisp process, perhaps to the point where the incremental time spent on the conditionalization renders the technique unusable.

Generation scavenging is a technique that can be implemented on stock hardware (again, other garbage-collection-based languages than Lisp implement generation scavenging) and some major Lisp implementations may soon implement it to see whether the amortization of collection over normal Lisp processing is effective.

The problem with garbage collection on Lisp stock hardware is not that the start-to-finish time with a stop-style collector is worse than the start-to-finish time for Lisp machines with incremental collection, it is that the pauses might be either distracting, frustrating, or intolerbale for real-time applications.

The main hope for stock hardware is that RISC technology and large, fast caches will enable the cycle count for the overhead steps in incremental collection to be small with respect to the overhead steps inherent in Lisp processing. That is, if, for example, incremental collection requires 5 additional instructions to be executed every 50 instructions, the impact of these additional instructions is minimal if they are single cycle instructions while many of the other 50 are multi-cycle instructions.

## 4. Compilers

Over the evolution of Lisp, compiler technology has improved dramatically. It is in this area, rather than in areas of clever design of Lisp implementations, that performance has been gained for Lisp programs over against programs in other languages.

Early Lisp compilers were quite simple; compilation involved compiling into machine language the control aspects of programs—conditionals, function calls and returns, and GO statements—as well as the binding of variables. Calls to low level data structure accessors and modifiers was accomplish with function calls. This style of compilation resulted in these low level routines being relatively slow, because the work they accomplished was on the order of single instructions while the protocol overhead for function calling is on the order of tens of instructions.

Numeric processing remained slow for many years because the representation of numbers in Lisp was not efficient, and because there were not good techniques for strong typing (to eliminate allocating intermediate numeric results in the heap) and because there were no good techniques for stack allocation of these intermediate results.

The first compiler to solve some of these problems was the MacLisp compiler in the early 1970's.[White 1970] This compiler had sufficient strong typing capabilities to enable programmers to express efficient numeric code, and MacLisp solved the stack allocation problem using a technique called 'pdlnums' (**P**ush **D**own **L**ist **NUM**ber**S**). The numeric performance of code produced by this compiler was benchmarked at speeds comparable to that produced by contemporary FORTRAN compilers.

Since then, strong typing has been applied to type checking situations, so

that not only are the low level data structure manipulation primitives 'open-coded'—coded as a sequence of instructions rather than as function calls—but the type checking usually necessary before the application of the primitives can be eliminated by reasoning about the types of the variables at compile time.

The first very high quality Lisp compiler was the S1 Lisp compiler [Brooks 1982a], [Brooks 1982b]. This compiler was the first to do reasonable register allocation, data flow analysis, binding environment analysis, and lexical closure analysis. Furthermore, it was the first compiler to include a pdlnum analysis phase in such a way that the flow of information of the analysis could be understood and modified.

More recently, Lisp compilers have been able to begin to produce code whose performance is similar to that of code produced by compilers for procedural languages like C.

The following are results from a small series of benchmarks, which were performed to measure the approximate relative speeds of C and Lisp in the best of situations. There are three benchmarks: triangular numbers, the Traverse benchmark, and the Puzzle benchmark.

The triangular numbers are defined as,

$$T(n) = \sum_{i=1}^{n} i$$

In the two programs—one in Lisp, the other in C—the Lisp code is in a very Lisp-like programming style, using temporarily defined functions in place of iteration constructs; the C code is an straightforward iterative formulation of the definition of triangular numbers. The Lisp code is about 30% faster than the C code.

Traverse is one of the benchmarks from the Gabriel Benchmark suite [Gabriel 1985], and it builds a random graph in the initialization phase and performs a garbage collection algorithm over it.

The initialization phase—building the graph—runs 30% faster in Lisp than in C, while the collection phase runs 20% faster in C than in Lisp.

The third benchmark is the Puzzle benchmark; it is one of the classic benchmarks used both in the Gabriel Benchmark Suite and in the Pascal/C world. Here C is 17% faster than Lisp.

All that can safely be concluded from this small benchmark series is that Lisp in not necessarily large factors slower than procedural languages.

## 5. Machine Architectures

The key to faster Lisp systems are faster processors with fewer cycles per instruction and many general-purpose registers; faster, larger caches, possibly partitioned into instruction, data, and stack caches; and larger, faster memories of at least 16 megabytes; faster paging disks.

Lisp is not an inherently large or slow programming language, but with the program development features it provides it is easy to write very large programs. These large, user-written programs are what drive the need for larger, faster machines.

## 6. Conclusions

Lisp is jumping the performance hurdle; this hurdle has been one of the reasons that Lisp has not been the programming language of choice for general-purpose programming. Lisp programs are large, because it is easy to write large programs.

## References

[**Brooks 1982a**] Brooks, R. A., Gabriel, R. P., Steele, G. L. *An Optimizing Compiler For Lexically Scoped Lisp*, Proceedings of the 1982 ACM Compiler Construction Conference, June 1982.

[**Brooks 1982b**] Brooks, R. A., Gabriel, R. P., Steele, G. L. *S-1 Common Lisp Implementation*, Proceedings of the 1982 ACM Symposium on Lisp and Functional Programming, August 1982.

[**Gabriel 1985**] Gabriel, R. P., **Performance and Evaluation of Lisp Systems**, The MIT Press, 1985.

[**White 1970**] White, J. L., **An Interim Lisp User's Guide**, Masachusetts Institute of Technology, Artificial Intelligence Memo No. 190, 1970.

# A Distributed Knowlege Network Model

**J. Steven Hughes**
Jet Propulsion Laboratory
Pasadena, California

## Abstract

This paper presents a model of a distributed system architecture that promotes the implementation of knowledge based objects in a data flow network. The model which is based on the integration of a message passing protocol, data flow concepts, the object-oriented model, and knowledge base processing, provides an alternative to a strictly hierarchical network control structure and also exploits parallelism. The objects manage knowledge bases through the use of procedures that have been developed using tools and techniques from a variety of system engineering disciplines such as symbolic processing, data management, and numeric processing.

## Introduction

The need for addressing distributed knowledge base processing is evident when one considers the tools that are now available to the systems engineer. In the area of hardware architecture, advances in microprocessor technology and the subsequent availability of cheap, powerful, and distributed workstations allow systems engineers to consider distributed solutions when faced with complex problems. In the area of software development, Artificial Intelligence (AI) research has made available a variety of tools and techniques that have been developed for addressing hard problems. The emergence of a subdiscipline termed symbolic processing, promotes the use of these tools and techniques. Expert systems for example are no longer limited to the research laboratories but are commercially available and readily applied to a wide range of practical tasks.

Increasingly, systems engineers will merge conventional methods, symbolic processing techniques, and distributed processing in the design and development of systems. The resultant systems will integrate disparate modules as well as manage networks that are more sophisticated than the typical distributed processing networks. These resultant networks will contain more powerful modules such as expert systems that will interact with one another in the problem solving process and will need to know the distribution of network components [5].

In the balance of the paper, a model of a distributed knowledge network is developed that is based on the integration of concepts from the object-oriented model, the data flow model, data management, and the Time Warp discrete event mechanism. This model supplies a distributed environment in which a variety of organizational structures may be implemented. Preliminary to the development of the model, there are brief discussions on the criticality of resource management, a review of computer processing trends that supply a perspective on the issues, and a brief overview of the four subject areas. Finally, two issues that the model addresses are discussed.

## Early Lessons

At a recent conference on Expert Data Base Systems, four early lessons from distributed AI research were presented. These include 1) the need for more formal software engineering, 2) the criticality of resource managment, 3) that new solutions result in old and new problems, and 4) the currently available tools are inadequate. The broad scope of these lessons reveals both the infancy of the subdiscipline of distributed AI and a number of directions for research. The second lesson, the criticality of resource management, is briefly discussed to help focus our attention.

Fox [1] views distributed systems as being analogous to human organizations and applies concepts and theories from the management science field of organizational theory to help in the design of distributed systems. He concludes that as problems to be automated grow in size, resource limitations appear which limit the success of the resulting implementations. These resource limitations can be viewed as being caused by "bounded rationality". Referenced as Simon's theory of bounded rationality, it implies that both the information absorbed and the detail of control are limited for a single entity in a system.

Fox observes that the encapsulation of both mechanism and information is primary to the proper structuring of an organization. When the resulting concepts from organizational theory are applied to the design of a distributed system, five requirements result:

- the products of the process must be well defined;
- the interactions between processes must be minimal;
- the effects of a process upon other processes must be understood;
- clear lines of authority must be recognized;
- clear lines of information flow must be recognized.

These five requirements not only focus attention on some distributed application development issues but also help define the characteristics of the distributed environment in which the application is to be implemented.

## Computer Processing Trends

From an application point of view, the mainstream usage of computers is experiencing a trend of four ascending levels of sophistication: data processing, information processing, knowledge processing, and intelligence processing. [2] Data processing involves data items which are considered mutually unrelated, such as numbers, character symbols, and multidimensional measures. Next in level of sophistication is information processing, the processing of data items that are related by some syntatic or relational structure. Knowledge processing is the processing of information items with associated semantic attributes and finally intelligence can be view as what is gained from a collection of knowledge items.

These trends help to distinguish the techniques and tools currently in use by system engineers. For example, information processing, a critical function in any modern business, is typified by the processing of information stored in data base management systems. These systems allow data items to be stored, updated, and retrieved in an efficient and useful manner, based on the constraints, relationships, and specifications stored in a data base schema.

In an AI application, these four trends can also be considered to be levels of processing that will take place either in support of or as an integral part of the implementation. Raw data items will first have to be collected and validated, relationships determined and implemented in relational structures, semantic meaning applied either declaratively or procedurally and ultimately intelligence gained.

## Conceptual Basis

Based on the preceeding discussions, it is apparent that we need a distributed environment that promotes system modularity, flexibility, and efficiency while allowing the integration of tools and techniques from a variety of disciplines. The model proposed in this paper attempts to address these issues and is the result of the integration of concepts from four subject areas. These include the object-oriented model, the data flow model, data management, and the Time Warp discrete event mechanism. The lesson previously mentioned about new solutions resulting in old problems, support the use of tools and techniques from other disciplines whenever applicable. The first three subject areas are in this category. The Time Warp discrete event mechanism is a relatively new tool and supplies a critical item in the integration. In the following we will briefly discuss each of these areas.

## Data Management

Martin [6] states that the term "data base" became popular about 1970. Since that time, a number of characteristics attributable to data bases have developed. These include data independence, speedy handling of spontaneous information requests, nonredundancy, versatility in representing relationships between data items, security, protection and real time accessability. He subsequently defines data management as a general term that collectively describes those functions of the system that provide creation of and access to stored data, enforce data storage conventions, and regulate the use of input/output.

Data Management is an important issue in knowledge processing, since as we have shown above, knowledge is derived from data. For example, an expert system is typified by the aspects of knowledge representation and search.

Knowledge representation implies the storage of data, syntax, relationships, and semantics. Therefore, once knowledge engineering is complete, there exists a data management aspect of the system whether the knowledge base consists of rules and facts or any other representation. In turn, the search aspect of an expert system has the largest impact on efficiency since in its uncontrolled form, combinatorial explosion occurs. Solutions to the combinatorial explosion problem are typically a combination of more efficient search heuristics and continued knowledge engineering. The continued knowledge engineering of course increases the data management load.

## Object-Oriented Methodology

Where the term data base introduces the concept of data independence, object-oriented introduces the concept of both data and procedural independence. According to Stefik and Brobow [7], an object is an entity that combines the properties of procedures and data since it performs computations and saves its local state. Some applicable attributes of the object-oriented model follow. Objects in an object-oriented environment communicate using messages that specify the receiving object, a method (procedure) for processing the message, and the data. External to the object, there is no knowledge of procedural implementation, data structure, or state. The object-oriented model also supports data and procedural abstraction and encapsulation.

## Data Flow

The concept of data flow has been ingrained in computer science from the beginning. For instance, early analog computers with their operational amplifiers and potentiometers were early data flow machines. Later, certain systems analysis and structured design techniques emphasized the creation of data flow graphs during the system analysis phase. Finally data flow hardware architectures are currently being considered as alternatives to Von Neuman machine architectures.

There are several attractive characteristics of the data flow model that will be considered here. The first is the concept of dependencies. A data flow graph typically consists of nodes and arcs where the nodes represent processing and the arcs represent data flow and dependencies between nodes. For example, a simple data flow graph with two nodes A and B connected to a third node C by two directed arcs toward node C, represents processing in node C that is dependent on prior processing in nodes A and B. Most importantly, node C is considered ready for processing as soon as the output from nodes A and B is available.

The next characteristic involves the ease of mapping many problems onto a data flow architecture. In most software applications, there will exist several data representations and most tasks will require conversions between those representations. For example, a single user request in an interactive application may require data from several sources. Each of these sources probably has its own representation and the data will generally require several conversions before final presentation. These conversions, considered as events may be expressed as either a series of filters along a pipeline or a series of objects communicating via messages [4].

The final characteristics involve parallelism. Data flow is considered an excellent model for exhibiting parallelism. A data flow graph resulting from systems analysis of a proposed or existing system helps exhibit much of the inherent parallelism of the system.

## Time Warp Discrete Event Mechanism

The Time Warp mechanism [3] is essentially a distributed operating system specialized for simulation. It uses virtual time (simulation time) stamped messages to signify when events are to occur. Objects or processing entities are dynamically created and destroyed and communicate via the time stamped messages. There is no required pattern of communication since objects may send a message to be received by any other object at the present or any future time.

## Concept Integration - Object-Oriented and Data Management

Zaniolo, et al [8] have reviewed the research involved with the "grand unification" of data base technology, the object-oriented model, and logic programming. The end result of this research is expected to be a powerful expert data management model which integrates the salient features of the three subject areas. While the full scope of the research is no doubt applicable, only a part will be discussed here in support of our proposed model.

Modern data base management systems typically center around a data dictionary. The data dictionary is itself a data base and consists of meta data (data about data) pertaining to the data in the target data base. This meta data can include item and group descriptions, logical data base description, sources of the data and users of the data. Typically a data dictionary, especially its schema (logical data base description), is static in nature. In other words, once the description of the data base has been specified in the dictionary, it is not easily changed.

However, the need for more dynamic data dictionaries is evident in at least two applications areas, CAD/CAM and knowledge based engineering. For example, in knowledge based applications where knowledge engineering plays a critical role, the data structures and their descriptions need to evolve over time. Specifically the cycles of data base design, dictionary definition and data acquisition are intertwined and need to be integrated. In other words, a data dictionary, or the meta data, should be able to be used as a knowledge base for data base design and planning.

The paper proposed a knowledge based management model based on the object-oriented methodology. In this model there is no clear distinction between data and meta data as far as implementation and management are concerned. Instead, a continuum of concepts, from tokens representing data base instances to objects representing goals and learning criteria would exist. Data base tokens, types, operations, and transactions would all be implemented as individual objects.

The integration of data and meta data using the object-oriented model, leads us to the concept of partitioning the knowledge in a knowledge based application along a functional line and then developing objects that manage the individual groups of knowledge items. This management would include not only the traditional controlling of access to the data, ensuring data integrity, supervising the distribution of data, and documenting the content and structure of the data base, but could also include more semantic level concepts such as organization via semantic nets and behavioral specifications. Behavioral specifications could include how individual knowledge bases relate to one another and the external environment when accessed.

## Data Flow Architecture - Data Flow and Time Warp

The next level of integration consists of applying the Time Warp mechanism to the data flow model to produce a distributed environment. As has been shown, the Time Warp mechanism is essentially object-oriented, allowing communication between objects using time stamped mes-

sages. These time stamped messages may be one of two types. First, event messages are the main elements of control since they not only pass message text between objects but also signify when an event is to occur. Specifically, when an object processes an event message, it executes with its local time set to the receive time of the message. The Time Warp mechanism insures proper ordering of object execution. Query messages, the second type of messages, allow the querying of the state of an object. The query, a request and corresponding reply, does not allow the state of the queried object to be changed and the local times of the objects involved are assumed to be equal.

The mapping of data flow graphs into an implementation of the Time Warp mechanism is readily apparent. Graph nodes are implemented using objects and the data flow dependencies are handled using time stamped messages. The flexible nature of Time Warp message passing mechanism will allow the implementation of varied organizational structures and possible dynamic restructuring of the system.

### Realization of a Knowledge Network

We next consider each node of the data flow architecture to be a knowledge base object. Having as a subset the properties of a data base management module, the object will be able to manage data and relational structures. With the additional capability of being able to handle semantic attributes such as behavior, an object acts as an expert in regard to its own knowledge base. This includes how the knowledge base relates in terms of dependencies to other knowledge bases and how it responds to updates, ageing of data, and requests for information.

A shift of emphasis should be noted here with regard to the nature of the model and the conceptualization of applications that use it. At least two concepts support a data driven view as opposed to a procedure driven view of processing. First, the data flow model supports the former since it is the underlying network architecture.

In a more subtle manner, the objects themselves take on more of the characteristics of the knowledge base they manage as opposed to the type of processing present in the object. The notion that messages are handled by methods (procedures) of an object, relegates the processing, whether an inference procedure or a simple array reference, to a role supportive of the knowledge base. The characteristics of the individual nodes are now more closely related to the knowledge that exists at the node and its dependencies involving knowledge bases in parent and sibling nodes in the data flow network. The processing consists of managing a class of knowledge items and supporting the data flow model by notifying all nodes that manage knowledge bases that might be affected by any local modifications.

### Issues Addressed

At least two issues are addressed by the Distributed Knowledge Network Architecture presented. The first involves the design of an interactive application that processes continuous streams of input data. Some general requirements of such a system are:

- validate incoming data,
- update the appropriate data base,
- handle effect of input data on current state of the system,
- perform ageing of processed data and subsequent removal,
- handle effect of removed data on current state of the system,
- handle user information requests,
- handle user processing requests,
- minimize response times,
- insure data security and integrity,
- dynamically alter processing mode to handle either a change in user needs or a change in incoming data rates.

The proposed model addresses in particular the requirements of handling the effect of local modifications on the state of the system. Through the use of dependencies in the data flow network, knowledge base objects can propagate changes throughout the system. This ripple effect can occur without hierarchical control.

The proposed model also addresses the issue of parallelism since data flow is considered a good model for exploiting parallelism. Parallelism can be classified as either temporal, spatial, or asynchronous [2]. Temporal parallelism is that which occurs in a pipelined architecture. In a data flow model, temporal parallelism would exist between adjacent connected nodes when both nodes process simultaneously. For example, consider two nodes A and B where the output from node A is input to node B. Temporal parallelism occurs when an input data stream causes both nodes A and B to process simultaneously. Speedup is realized by considering the time required to do the same processing when node A is allowed to process and then is blocked as node B processes.

Spatial parallelism is typified by a Single Instruction Multiple Data (SIMD) hardware architecture. In this model, the data is partitioned out to several processing elements that all perform the same function. In the data flow model, if a particular node was determined to be a bottleneck and if the data being processed at that node was partitionable then either an SIMD machine could be installed at the node or the processing could be distributed to more than one identical node on separate processing elements to achieve speedup.

Asynchronous parallelism occurs when two or more objects process simultaneously without lockstep control.

This implies a loose dependency between the asynchronous objects. The discrete event mechanism inherently allows this.

### Conclusion and Qualifications

This model was developed while implementing a distributed situation assessment prototype. Three modules had been developed independently and the remaining task was to perform the integration. The modules included a message parser for handling free text messages, a preprocessor module that used a numerical clustering algorithm for data reduction, and a rule based expert system that infered higher level attributes about the data clusters. A user interface module was in the process of being developed. The added requirment of handling a continuous stream of input data prompted the idea of developing data base objects supporting the three modules. The evolution to knowledge base objects, where the procedural aspects of the original modules are implemented as methods, resulted after considering the object-oriented model in more detail.

The model is still somewhat intuitive since language constraints and resource limitations have limited the implementation effort. These constraints are in the process of being resolved.

The availability of the Time Warp Simulator on a uniprocessor system provides a friendly environment for the development of the application. For example, the simulator has its own development tools, the system debugging resources of the host machine are available, and the initial development and debugging of a distributed system in a simulated concurrent processing environment will certainly be easier.

### References


[1] Fox, M., "An Organizational View of Distributed Systems," IEEE Transactions on Systems, Man, and Cybernectics, Vol. SMC-11, No.1, January 1981.

[2] Hwang, K. and Briggs, F. A., Computer Architecture and Parallel Processing, McGraw-Hill, 1984.

[3] Jefferson, D. and Sowizral, H., "Fast Concurrent Simulation Using the Time Warp Mechanism," SCG conferences on Distributed Simulation, San Diego, January 1986.

[4] Laussana, R., "Pipelining with Unix," Computer Graphics World, April 1985.

[5] Lesser, V. and Corkill, D., "The Distributed Vehicle Monitoring Testbed: A Tool For Investigating Distributed Problem Solving Networks," The AI Magazine, Fall 1983.

[6] Martin, J., Principles of Data Base Management, Prentice-Hall, 1976.

[7] Stefik, M. and Brobow, D., "Object-Oriented Programming: Themes and Variations," The AI Magazine, Winter 1986.

[8] Zaniolo, Ait-Kaci, Beech, Cammarata, Kerschberg, and Maier, "Object Oriented Database Systems and Knowledge Systems," Expert Database Systems, Proceedings from First International Workshop. 1986.

# Knowledge Representation Issues

**Rebecca Wise**
Amdahl Communications Systems Division

**Abstract**

Knowledge representation is an area of active research within the field of Artificial Intelligence. This paper gives an introduction to this field. First, some justification for even discussing knowledge representation is presented. Next, an introduction to the knowledge representation field is given, followed by some criteria for evaluating alternative representations. A survey of several popular representations is given next. Finally, some areas of research into the knowledge representation problem are discussed. The bibliography includes several references to allow further study into this area.

Much attention has been given recently to the expert systems methodology for problem solving in the Artificial Intelligence arena. However, expert systems are just one of the many techniques that Artificial Intelligence research has made available for use in problem solving. There is a great deal of information available on the expert systems methodology. However, there has been little emphasis placed on the alternatives to expert systems. This paper addresses that void by providing information on the other mechanisms for representation and problem solving that are in use or have been used.

In addition, there are several problems facing the Artificial Intelligence researchers that center around the knowledge representation issue. Many people believe that true intelligence can not be achieved without incorporating a vast amount of commonsense knowledge into the computer's base of knowledge. This commonsense knowledge problem is really a problem of how to represent and retrieve this knowledge. Thus, the problem is one of finding an appropriate knowledge representation mechanism.

In addition to the problem of commonsense knowledge, there are huge knowledge bases that must be constructed even in specialized domains. The technology simply does not exist to properly handle such enormous amounts of data.

## Definition of Knowledge Representation

Knowledge Representation refers to the method of storing the information about a problem. In the traditional software environment, data structures are considered to be the knowledge representation technique. The knowledge representation that is used for a problem is closely related to the technique that is used to solve the problem. The solution for a given problem includes both the algorithm used to solve the problem and the knowledge representation selected to store the information used in the solution process. The knowledge representation selected to solve a problem should, in some sense, make the solution to the problem appear trivial (in the best case) or at least obvious, making it an integral part of the problem solving process.

## Types of Knowledge

In talking about knowledge representation, it must be recognized that there are different types of knowledge that must be represented. These different types of knowledge are described below:

- **Objects** - Objects are entities that can be reasoned about as a unit. Examples of objects include dog, chair, car, as well as facts such as 'the ball is red'. Although objects can still be made up of parts, the object can also be thought of as a complete unit. As an example, a dog has a heart. A heart is an object; however, a dog can be thought of as an entity without considering the heart. Thus, the relationships between objects must be accounted for when representing objects.
- **Events** - Events have a time component associated with them. Events can be thought of in terms of state changes in the environment. The event causes the change from state A to state B. The temporal nature of events must be accounted for in addition to the state transitions and the details of the actual event.
- **Performance** - Performance knowledge refers to a sequential act or a procedure for performing a task. There are planning aspects associated with performance knowledge when alternative actions can be performed to produce the desired result. There are also time considerations in performance knowledge in addition to the specific how-to knowledge.

- **Meta-knowledge** - Knowledge can be thought of as a hierarchy. The meta-knowledge refers to the system's knowledge about its knowledge base. This higher level knowledge is used to guide the reasoning or problem solving process. The domain of knowledge is a piece of meta-knowledge for a system.

This list should provide some insight into the problem of knowledge representation by highlighting the diverse types of information that must be accounted for within a system. Knowledge from each of these classifications is required to solve real world problems..

### Uses of Knowledge

In order to understand the knowledge representation problem, the ways that knowledge can be used must first be understood. There are three uses for knowledge: acquisition, retrieval, and reasoning.

To use knowledge, there first must be some knowledge within the system. Thus, the first step is knowledge acquisition. The process of knowledge acquisition can be as simple as prompting an operator for input or as complex as an automated acquisition process that can deduce new information from existing information. Part of the knowledge acquisition process involves formatting the information into the chosen representation. Updating of a knowledge base is a critical part of the acquisition process.

Once knowledge has been stored, it must be retrieved. The appropriateness of a representation is judged in part by how easy required information is to retrieve. As in conventional software systems, the type of access required must be considered in deciding upon the storage mechanism.

Finally, the knowledge must be used in a reasoning or problem solving process. As stated earlier, there is a close relationship between the problem solving process and the representation of the knowledge. The representation should facilitate the problem solving process. The purpose of having the knowledge available is to allow the system to reason about the knowledge in order to solve the problem.

### Types of Reasoning

There are several different methods of reasoning about knowledge. Some of the more common methods are listed below. The list represents many of the popular methods of reasoning.

- **Formal** - Formal reasoning is mathematical in nature. This reasoning is based in the well defined rules and laws of logical inference. Formal reasoning is used extensively in theorem proving programs. In addition, the Prolog language is based around this type of formal reasoning.

- **Procedural** - In procedural reasoning, there are specialized procedures that encode the reasoning process for specialized cases. Thus, the reasoning is specific to the problem.

- **Analogy** - In analogic reasoning, characteristics are inherited for an object from a similar object. As an example, we know that sparrows fly and that robins are like sparrows. Using analogic reasoning, we conclude that robins probably fly. Many psychologists feel that most human reasoning is analogic in nature. The MCC research into knowledge also centers around the use of analogic reasoning. The problem with analogic reasoning is the enormous amount of information that is required to form useful analogies. Thus, the problem of large knowledge bases appears again.

- **Generalization** - Generalization is related to analogy. In generalization, we deduce that if a certain significant number of members of a class display a given characteristic, then all members of that class probably will display that characteristic. As an example, robins have wings, blue jays have wings, sparrows have wings, therefore all birds probably have wings. As with the analogic reasoning, a vast store of knowledge is required to perform reasoning by generalization. In fact, an enormous amount of information is required to even begin to identify class membership and characteristics significant enough to warrant thinking about.

- **Meta-reasoning** - As with meta-knowledge, meta-reasoning concerns reasoning about the reasoning process and the problem domain. Knowledge about the domain of the world that the problem exists in and how problems are addressed in that domain, fall under the guise of meta-reasoning.

### Evaluating Alternative Knowledge Representations

To evaluate alternative representations, first the features of a good representation must be known. Next, the criteria for evaluating representations is discussed. Finally, the different classifications for representations must be understood to allow comparisons to be drawn.

#### Features of a Good Knowledge Representation

As stated previously, the knowledge representation chosen for a problem should facilitate the problem solving process. The method of solution should appear at least obvious, if not trivial, given the proper representation mechanism. If the representation interferes with the solution to a problem, then a new representation should be found.

The representation chosen for a problem should be understandable. Thus, an outsider should be able to see a reasonable connection between the problem and the representation (the representation should "make sense"). For example, for the image processing problem, if the data input is the digitized representation of a scene, a logical

representation might be one which represents each bit of the image in a field with some indicator of the intensity of the image in that area. The scene would then appear as a grid of these fields representing the image. This representation intuitively makes sense to an observer. It is likely that a logical representation will facilitate the problem solution; however, this is not always the case. Ease of solution is the overriding factor in this situation.

In addition, the representation chosen for a problem should expose the constraints involved in the problem. The knowledge that is important to the solution should be explicitly represented.

A representation should be computable. It should completely represent the problem, in a concise manner requiring as little processing resources as possible.

Finally, the knowledge representation should display the appropriate level of detail for the needs of the problem. Unimportant details for one stage may be critical at another stage. Thus, the details should be suppressed by the representation but not deleted. These alternate views may require different access methods into the representation, but the representation remains the same.

### Considerations for Knowledge Representation Selection

There are several aspects of the knowledge representation problem that must be considered when making a selection. As mentioned previously, the granularity of the data required must be considered. In addition, the scope of data required for the problem solution is important. Irrelevant information can hinder the problem solving process.

Indeterminancy is another aspect of knowledge representation. There are multiple ways to represent the same fact. For example, the statements "a bird has wings" and "wings are a part of a bird" state the same fact from two different perspectives. The representation and the problem solving process must be able to determine this equivalence. Related to this problem is the fact that there can be equivalent yet distinct representations for the same information. At the lowest level, all representations are the same since they all break down to bits in storage. However, different representations allow different types of processing to be performed. In certain situations, it might be advantageous to translate the knowledge base from one representation to another to allow for more efficient processing.

As with software programs, a representation should support modularity. Modifications to one portion of the knowledge base should only effect the portions of the knowledge base related to the altered fact.

As stated previously, there is explicit and implicit knowledge that must be considered in selecting a knowledge representation. It is critical to insure that the information that is important to the problem solution be explicit in the representation.

Uncertainty is an important factor in the problem solving process. When attempting to solve real world problems, statements that are certain or conclusions that are 100 percent correct are difficult to obtain. Instead, these problems require methods to handle "maybe" answers, or "I think this is the case". The area of uncertainty extends to both the reasoning itself as well as the facts. Methods of propagating these uncertainty ratings through the reasoning process must be considered.

Finally, there is the selection between procedural representations and declarative representations. The debate within the field of Artificial Intelligence still rages as to which representation is best. The distinction between these two classifications is described in the next section. The debate itself has advanced the level of interest and research in the knowledge representation area.

### Classification of Knowledge Representations

There are numerous methods of classifying representations. The classifications used here are broad categories only for the purpose of allowing comparisons between the different representations. Following is a list of four classifications for knowledge representations:

- Declarative structures provide distinct separation between the knowledge and the inference engine (the reasoning process). These representations allow a great deal of flexibility in the structure of the representation. Traditional tables are examples of a declarative structure

- Procedural representations encode some of the reasoning process in the knowledge base, allowing objects to know how to reason about themselves. This representation generally allows for a simpler reasoning process at the expense of the flexibility of the structure. This difference is the central issue in the debate between the proceduralists and the declarativists.

- Most expert systems use the production representation. This representation uses if-then rules to encode the knowledge. The inference engine then reasons through the rule base using either forward or backward reasoning. Forward reasoning proceeds from the data to a solution; backward reasoning proceeds from a hypothesis to see if the data supports the hypothesis.

- There are several specialized representations that can not easily be classified under one of these categories.

### Survey of Knowledge Representation Techniques

Following is a description of several popular representation techniques. This list is not exhaustive. However, the knowledge representations listed here represent a cross section of the techniques available for solving a variety of problems.

- **Semantic Nets** - Semantic nets are connected graphs containing nodes and arcs. The net shows relationships among related concepts. This technique was first used to store dictionary-like word definitions. The nodes of the graph represent the concepts while the arcs show the relationships between the nodes. Retrieval of nodes and retrieval of related nodes within the net are major issues.

- **State-Space Representation** - The state-space representation is generally a tree-like structure that contains an entry for each possible state that the system could be in. This representation is popular in game-playing programs where each entry represents a board position. The transitions from one entry to the next represent the legal moves. Theoretically, this representation contains all possible states. Reasoning for this representation generally takes the form of a search of the tree to determine the optimal path. Much of the early work in Artificial Intelligence centered around optimizing search strategies for just this purpose. Heuristics are used to control and limit the search.

- **Semantic Primitives** - Semantic Primitives have historically been used in the natural language area of Artificial Intelligence. Essentially, semantic primitives represent atomic ideas by translating verbs to their basic meaning. Sample primitives include the following: move, ingest, transfer-ownership, etc. This representation is used to store the basic meaning of a construct.

- **Case Grammars** - Case grammars are similar to the semantic primitives mentioned previously. Case grammars associate the verbs with the objects normally required to properly interpret the sentence. For example, with the verb "throw", the following items are generally expected to be referenced either explicitly or implicitly: the object being thrown, the person (generally) doing the throwing, possibly a person to catch the object, the original location of the object, the ultimate location of the object, and perhaps the path the object took. Thus, in analyzing a sentence containing the verb to throw, the system has some knowledge of what types of objects might be in the sentence.

- **Frames** - Frame representations are generating a great deal of interest. Frames are an extension to case grammars in some respects. With frames, there are slots in the representation to be filled with information. The slots in general are predefined to give some knowledge of what should be present. There can be procedures associated with slots that specify how the information is to be obtained if the information in that slot is required. In addition, frames generally support a default value for a slot, as well as perspectives to allow different default values in different situations. Finally, an important feature of frames is the inheritance of slot values from superordinate frames. A hierarchical relationship can be defined in frames using the is-a link. Then, specification of a value for the superordinate frame allows the inheritance of that value to the subordinate frames. As an example, there exists a chair, called chair1. The frame for chair specifies a default value for number of legs of four. In addition, the slot for composition specifies a default of wood, with an additional perspective of royal chairs with a composition default of brass. The frame for chair1 specifies an is-a link to the chair frame. Thus, by default, the system determines that chair1 has 4 legs. This default must be overridden by an explicit value in the chair1 frame if the chair happens to have three legs (ie a stool). Frames are being used extensively in research into the storage of commonsense knowledge.

- **Scripts** - Scripts are similar to frames in that there is a predefined set of information. However, scripts incorporate time by specifying a sequence of events that occur for an event. The classic example of a script is the restaurant. If the system knows that the bill is being paid, the script provides the information that food has probably been served and eaten. Scripts allow for different contexts as well. For example, when a person enters a restaurant and sees a plastic counter, lines of people waiting at registers, and plastic tables, the assumption is made that this is a fast food restaurant. The expected sequence of events is different here than at an exclusive restaurant where the host approaches the patron, in a tuxedo, and seats the party at a table covered with a lace tablecloth. Scripts allow the sequential actions in these various situations to be encoded to allow interpretation of information generated from these contexts.

- **Transition Networks** - Transition networks consists of nodes and arcs. The arcs represent the transitions from states, represented by the nodes. The network has explicit start and end states. A variation of transition networks, called Augmented Transition Networks (ATNs), are used extensively for sentence parsing. The ATNs differ from traditional transition networks in that conditional processing is allowed on the arcs. In addition, side effects, such as structure building, are included which allows the network to output not just valid/invalid but to output the parse tree as well.

- **Procedural Representation** - As stated earlier, procedural representations encode knowledge of how to reason about an object in the representation for the object. The classic example of a procedural representation is the SHRDLU Project (also referred to as Blocksworld). SHRDLU was built as a natural language processing system. The domain of SHRDLU was a world consisting of a room with blocks of different shapes, sizes, and colors and a robot that could move the blocks around. The blocks had information

about their characteristics. In addition, the sequence of actions required to perform a request were encoded in the representation for the request. The SHRDLU system was quite proficient in its limited domain. It should be noted that the use of a restricted domain for prototyping has become a popular research tool in Artificial Intelligence.

- **Logic** - Logic representations encode all information in terms of facts and rules. The facts can then be reasoned about using the rules and the laws of legal inference. The Prolog language is built around the use of the Logic representation. Many of these systems are based on first order predicate calculus or propositional calculus. Much work is going on in determining the use of the logic representation with relational databases.

- **Production Systems** - Expert systems generally use the production knowledge representation. The knowledge is encoded as IF-THEN rules. Reasoning progresses based on the ability to satisfy the rules in the knowledge base. Due to the preponderance of information on expert systems, this topic will not be addressed further.

- **Direct Representations** - These representations are useful in circumstances where the data to be processed is already available in a reasonable representation. An example of this is the use of a grid for image processing. When dealing with a digitized image of a picture, the image is generally stored with some indication of each pixel's intensity. The grid representation is a direct representation for the digitized image. No additional structure or other information is added to the representation to facilitate processing.

- **Special Purpose Representations** - In some cases, combinations of the above techniques are required for an applications. Other cases actually require multiple representations for various stages of the analysis. The problem of speech recognition is generally addressed using multiple representations of the speech input. Frames are often combined in networks or with other representations to augment their capabilities.

This list should provide an overview of the numerous techniques available for solving a wide range of problems.

### Current Problems in Knowledge Representation

Several problems are actively being researched in the knowledge representation area. Work in these areas is performed not only by Artificial Intelligence researches but also by cognitive scientists.

The problem of encoding commonsense knowledge, also referred to as world knowledge, is under active research. Many people feel that the difficult problems of Artificial Intelligence can not be addressed until the computer systems can use commonsense in solving a problem.

Related to the commonsense knowledge problem is the problem of large knowledge bases. As the amount of knowledge available to a system grows, the access methods must be improved to allow the information to be retrieved and used in the reasoning process. This work relates to that in the traditional database field as well as that dealing with large databases.

Context information is used to understand the situation at hand. An understanding of how to represent context as well as knowing when to switch contexts is another important area of research.

Many people are investigating the nature of representation within the human brain and how people represent and retrieve the vast amount of knowledge that is applied to solving everyday problems. Research into this area is also being extended to trying to actually simulate the brain in hardware.

### Conclusion

This paper attempted to give an overview of the numerous representation alternatives available to address problems in Artificial Intelligence. The features of a good representation were presented to give some criteria for evaluating competing representations. Some directions of further work in the area of knowledge representation were presented.

For further study in the knowledge representation area, refer to any introductory Artificial Intelligence text. Most AI texts give an overview of knowledge representation but include the author's bias to his favorite representation. Frames and scripts are treated extensively in Schank.

The area of knowledge representation offers numerous opportunities for study. It is important to recognize that expert systems are not the only approach to problem solving in Artificial Intelligence. While expert systems are a valuable tool, many problem sets do not fit well into that paradigm. Many other possibilities exist.

### References

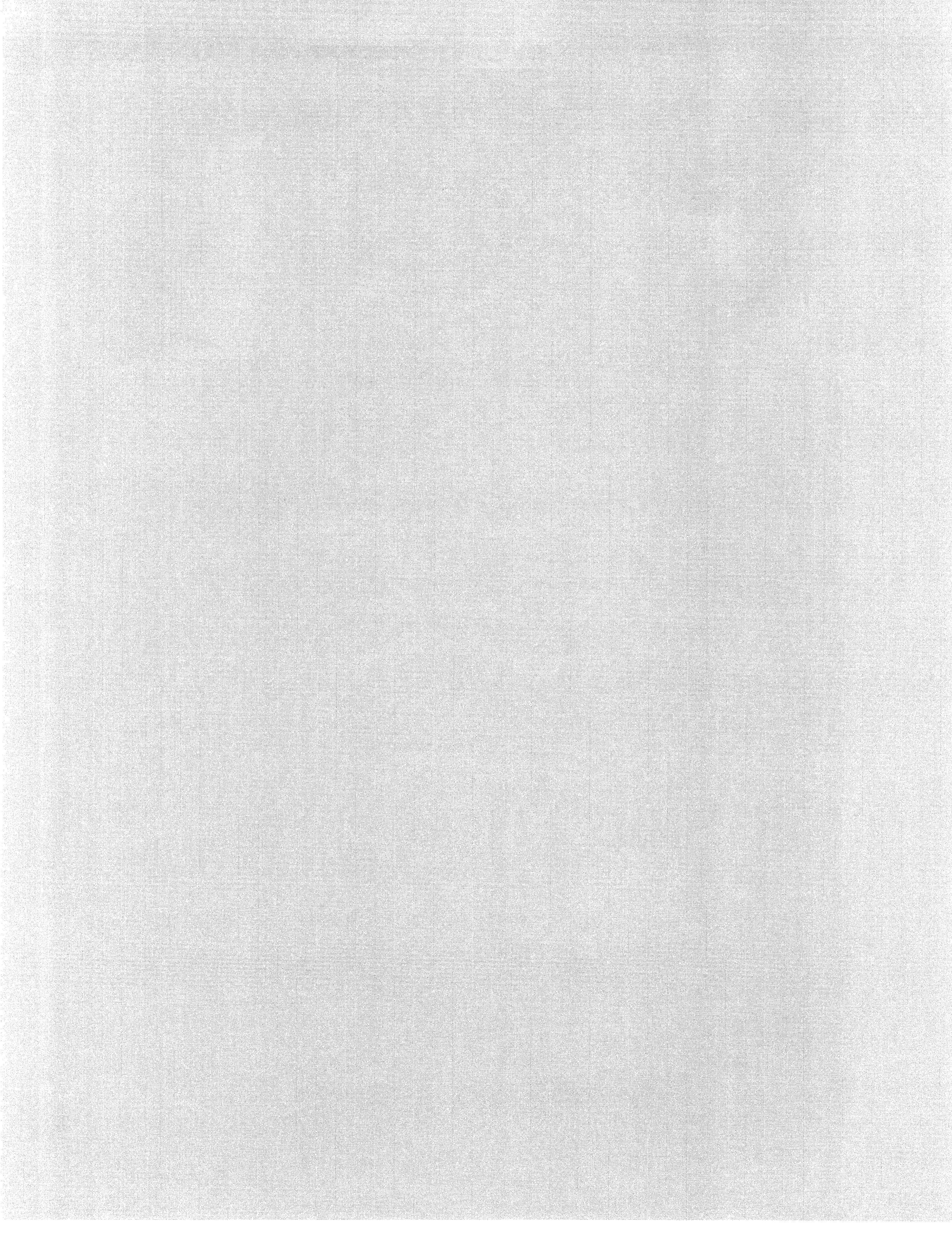

[1] Bass, Avron, and Feigenbaum, Edward; Handbook of Artificial Intelligence - Volume 1.

[2] Schank, Roger and Childers, Peter; The Cognitive Computer.

[3] Winston, Patrick; Artificial Intelligence.

[4] Charniak, Eugene, and McDermott, Drew; Introduction to Artificial Intelligence.

# BUSINESS APPLICATIONS SIG



# Applications Software Design for the Multi-lingual Environment

**Paul Mistretta** and **Phil Racine**
The Fasbe Group
Lowell, Massachusetts

**Abstract**

In an environment where distributing processing loads may involve maintaining a network of VAXes using DECNET, it is becoming increasingly important to retain control and auditability over these systems at a minimal cost. For a large corporation, these systems may span international boundaries. From this environment arises the need for an application that can use the same set of executable images regardless of local language requirements.

Systems are not considered friendly if frozen in American language. A business application must communicate to its' users in their native language. In a multi-lingual environment, the methodology employed by the system to accomplish this is conducive to the successful support and on-going maintenance of the system. What design considerations exist for applications software that be translated without modifying the software? How can one have the same set of programs running at several different installations, each with a different native language ?

This program session will present the areas of Application Software Design that are pertinent to the Multi-lingual environment. Some of the design elements covered will be: Design, Translation, Auditability, Consolidation, and Maintenance routines.

## Multi-lingual Applications Environment

### Anticipated Market for the System

With the recent technology developed by DIGITAL, there is an increasing variety of computing power being used throughout the world. The combination of hardware development advancements and some fairly significant cost reductions have brought more computing power into applications users' hands. At the forefront of this technology are distributed data processing networks. The total processing for a corporation may be spread across many physical installations. Depending on the nature of the system, some installations within a system may span international boundaries.

Some data processing organizations are finding it worthwhile to distribute the systems throughout the corporation. This places computing power directly into the users' hands who can make best use of it for the company. This can mean some good things and some bad things for the DP shop. Among the positive aspects are a reduced load on the hardware for the central site. It is not necessary to have one big machine that supports all users throughout the company. This also can reduce communication costs, because you don't have users accessing the system by telephone lines. This means the main system is less burdened, and can concentrate on servicing the main site.

It can make a lot of sense to distribute computing resources throughout an organization. Depending on a company's business concepts, decentralization can be a desirable way to go. From a technological standpoint, there are distinct advantages, some of which can be more effective use of computing resources, and increased productivity for the application end users. However, from this distribution arises the need to support and control the various installations in the company. It is especially important for an organization to have financial systems possessing integrity and auditability. It is probably safe to say that a company would not want several different financial systems to be running within a successful organization.

The United States has recently enjoyed a period of reduced inflation and increased business activity. A major portion of current and potential business has developed in the overseas market. Traditionally, Western Europe is one of the major sources of business for American corporations. With this international business, it is quite conceivable that a DP organization may need to support computer installations overseas. A company may have these installations in countries with different language requirements. The concept of the multi-lingual environment arises from the approach of using distributed systems in an interna-

tional environment. It can also be applied to an application that services sites with different language requirements that are not necessarily part of the same system.

In selecting the computer systems to support these sites, it is important to select the proper hardware and systems software to support the desired functions. Within a distributed system, the sites will want to select a computer that can provide resources to their users adequately and efficiently. The size of the computer should be appropriate for the size of the installation. So, within a distributed system, you may find large computers at the main sites, medium-range computers at the intermediate sites, and smaller computers, at the smaller sites.

If an organization chooses to use a configuration with different operating systems, there is more work involved in controlling the software at the user sites. However, if an organization chooses systems that use the same operating system software at each site, some advantages start to arise. With one set of system software, it is now possible to run the same application software at each site This allows for greater integrity and control of the sites within the organization. This becomes important for the organization that wants to decentralize their computer operations, but while doing so, maintain control and integrity over the system. Once the system software is set, the environment for the application software can be defined.

A good deal of an application systems' worth is measured by how easy it is to use. This can be directly related to the productivity of the end users who use the system. Most of the financial applications software that currently exists is written in the United States for Americans. This constraint makes it more difficult for foreign installations to use a system that does not speak in their native language. Granted, most installations are probably fluent in English in some form or another. But, wouldn't a computer system, especially an important application, be more user friendly, more productive, and more valuable if it could communicate with the users in their own language ?

The concept of being able to customize the user interface for an application can be very intriguing. It can obviously be done by programmers changing the software, but how do you support multiple versions of a application ? From a support standpoint, it can be very attractive to have an application in which the user interface text can be translated without changing the software.

### Computing Needs of an International Corporation

Within a DP organization that services multiple sites, it is highly desirable for the systems to be supportable with a reasonable amount of resources. If there are multiple versions of a system to support different languages, this can become very cumbersome to manage. One set of executable images that can support multiple languages is a very attractive solution. Wouldn't it be nice to have software that allows the user to translate it ?

There are many activities related to supporting the users of multiple sites in a DP organization. These include satisfying users' requests, solving complex application problems, and implementing software modifications. This can become an excruciatingly difficult nightmare if there are many versions of a particular application. This is obviously a situation that one wants to avoid.

The integrity and auditability of the system is that much greater if one set of executable images can service each installation. A successful company relies heavily on accurate and timely financial information. If the same set of executable images is used at each site, the operations, maintenance, and support resources can be optimized. The consolidation of financial data should be much smoother. Software modifications can be implemented once to a commonly-used set of programs. The technical staff can concentrate their skills on one system, thus becoming more proficient with that application. The support network can be much smoother because they're only dealing with one set of application programs, rather than multiple versions.

Every successful organization wants to optimize the utilization of resources. A system that uses one set of executable images makes this possible.

### Language Independent Applications Software

With the objective of having one set of executable images servicing installations with many different languages, there are some inherent characteristics present in these programs. First and foremost is a strict requirement against using hard-coded text for user interface. User interface includes screens, messages, report headings, and control options. All user interface text must be fully translatable. There should be no software modifications required to handle a supported translation. In addition to translating the user interface text, an application should allow the date format and decimal presentation to be modified, to provide optimal service to a multi-lingual site.

These characteristics define the requirements for multi-lingual application software. The design information that follows is the implementation of the multi-lingual design into the applications of The Fasbe Group. We feel it incorporates the important concepts of a multi-lingual system.

### Characteristics of a Multi-lingual Financial Application

Our application is built with a layered architecture. This employs different levels of software for the system functions. The Application layer is used to communicate with the end user. Combined with the Algorithm layer, it contains the processing logic for the application system. These levels are purely hardware and operating system independent. To provide the translated user interface text to the application layer, a layer of subroutines are used. These subroutines access data files to extract the user interface text, and return it to the calling program. One

subroutine exists for each kind of text: (Messages, Report Headings, Control Options, Date Format, and Decimal Format). The data entry screens are translatable through SCOPE, our screen management package. SCOPE employs a source file for the screen definition. This source file can be translated through a on-line program.

The messages, report headings, and control options were typically hard-coded as literal values in programs. These text items are referred to symbolically in a multilingual system. A unique identification code is assigned to each message, report heading, and control option. At run time, the multi-lingual subroutine layer uses this key to retrieve the message or report heading. All text is fully translatable, and the symbolic referencing technique allows one set of executable images to service any supported translation. All data fields that store system indicators (Y/N, etc.) always store an American value. This will reduce the programming changes to implement the multilingual design into existing applications.

Within the application, there are four categories of translatable user interface text: Messages, Report Headings, Screens, and Control Options. The messages include error, instructional, informational, or prompting text sentences to the user. This is any message not included as part of a data entry screen. Messages are stored on an indexed data file and referenced by a symbolic key. Report Headings are classified by program, and are also stored on an indexed data file. The subroutines retrieve and/or display the messages and report headings from the file at run-time. Data entry screens consist of prompts and help messages that can be translated. Each screen translation is stored in separate source and executable form files. The Control Options are stored in a data file and accessed at run-time by a subroutine.

## Multi-lingual Application System Concepts

### Multi-lingual Utilities

Several on-line programs for the messages and report headings were developed to establish the records in the file that store the text. There is a set of file maintenance, translation, report, send, insert, and integrity check programs for both the messages and report headings. The file maintenance and translation programs establish the text records on the file. Once a file has been established, the send and insert programs are used to update a file with modified text. The report and integrity check programs help to verify the contents of the text files and provide a level of integrity and control for the system.

There are on-line translation programs for the data entry screens, and control options. A global definition exists for the date and decimal formats. These are used by the date and decimal formatting subroutines which process those data items at data entry time and output time.

#### *Message Utilities*

A file maintenance program (add, change, delete) is initially used to set up the message text records. This initially establishes the message records with American text on the data file. This program is used by the system developers to establish the text records on the file. Messages can be Fatal, Warning, or Informational type. This is specified in the Severity field. When a program needs a message, it is added to the message file. A certain amount of care should be taken here to control the amount of messages in the file. In order to provide a smoother translation, it is desirable not to have many variations of the same message. When possible, it is preferable to use an existing message on the file, rather than add a new one.

The message key structure is designed to provide a unique and symbolic reference for each message. It also serves as a run-time guide to the message subroutine for loading frequently-used messages into a memory table.

It consists of three parts:

The first part is a six-character code that indicates how the message is used. A message can be program-specific, subsystem-specific, or a system-wide type. The second part is a three-character code, that indicates the type of message. There are Informational, Invalid, Help, ASK=Prompt, OPT=Prompt for options, FIL=File Messages. The remainder of the key is a programmer-defined 21 character area that is used to describe the message itself. For file lookup and storage efficiency, it is important to design a key that is technically efficient, and symbolically meaningful.

To translate the messages, an on-line translation program allows the user to enter the translated text for the messages. The user can select between sequential and random translation mode. In sequential mode, the program displays the next message to be translated. In random mode, the user can enter the key of the message to translate. The translated text is then stored on the data file, along with the American text.

At run-time, the message subroutine is employed by the application layer to read the messages from the data file. To optimize the retrieval and display of frequently-used messages, a run-time table is employed to store the messages in memory, from which they are later accessed. In this day of inexpensive memory, it is desirable to improve application performance by storing messages in memory, so the access is quicker than it would be if they were stored on disk. As a general rule, all program-specific messages are loaded into the memory table when a program starts. This is limited by a number of 100 to not significantly degrade program initialization. The first part of the message key allows for a initial search, (START) and then sequential reads for each message in that program category. Only the informational and warning messages are loaded into the table. The fatal messages will always remain on the file, because they are used when the program is ready to terminate, so the extra retrieval time is insignificant. When it comes time to display a message,

the subroutine logic goes to the table or the data file to get a message.

A good side-product of collecting all messages for a system in the report is that it provides a format for reviewing all the messages together. This gives the user the opportunity to utilize the terminology of the business in their application system. One can also ensure a consistent means of communicating the messages from the system to the user.

*Report Heading Utilities*

A file maintenance program (add, change, delete) is initially used to set up the report heading text records. This initially establishes the report heading records with American text on the data file.

The key to the Report Headings record consists of two parts: The first part is a six-character ID which identifies the program that uses the heading. The second part is a sequential number that is incremented for each heading within a program set. The headings for a program are retrieved from disk when needed.

The sequential and random translation modes are supported with this function. The sequential translation mode will automatically display the next report heading that is eligible to be translated.

To translate the report headings, an on-line translation program allows the user to enter the translated report headings. The translated text is then stored on the data file, along with the American text. When translating the report headings, the user must be aware of the format of the report detail lines. Some care must be taken so the column headings line up with the data lines. The column numbers are displayed on the screen to assist the translator in lining up the detail line columns with the titles.

The data record layout for the text records contains both the American and translated text. This stores the translated and un-translated text side-by-side on the record. This allows the subroutine to choose which text is to be used at run-time. There is a switch on the user record that controls whether he looks at the translated or un-translated text record.

Reports are available to print the messages on the system. The report can be sorted by message key, message type, or the actual text of the message. The user can select between translated and un-translated text. The report headings are sorted by program, and the user can select to print headings for a particular report. The user can choose between un-translated and translated text for the report headings also.

*Screen Utilities*

The screen management utility, SCOPE provides an ESSAY source file that is compiled into an executable .FRM file which is then used by the system at run-time. The source file containing the screen definition is translated with an on-line program, which writes a new source file with the translated text. This source file was one of the primary characteristics of SCOPE that was of great value to the design. Some other useful features of SCOPE for the multi-lingual design are the ability to specify the decimal presentation format as an installation parameter, and the ability to define a field as being of date type.

The user can translate screens by subsystem. Within a subsystem, the user can choose to translate screens in sequential mode, or at random by entering the screen name. The screen names for each subsystem are stored on a data file, which is read in by the program at run-time. In sequential mode, the program brings up the screens to be translated one at a time. In random mode, the screen names for a subsystem are displayed for the user to select. The program allows the user to translate prompts and help messages, which are then stored on the new source file. The date format is automatically converted to the global system format. When translating prompts, the user is restricted to working within the current length of the American text. The SCOPE utility can be used to further manipulate the prompts and fields if desired, but technical experience is required to do this. The translations for help messages can use the full 80 characters available.

This source file is then compiled into the executable file. With the process logicals and multiple directories that are available on the VAX, the system can save multiple sets of form files, each of which can contain a different translation.

*Control Option Utilities*

Control options are verbs that the user enters to control functions of the application software. At the bottom of the data entry screens, the user is prompted for an option that will guide the next action for the program to take. These verbs were identified during conversion, and placed on a data file. There is an on-line translation program which allows the user to translate these verbs. The translation is placed on the file with the American text, using the 'side-by-side' technique, allowing the language variation at run-time.

There is one rule that must be followed during translation of the options. Basically, all options within a group must start with a unique first character. The application prompts for one character options, so there must be a unique first character at each system prompt.

Within the system, there are many option relationships which make up the different groups. These relationships are built into the translation program, which guides the user through the translation process.

At run time, the options are loaded into a memory table from the data file during program initialization. This establishes the text that is used for both display and to check data entry responses. It was a design objective to use the same data for both purposes. This ensures that the system will always display the same text as the application program is checking for.

*Send/Receive Utilities*

Once the system is up and running at an installation, the users' translations are stored on the message file used with the application. As software modifications are implemented, there also changes and additions to the message and report headings file. In order to update existing sites that have translated text, these utilities were created. They allow the files to be updated with only those text records that are new or have changed. There are indicators on the text records indicating that they are eligible for translation and to update an existing site. The SEND program extracts text records that are eligible to be sent from the file, and builds a transmit file to be sent to the site (by tape or DECNET). The send indicator on the record controls which messages/report headings to be sent. Once the file has arrived at the site, the RECEIVE program will update an existing file with those records that are new or have changed. After the records are in the file, they are marked as being eligible for translation. This causes the translation programs to display them in sequential mode. Any message or report heading can be translated in random mode whenever the user desires. This accomplishes the updating of messages and report headings to an existing site when required for software updates.

*Integrity Check Utilities*

After the application has been installed at a site, it is conceivable that some problems may arise. It becomes desirable to eliminate the message and report heading file as a possible source of the problem. You would not want the system to fail based on a missing message or heading. This utility will compare the file from the site against a known good file, and report on any differences that are found. This includes: A text record missing from either file, and a count of text records on each file. The send/insert and integrity check utilities provide the central DP site with a means to control the integrity of the message/report heading files at the satellite installations.

## Conversion Approach

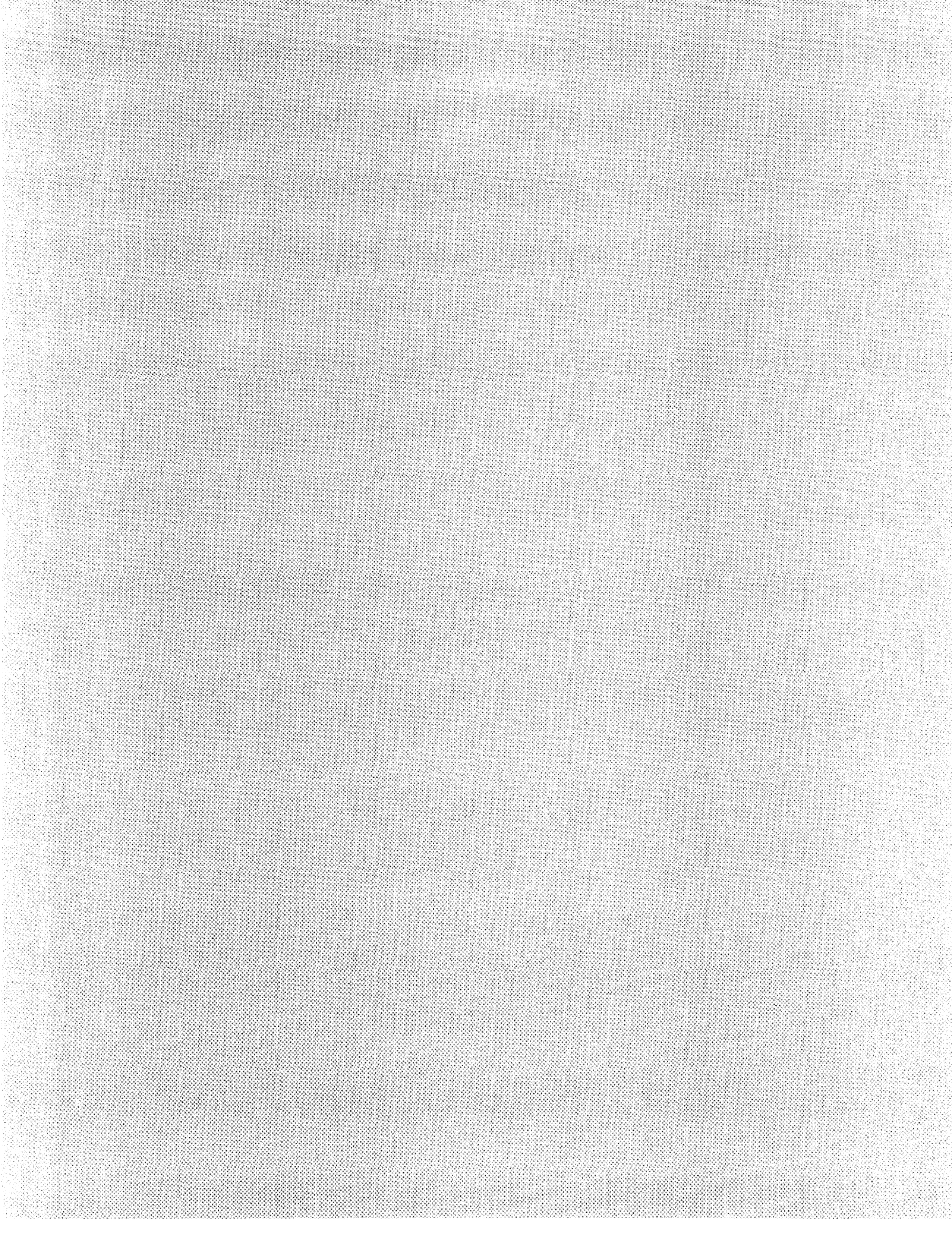
During our development project, we started with an existing application, and modified it to support the multilingual environment. This involved a series of steps to attack the problem. First, the utilities to maintain and translate the text files were developed. This was followed by the development of the run-time subroutines to support the application layer. This makes up the supporting mechanism for an application program.

Each program was reviewed to locate the hard-coded text and literals that were employed as part of the interface. These were each given a symbolic ID, and added to the text files with the file maintenance programs. Any output field that includes a date or decimal had to be modified to use a common routine. This routine arranged the date/decimal format per the global system definition. The program would then process this field as required. This modular approach localizes the multi-lingual dependent routines in one place. The conversion of three applications (G/L, A/P, and Security) was accomplished by two programmers in 4 months. Our system is not flawless, but we feel that it is capable of supporting any translation that reads left to right and uses a VT220 keyboard. We hope to receive feedback from the translators who use the system, so that we can make it a better system in the future. As the system matures, we will recognize the areas to improve and make adjustments accordingly.

The VMS features provided us with a superior development environment in which to accomplish this project. Without much experience with other vendors' machines, it is difficult to relate this design to other systems. However, we can definitely say that the VAX/VMS development environment was a key part of our design and development effort. This proves that it is possible and feasible to develop, maintain, and support multi-lingual applications software.

The credit for the multi-lingual design concepts should go to Gary Neidhardt, the Director of Software Development at The Fasbe Group. Upon being hired by the company, it was our charter to implement the multilingual design into some existing applications for a Fortune 250 company. This becomes a feasible and economic approach for a software development organization to develop and maintain applications software for the multi-lingual environment.

# COMMERCIAL LANGUAGES SIG



# GUIDELINES FOR WRITING VAX COBOL PROGRAM GENERATORS

RAY DAVIS
Sheffield Steel Corporation
Sand Springs, Oklahoma

## AN EXAMINATION OF METHODOLOGIES AND HELPFUL HINTS FOR THE DEVELOPMENT OF CUSTOMIZED COBOL SOURCE CODE GENERATORS

A recent COMPUTERWORLD article explored the re-imergence of COBOL as the language of choice in many data centers across the United States. Many of these shops, after trying to implement 4GL(4th Generation Language) packages, found that the overheads of the package were too great or the ability of the package to create the complex code the users needed was lacking, Often, the packages that possessed the ability to create the complex procedures resulted in very long spagetti code modules that were extremely difficult to maintain, or close to impossible for their average user to develope. The article went on to develope the hypothesis that COBOL can remain a very cost effective tool with the growth of new code generation tools.

Numerous manhours are expended each year writing and maintaining very large and complex COBOL programs. With COBOL still a heavily used business programming language, considerable time is used creating Working Storage modules, I-O Sections, Etc., the general 'housekeeping' of COBOL. While there are numerous program generators available for creating standard maintenance programs, inquiry routines, print jobs or programs similiar to these, many companies don't like to expend the dollars to acquire something that does not have that 'customization' they have grown to love or the code that the generator creates is not to their own standards and would be difficult to maintain. Often, these third party generators, trying to cater to the needs of many, are so 'soft' that operation of the generator sometimes takes more time than actually coding the program by hand or the code produced is so generic that variable names all begin to look alike. To acquire the highly customized and yet have the benefits that a generator provides may indicate that the company should examine the possibilities of writing its own source code generator.

A custom written program generator can become an enormous asset in any shop. It creates programs that have similiar logic flows, thus simplifying documentation and maintenance. Perhaps more importantly, a good program generator can help implement programming standards. This highly structured, standardized code can greatly reduce the maintenance overheads. A generator typically creates these programs over a short timeframe, and can be used as the basic skeletal structure for much more complicated programs. There are several basic steps that can be utilized to create your own program generator, and their simplicity belays the mistique that seems to surround program generators. These steps can be implemented in many programming languages but for example purposes, we will use a generator producing a VAX COBOL isam (Index Sequential) maintenance program.

### Step 1: Design the Ultimate Maintenance Program

The most important part of any program generator is the final product. In that our final product in this example is an isam file maintenance program,we should create a standard maintenance program that includes all of the bells and whistles that the company requires in a program of this nature. This program should be staff reviewed, letting everyone's input contribute to the final product, thus insuring that this program has everything the company would want in a standard maintenance program. Remember, the basic code generated in this program will be replicated time and time again, so an error-free program that adheres to company standards is very important. Once this basic maintenance program is completed, we can proceed to the first design stages of our generator. For our example, a basic program we want to generate is shown below:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. GL03P030.
AUTHOR. RAY DAVIS.
INSTALLATION. SHEFFIELD.
DATE-WRITTEN. 02/03/86.
DATE-COMPILED.
SECURITY.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. VAX-11.
OBJECT-COMPUTER. VAX-11.

SPECIAL-NAMES.
  SYMBOLIC CHARACTERS
    CR-VAL  CTRL-Z-VAL  CTRL-N-VAL CTRL-O-VAL ESC-V
  ARE 14        18          15          16       28.

INPUT-OUTPUT SECTION.

FILE-CONTROL.
  SELECT GL03
    ASSIGN TO GL03
    ORGANIZATION IS INDEXED
    ACCESS MODE IS DYNAMIC
    RECORD KEY IS GL03-VOUCHER.

DATA DIVISION.

FILE SECTION.

FD GL03.
01 GL03-RECORD.
   03 GL03-VOUCHER                PIC X(3).
   03 GL03-STD-VOU-FLAG           PIC X.
   03 GL03-VOUCHER-DESCRIPTION    PIC X(30).
   03 GL03-VOUCHER-EXPLANATION    PIC X(30).

WORKING-STORAGE SECTION.
77 WS-COMMENT         PIC X(9) VALUE "WS-BEGINS".
```

```
77 SWITCH           PIC XXX     VALUE "OFF".
77 SELECTION        PIC 999.
77 ANS              PIC X       VALUE SPACES.
77 RCD-SW           PIC X       VALUE "X".
77 LINE23           PIC X(79)   VALUE SPACES.
77 LINE24           PIC X(79)   VALUE
   "PF1-BACKUP PF2-UPDATE PF3-DELETE PF4-EXIT".

01 CONTROL-KEY.
   03 FIRST-CHAR-CONTROL-KEY  PIC X.
      88 ESC                  VALUE ESC-V.
      88 CR                   VALUE CR-VAL.
      88 CTRL-Z               VALUE CTRL-Z-VAL.
      88 CTRL-N               VALUE CTRL-N-VAL.
      88 CTRL-O               VALUE CTRL-O-VAL.
   03 REMAINDER-KEY           PIC X(4).
      88 PF1                  VALUE "P".
      88 PF2                  VALUE "Q".
      88 PF3                  VALUE "R".
      88 PF4                  VALUE "S".

PROCEDURE DIVISION.
MAIN SECTION.
MAIN-1.
   OPEN I-O GL03.
   PERFORM DRIVER UNTIL
     SWITCH = "EOJ".
   CLOSE GL03.
   DISPLAY " " ERASE SCREEN.
   STOP RUN.
MAIN-EXIT.
   EXIT.

DRIVER SECTION.
DRIVER-1.
   PERFORM DISP-SCR.
   PERFORM GET-KEYS.

   IF SWITCH = "MOD"
      PERFORM MOD-DATA UNTIL SWITCH = "XXX"
      GO TO DRIVER-EXIT.
   IF SWITCH = "ADD"
      PERFORM KEYIN
      PERFORM ADD-RECORD
      GO TO DRIVER-EXIT.
DRIVER-EXIT.
   EXIT.

GET-KEYS SECTION.
GET-KEYS-1.

   DISPLAY "VOUCHER" LINE 3 COLUMN 1 ERASE LINE.
   ACCEPT GL03-VOUCHER KEY IN CONTROL-KEY
      UNDERLINED LINE 3 COLUMN 40.

   IF PF4 MOVE "EOJ" TO SWITCH
      GO TO GET-KEYS-EXIT.

   READ GL03 INVALID KEY
      GO TO GET-KEYS-2.
   MOVE "MOD" TO SWITCH
   PERFORM DISP-DATA
   GO TO GET-KEYS-EXIT.
GET-KEYS-2.
   DISPLAY "ADD THIS RECORD?" LINE 23 COLUMN 1
     ERASE LINE.
   ACCEPT ANS LINE 23 COLUMN 40 UNDERLINED.

   IF ANS = "Y"
     MOVE "ADD" TO SWITCH
     GO TO GET-KEYS-EXIT.
   MOVE "XXX" TO SWITCH.
GET-KEYS-EXIT.
   EXIT.
MOD-DATA SECTION.
MOD-DATA-1.
   DISPLAY "ENTER SELECTION:" LINE 23 COLUMN 1
     ERASE LINE.
   ACCEPT SELECTION KEY IN CONTROL-KEY WITH
     CONVERSION LINE 23 COLUMN 40 UNDERLINED.
   IF PF1 MOVE "XXX" TO SWITCH
     GO TO MOD-DATA-EXIT.
   IF PF2 PERFORM MODIFY-RECORD
     MOVE "XXX" TO SWITCH
     GO TO MOD-DATA-EXIT.
   IF PF3 PERFORM DELETE-RECORD
     MOVE "XXX" TO SWITCH
     GO TO MOD-DATA-EXIT.
   IF PF4 GO TO MOD-DATA-EXIT.
   IF SELECTION = ZERO
     PERFORM DISP-SCR
     PERFORM DISP-DATA.
   IF SELECTION = 1 PERFORM K001.
   IF SELECTION = 2 PERFORM K002.
   IF SELECTION = 3 PERFORM K003.
MOD-DATA-EXIT.
   EXIT.
KEYIN SECTION.
K001.
   ACCEPT GL03-STD-VOU-FLAG KEY IN CONTROL-KEY
     LINE 04 COLUMN 40 UNDERLINED PROTECTED.
   DISPLAY GL03-STD-VOU-FLAG LINE 4 COLUMN 40.
K002.
   ACCEPT GL03-VOUCHER-DESCRIPTION KEY IN CONTROL-KEY
     LINE 05 COLUMN 40 UNDERLINED PROTECTED.
   IF PF1 GO TO K001.
   DISPLAY GL03-VOUCHER-DESCRIPTION LINE 5 COLUMN 40.
K003.
   ACCEPT GL03-VOUCHER-EXPLANATION KEY IN CONTROL-KEY
     LINE 06 COLUMN 40 UNDERLINED PROTECTED.
   IF PF1 GO TO K002.
   DISPLAY GL03-VOUCHER-EXPLANATION LINE 6 COLUMN 40.
KEYIN-EXIT.
   EXIT.
DISP-SCR SECTION.
DISP-SCR-1.
   DISPLAY "SHEFFIELD STEEL CORPORATION"
     LINE 1 COLUMN 30 ERASE SCREEN.
   DISPLAY "G/L VOUCHER DEFINITION MASTER"
     LINE 2 COLUMN 30.
   DISPLAY "1.STD VOU.FLAG:" LINE 04 COLUMN 1.
   DISPLAY "2.VOU.DESCRIPTION:" LINE 05 COLUMN 1.
   DISPLAY "3.VOU.EXPLANATION:" LINE 06 COLUMN 1.
   DISPLAY LINE24 LINE 23 COLUMN 1 UNDERLINED.
DISP-SCR-EXIT.
   EXIT.
DISP-DATA SECTION.
DISP-DATA-1.
   DISPLAY GL03-STD-VOU-FLAG LINE 04 COLUMN 40.
   DISPLAY GL03-VOUCHER-DESCRIPTION LINE 05 COLUMN 40.
   DISPLAY GL03-VOUCHER-EXPLANATION LINE 06 COLUMN 40.
DISP-DATA-EXIT.
   EXIT.
ADD-RECORD SECTION.
ADD-RECORD-1.
   WRITE GL03-RECORD INVALID KEY
     PERFORM ERROR
     GO TO ADD-RECORD-EXIT.
ADD-RECORD-EXIT.
```

```
MODIFY-RECORD SECTION.
MODIFY-RECORD-1.
    REWRITE GL03-RECORD INVALID KEY
        PERFORM ERROR.
MODIFY-RECORD-EXIT.
    EXIT.
DELETE-RECORD SECTION.
DELETE-RECORD-1.
    DISPLAY "ARE YOU SURE" LINE 23 COLUMN 1
        ERASF LINE.
    ACCEPT ANS LINE 23 COLUMN 40 UNDERLINED.
    IF ANS EQUAL "Y" GO TO DELETE-RECORD-2.
    GO TO DELETE-RECORD-EXIT.
DELETE-RECORD-2.
    DELETE MASTER INVALID KEY
        PERFORM ERROR.
DELETE-RECORD-EXIT.
    EXIT.
ERROR SECTION.
ERROR-1.
    DISPLAY "ERROR-CONTACT-SYSTEMS" LINE 23
      COLUMN 1 ERASE LINE UNDERLINED.
    ACCEPT ANS LINE 23 COLUMN 40 UNDERLINED.
ERROR-EXIT.
    EXIT.
```

Step 2: Identifying The Variables

This next step consists of taking a listing of the maintenance program and going over it line by line, highliting any item that will change from program to program. Obvious items would be field names, data types, cursor positions, file names, etc. Not so obvious would be page number (what happens when your file definition exceeds the number of lines on the screen? Go to another page? Another Column?). Each variable that you identify may become an item that must be keyed into the generator when you run it. One situation that occurs in many shops is the use of include modules or dictionaries for data definitions. The program generator makes a good place to create these definitions because in the case of the maintenance program, you must key in the field names anyway.

Step 3: Coding the Data Entry for the Generator

Once you have identified all the variables within the program format you will generate, you should now divide these variables into two categories, re-occuring and non-reoccuring. The non-reoccuring variables (author, date, etc) should be captured in working storage while the reoccuring variables (variable names, file names, etc) should be stored in a file or array structure to allow repeat access. The information you capture is dependent on how sophisticated you want your generator to be. If you are using a generator product like CDD, you will probably want to capture additional field level information. COBOL field structures will require information to tell the generator, yes, I want this in the CDD but I do not want it as a keyin field in my program. Because a real easy way to identify structures in a COBOL file is the level number, it may be easiest to require the level number on each field name. Also remember that packed data fields require 'with conversion' on display and accept, so identification of the field structures is necessary.

Step 4: Creating the Generator From Data Entry

Now that the basic 'dictionary' for the generator is in place, the next step is to process this information back out in the form of a COBOL program. COBOL provides sufficient string manipulation so we chose to write the generator itself in COBOL. Some hints you may want to utilize when writing a generator in COBOL are:

A) Make extensive use of the new STRING verb
B) Have a SPACE7 and SPACE12 variable valued space defined in working storage for indentation in the generated program
C) Pre-define many of the section and paragraph names in working storage of the generator
D) Set up counters with good mnemonic names (PAGE-CTR,LINE-CTR,COLUMN-CTR)

The basic flow of a generator could be as follows:

1) Perform Data Entry
2) Create IDENTIFICATION,DATA & WORKING STORAGE sections with data captured in generator data entry (non-reoccuring)
3) Create MAIN and DRIVER sections with similiar non-reoccuring data
4) Process one pass thru the reoccuring data to create the KEYIN section
5) Process another pass thru the reoccuring data to create the DISPLAY-SCREEN section
6) Process another pass thru the reoccuring data to create the DISPLAY-DATA section
7) Create file I/O and error routines with the non-reoccuring data
8) EOJ

Obviously, by writing your first generator with a 'one screen per program' limit you will eliminate the need for tracking multiple screens. Once you have perfected the single screen program, you can expand to multiple screen generation by adding the code to track lines per page and pages per program. Other hints you may want to keep in mind when designing your generator are:

a) If your shop uses standard naming conventions, make use of these in your generator. For example, if a file defined as GL03.MAS has 'GL03-' preceeding each field name you need not enter that prefix for each field name but can have the generator do it for you
b) Use the generator to help enforce corporate standards. If your company uses PF4 as the exit key, write the generator in this manner and it will help evolve these standards throughout the system. Because a good generator will be used by programmers to develope highly customized programs (skeletal starting code) this has the effect of accelerating the implementation of standards.
c) Structure your generator as highly as possible. These custom written products have a tendancy to become 'enhanced' as the staff uses the product and finds new ways to add to the generator functionality.
d) While 100% program generation is great, many times a generator that can produce 50% of a program (ie; a print generator that creates the working storage code for heavily formated reports) will be utilized more than the generator that creates the entire product because it can be used by programmers to reduce the heavy

grind of coding these large sections of repetative code and still keep their 'style' intact.

Keep in mind when writing a generator that many programmers consider everything in a COBOL program above the Procedure Division as 'overhead'. Style, as it is known in programming is just not that important in these areas so a generator that will take away the time consuming task of coding these 'overhead' sections will probably be met with open arms. Any code created by a generator in the Procedure Division should be well documented and easily modified by the staff because in many cases, it will be modified anyway...

# DATA ACQUISITION, ANALYSIS, RESEARCH, AND CONTROL SIG

# MicroVAX Ancillary Control Process (ACP) for Realtime Human-Machine Interface

**Thomas Kane**
Tom Kane Computer Engineering
Glendale, AZ

## Abstract

This paper discusses the design and implementation of the human factors engineering of a Computer Aided Instruction (CAI) system. The unique demands of the CAI system were met by designing a custom VAX/VMS Ancillary Control Process (ACP). The ACP controlled and managed the graphics display as well as accepting mouse and keyboard input.

## Human Engineering

Since the system was intended to be used for training purposes, human engineering was considered a high priority. Many students would use the system for only a few weeks. For this reason an important design goal was ease of use and simplicity of operation. This goal proved to be harder than expected to achieve because some users (courseware authors and instructors) would use the machine regularly while students would use the system only for a limited time. In addition, opinions on human factors vary widely.

Colors for the menu display were chosen to enhance readability and to indicate the availability of functions. IN general, functions which are available are displayed in yellow. Functions which are not available are displayed in white or gray. It was decided to display unavailable functions (rather than eliminate them entirely) so that students would always see the same menu. In particular, it was determined that new users of the system might occasionally search the menu trees for a function that was previously enabled only to discover that the function was no longer available.

The following human factors goals were considered during the design of the system:

- consistent method of data entry
- minimal entry action
- minimal memory load
- flexibility

## Graphics Display

Figure 1 illustrates the areas of the graphics display screen. The graphics display screen is discussed in the following sections.

### Function Menu History Area

The function menu history area displays the previous menu choices which placed the user at the current position within the hierarchical menu tree. These items are never selectable, but provide a map for navigating the menu tree. The items are displayed as white text on a black background.

### Function Menu Area

The function menu area displays menu selections which are appropriate for currently active function. Available options are displayed using yellow text on a black background. Non-available options are displayed in gray. As the crosshair is moved past an available menu option, its color changes from yellow to green. Courseware authors determine which options are to be available at any given time. Non-available options are displayed in gray (rather than eliminated from the display) so that menu displays are always consistent. This prevents the situation where a student searches the menu tree for a function he just used, only to find that the option is currently not available.

### List Selection Area

The function menu area is also used to display lists from which the operator may select one item (such as directory lists or help topic lists). The colors used for display is identical to the function menu area.

### Utility Area

The utility area contains frequently used options which are desirable to display at all times. Examples are logoff and toggle communication area. Items are displayed using the same colors as the function menu area.

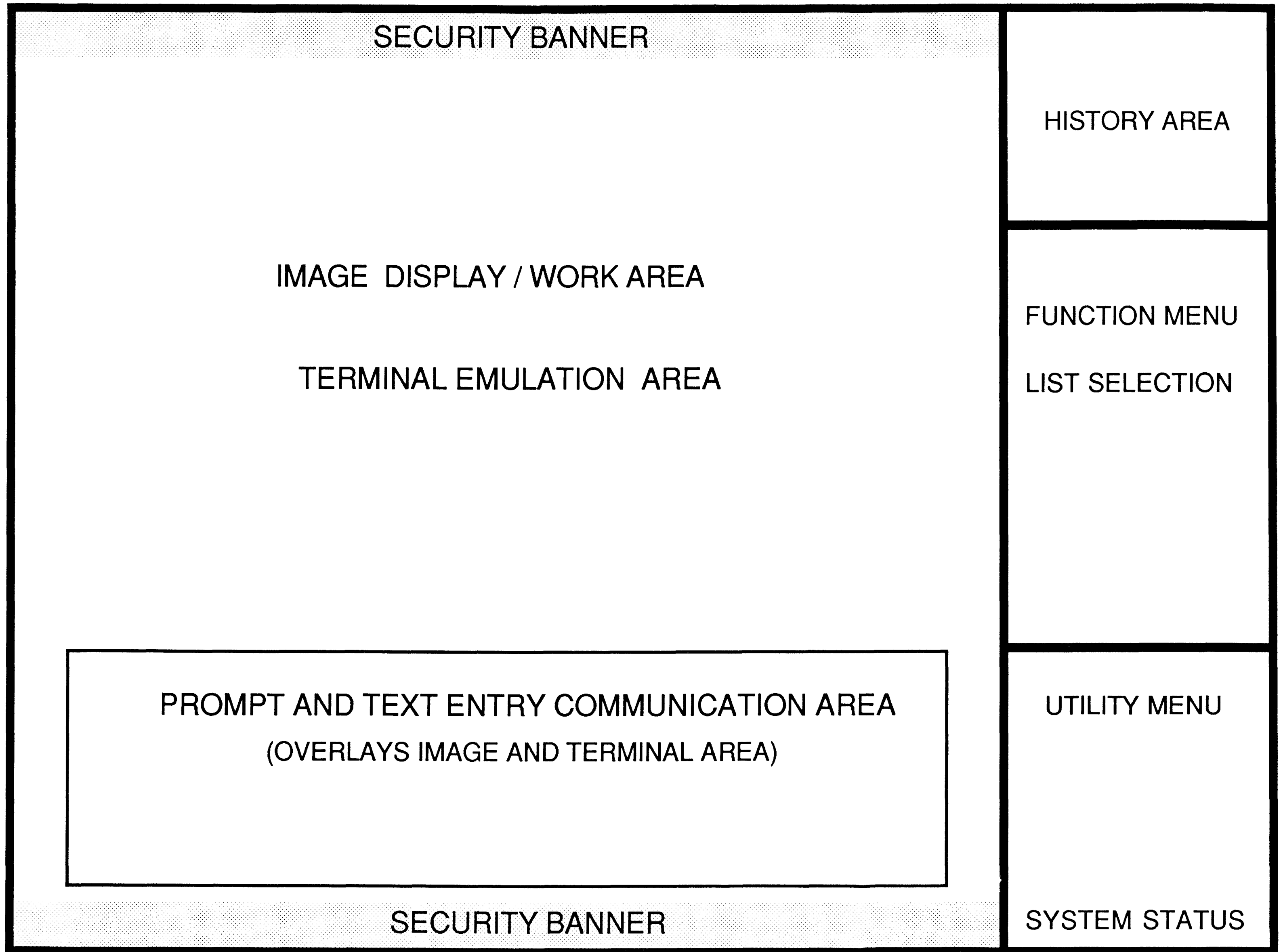



FIGURE 1

### System Status Indicator

The system status indicator is used to indicate the current system status to the workstation operator. When the system is waiting for input, the word 'READY' appears in green letters. After input is successfully accepted, and passed to the system for processing, the word 'PROCESSING' appears in red letters. The processing message remains until a request for input is received by the ACP.

### Prompt and Text Communication Area

The communication area is used to prompt the user for data entry and to display advisory and error messages. Prompts and advisory messages are displayed using white letters on a blue background. Error messages are displayed as white text on a red background.

### Image Display Area

The image area is used for the display of imagery associated with the current lesson. The image area is also used to present courseware and forms to the student and consists of a 24 by 80 column display. This area is used to emulate a VT100 terminal and a TEKTRONIX 4105 terminal. For VT100 emulation, a monochrome display is generated. Reverse video, bold, blink, double height and double wide text are supported. For TEKTRONIX emulation, 8 colors and pixel graphics are supported.

### Security Banners

The security banners present the security classification of the currently displayed image. The security banner consists of white text on a red background.

## System Requirements

The Computer Based Training system was designed to meet the following requirements:

- display of black/white imagery in 16 shades of gray
- display of imagery annotation (line, symbol and text) in 8 colors
- display of Computer Based Training course material emulating a TEKTRONIX 4105 terminal
- Forms display and update (using FMS) by emulating a VT100 terminal
- Color display of menus
- Menu selection using the mouse
- Textual keyboard input

In order to maximize the use of existing software, the ACP was designed to perform two types of terminal emulation. VT100 emulation allows the use of standard DIGITAL editors and Forms Management (FMS). TEKTRONIX emulation allows the existing Computer Based Training software to run with minimal modifications. The emulations are never performed concurrently. The ACP is either emulating a VT100 or a TEKTRONIX terminal in the terminal area of the screen.

## ACP Design

A MicroVMS Ancillary Control Process (ACP) was chosen to meet the system requirements. The ACP fits logically between the hardware devices (graphics display, keyboard, mouse) and the workstation application software. As a privileged process, an ACP often executes in kernel mode allowing it to access VMS data structures. As a process, an ACP can call VMS system services and Run-time Library routines. In addition, an ACP has the full VMS 32 bit address spaces.

DIGITAL has traditionally used ACPs to manage non-sharable resources, such as disks, tapes and communication network hardware. ACPs are currently used to control tape devices and to allow several application programs to 'share' network devices. Additional processing can also be performed on data as it passes between hardware and the application programs. NETACP, for example, implements several layers of the ISO network model.

The Computer Aided Instruction ACP performs the following functions:

- accept keyboard input
- accept mouse input
- share the input and output devices between several processes
- manage the various zones of the graphics display
- temporarily stop process activity by delaying the completion of QIO read requests

Since all input passes through the ACP before it is delivered to the appropriate application process, the ACP can accept menu selections and convert them to a sequence of key presses to be passed to the application. This feature is especially useful when running software which requires a sequence of function or control keys to be pressed (ie editors and FMS). The application software receives the input and runs accordingly, without knowing that the input keystrokes were actually generated by the ACP.

The ACP maintains a separate data structure for each of the four workstations it supports. Each workstation database contains variables which define the current state of the workstation, including emulation state, cursor position, current color, crosshair location, etc.

Terminal emulation (VT100) is provided by the use of a VMS virtual terminal. The standard TT class driver is joined to a unique 'port' driver. The 'port driver' passes all data from the application to the ACP for display. The

ACP passes data through the 'port driver' back to the application. This technique minimized the amount of software required to implement terminal emulation. The VMS TT class driver understands all standard terminal QIO requests and simply passes data to the ACP. Likewise, the ACP passes input data to the TT class driver who delivers it back to the user with any associated ASTs and status blocks.

The terminal emulation software is designed using a state machine approach. A state machine defines the current state of software by describing each state transition. Each character received and processed by the terminal emulation software causes a state transition to occur. Escape sequences are handled in this manner. Receipt of an escape character (ASCII 1B) causes a transition to state ESC. Receipt of a J when in this state causes part of the screen to be erased. Receipt of a J in the normal state results in the displaying of a capital J.

Mouse and keyboard input is achieved by using the VMS connect to interrupt mechanism. All interrupts from the devices cause data to be stored in a system mapped buffer. The character handling routines that process data from the mouse and keyboard also use non-paged pool when data must be passed to the ACP. The interrupt service routine buffers characters (with line number) into a ring buffer which is doubly mapped in system and process space. A fork routine is started to empty the ring buffer. The fork routine removes data and determines the context of the data from data stored in workstation databases. The appropriate workstation database is located by using the line number stored with the data as an index into a list containing the addresses of the 4 workstation databases.

The graphics card (MATROX QG-640) is controlled by a modified VMS driver. The graphics board supports 3 dimensional graphics, raster pixel moves and a hardware supported cursor. The ACP places graphics commands in a display buffer (within the workstation database) and issues a QIO to the driver. Since the workstation database is locked in memory, direct IO is used. The data is moved via programmed IO (1 byte at a time) to the graphics hardware control/status register. The driver also contains an alternate startio entry point which is used for cursor movement. If the driver is busy, the cursor movement (alternate startio routine) simply stores the new crosshair position in the Unit Control Block (UCB). After the current IO completes, the driver issues the update crosshair commands. If the driver is not busy, the alternate startio routine updates the crosshair immediately by issuing the necessary graphics commands to the graphics controller..

Application programs request the display of menus and images by issuing request to a pseudo driver (GT:). Communication between components of the IO subsystem is achieved using non-paged pool. The QIO system service (EXE$QIO) allocates an IO request packet for requests to the GT device. FDT routines in the GT driver allocate additional non-paged pool to hold data to be passed to the ACP. The address of the system buffer is stored in the IRP at offset IRP$L_SVAPTE, so that it can returned to pool as part of IO completion. In addition, the number of bytes of pool charged against the requesting process is stored in the IRP so that quota can be updated during IO completion.

The Virtual Terminal (VT) port driver uses non-paged pool to pass characters to the ACP. For large data transfers, the VT startio routine allocates a system buffer of sufficient size to buffer the data. For single character (or small requests) the VT driver allocates a small request packet (SRP) from the SRP look-aside list.

## Conclusions

The ACP design has proven to be flexible and efficient. By isolating all human factors issues to a single image (the ACP), changes are easy to implement and are done transparently to the application software. In addition, all communication between application software and the ACP is achieved via pseudo devices. The association to devices is done at run time using the $ASSIGN system service. No application code is linked to the ACP. The interface is thus easier to define and modifications are simpler to implement.

One problem was encountered during the implementation. In order to update the system status indicator, it is necessary to know when a read is currently outstanding to the virtual terminal. The port/class driver interface does not explicitly pass this information. It is assumed that all data received from the port driver will be passed to the TT class driver. The TT class driver then buffers the data in the type-ahead buffer or returns it to the application program. One possible solution is to check the Virtual terminal Unit Control Block (UCB). Presumably the information will be available here.

# A DISTRIBUTION DATA BASE FOR REAL TIME DATA ACQUISITION AND PROCESS CONTROL*

David L. McGuigan and Robert W. Carey
Scientific Software Division
Computations Department
Lawrence Livermore National Laboratory
Livermore, California

## ABSTRACT

We are developing a distributed data base for real time data acquisition and process control. The data base has a relational structure with the added capability of associating executable code with data access providing the flavor of an object oriented system. This flavor of an object oriented system allows the development of a hierarchy of objects that are made up of lower level objects already defined. This structure provides for a unified view of the system operation reducing development costs while enhancing system flexibility.

The contents of the real data base are described by an offline configuration data base. Management of the offline configuration data base contents is provided through an RDB (1) data base on a VMS host. The user interface to the configuration information is provided through the use of the FMS-11 Forms Management System. This provides a controllable interface that is easy for novice users to utilize.

The system works on a network of MicroVAX II's running ELN and additional host systems running VMS. It is used to support a complex control system application currently in development.

## INTRODUCTION

We are developing a control system for a complex application that requires highly interactive user interfaces and signal reporting for approximately 3000 I/O points every second. The control system is organized into two levels. The first level interfaces directly with the process hardware and is called the Process Interface level. This level provides control capability through a video terminal interface. There is very little human engineering at this level. The second level is called the Supervisory level. This level incorporates a high degree of human engineering in the interface supplied to the operators. Graphics displays with touch panels are the interface. This level provides very high level control functions such as sequences.

The control system software provides generic data acquisition and control capability. It is based on a distributed data base that maintains all the real time values in the control system. The data base provides the capabilities of an object oriented system to increase system modularity and easily support data driven software. The use of a distributed data base was driven by a desire to reduce system complexity. This data base provides a common view of the system to both levels of the control system.

In this paper we discuss the object oriented distributed data base system. We start with a discussion on the philosophy of an object oriented system and its applicability to a control environment. Then we describe our implementation of that philosophy. A discussion of some of the internal details including system configuration, system generation, the local data base interface, the remote data base interface, and some important underlying data structures is included. We then summarize with the current status of the project and its future directions.

## THE OBJECT ORIENTED SYSTEM

In a process control system, it is easy to identify objects that must be controlled and monitored. These are things like valves, pumps, and temperature monitors. In a conventional control system, there are some data that represent these objects that are available to the application programs. When it is necessary to perform an

*This work was performed under the auspices of the U.S. Department of Energy by the Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

action, these data are accessed by the application program. The proper method to perform the action is then determined, and the action is executed in the system. It is the responsibility of the application program to know how an action must be carried out. This requires the application programs to accomplish two complex tasks; to determine WHAT must be done and HOW to do it. The following subsections deal with an object oriented approach to system design. We will briefly define what a classical object oriented system provides using Smalltalk (2) as an example. We follow this with a discussion of our implementation that gives us much of the same flavor.

### Object Oriented System Defined

In an Object Oriented System, an object is a complex data structure and a set of operations that can be performed on that data structure through a well defined interface. To perform an operation on the object, the user sends a message to the object asking it to perform the operation. The object responds with a message reflecting the completion status of that operation. The benefit of the object oriented approach is the enhancing of modularity. Determining WHAT to be done is separated from determining HOW to do it, resulting in reduced system complexity. If the information or method of accomplishing an operation changes for an object, only the object needs to be modified as long as the well-defined interface is not affected.

The use of an object oriented approach facilitates a hierarchical system development. The lowest levels handle the details of the input/output devices and the higher levels abstract the lower level devices into something more meaningful to the user. As an example, there could be several binary points that give different information about a valve. There would be objects representing each hardware binary point. The valve object would then reference the point objects to determine its information. If the hardware system was changed from CAMAC to RS232, only the binary point objects would be affected and the valve object would remain the same.

In a control system there is assuredly a multiplicity of similar objects, such as valves or temperature monitors. Using the terminology from the Smalltalk Environment, these are CLASSES of objects. Classes represent objects in the system that behave in a similar fashion. An INSTANCE of a class is the representation of a particular object in the system, such as Intake Valve 23 and Outlet Temperature Monitor 5. All instances in a class accept the same messages and use the same logic, or METHODS, for executing the message requests. The contents of the data structure, or INSTANCE VARIABLES, allows each instance to perform in its system specific manner. In the above example, each valve would reference a different set of binary points, but each set would be referenced in the same way.

### The Run Time Object Data Base

Our system has several features that parallel the Smalltalk Environment. We have GROUPS that are made up of MEMBERs. Each GROUP is of a defined GROUPTYPE that has a specific set of PROPERTIES. A GROUPTYPE is equivalent to a class, a MEMBER is equivalent to an instance, and a PROPERTY is equivalent to an instance variable. The methods of a member are defined by routines or ACTIONS that are associated with the accessing of properties. Accessing a property is the way a message is sent to a member to perform a method. The properties can be defined to have ACTIONS executed before or after the reading or writing of a property, providing much flexibility in how the methods are accomplished.

In our system, there can be several GROUPs of a GROUPTYPE. The necessity of having several GROUPs to represent the same type of object comes from the distributed nature of our data base and will be discussed in more detail in the section describing updates around the network.

The groups are organized in a relational structure. The MEMBERS make up the rows and the PROPERTIES make up the columns. We then extended the relational structure to provide the capabilities of an object oriented system. The relational structure was chosen because it maps directly onto the structure of the configuration data base maintained in an RDB data base. The relational structure also allows the system to handle data base requirements that are not optimally supported by a totally object oriented system.

This organization has the benefits of both the relational data base and an object oriented system. It provides the benefits of an object oriented system through modularity and easy abstraction. This easily supports the definition of objects, such as valves and temperature monitors, that the user programs can reference without having to consider the details of the implementation of those objects. It also provides the standard data base functionality of a relational system. The users only have to learn one set of interface routines to access the data in a relational or object oriented manner.

We call the system the Run Time Object Data Base and it is a collection of several smaller subsystems. There is an interactive program used to manage the configuration information called the Configuration Manager. There is also a utility program that translates the configuration information into a form that can be built into the running system called the Object Allocation System. These tools make up the offline support. The run time system is made up of two subsystems, the first of which is a set of routines called the Run Time Object Interface (RTOI) that the user programs utilize to access the data base. The second major component of the run time system is the Network Server, which is implemented as a process per node that maintains the consistency of the data base around the network. The software organization is shown in Fig. 1.

## SYSTEM INTERNALS

This section provides a detailed description of the software architecture and the functionality provided that implements the distributed real time control data base. We start with a description of configuration definition and data object allocation to different nodes in the network. Then we discuss the run time system by describing operation of the

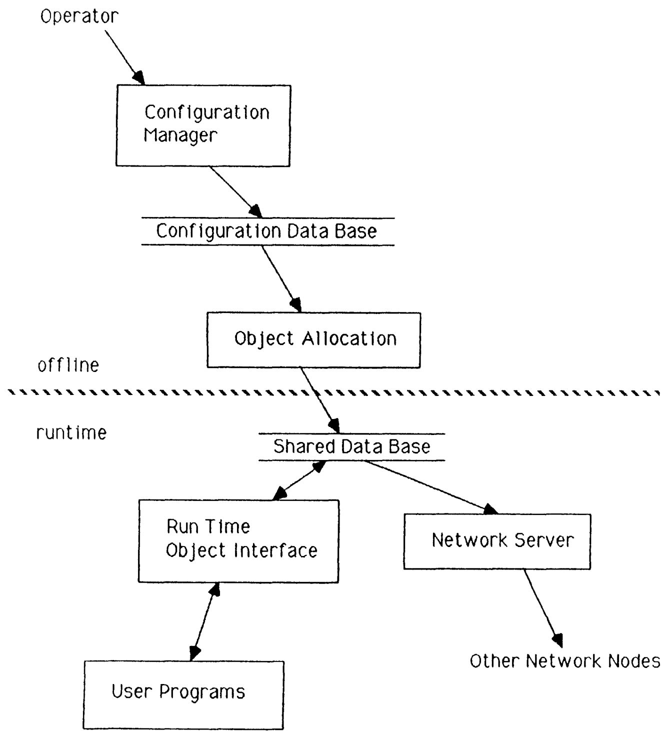



Figure 1.

data base in a local node and then extend that to a network of cooperating nodes. At the time of the writing of this paper, the single processor version was up and running and work was proceeding to extend it to the distributed architecture. For ease of presentation, we will describe it as if it were implemented according to the current design. Complete details on the status and performance of the system are provided in a following section.

## The Offline System

Configuration Management. The Configuration Management brings together RDB and FMS to allow for menu driven configuration definition. With this tool, the user can define data GROUPTYPEs, GROUPs, GROUP locations, and the initial run time values for MEMBERs. These logical organizations of data are defined in RDB as tables that are used to generate a run time data base. The Configuration Manager allows for the creation, modification, deletion, and selection of control system data objects. It provides a menu for access to different types of data definition, field validation on data entry, and help processing. There are three main system configuration tables plus all the actual MEMBER tables that are managed by the Configuration Manager.

The three system configuration tables that describe the structure of data managed by the run time system are the GROUP, PROPERTY, and GROUPLOCATION tables. The GROUP table describes the run time data tables. It contains the GROUP name, the GROUPTYPE, and the primary location of a GROUP in the network. The PROPERTY table contains the GROUPTYPE definitions. It describes the attributes of each different GROUPTYPE. Each attribute or PROPERTY of a GROUPTYPE is defined by the owning GROUPTYPE name, the PROPERTY name, the names of the routines that perform the pre and post read/write actions, the field type, the field size, and whether the field contains static or dynamic data. The GROUPLOCATION table defines all network locations that a table is distributed to. The significance of GROUP locations in the network is presented in the section on the run time system.

The MEMBER tables comprise the actual run time data base that describes the application system. The design of the configuration data base is of paramount importance since it is directly translated into a run time format. Adherence to standard data normalization rules for the relational data model are the first set of guidelines. MEMBER tables are described by a set of PROPERTIES that are organized as a primary key and a set of attributes. All attributes should directly describe the primary key and not be dependent on other attributes.

Beyond normalization rules for data base design, we must take into account the concepts of the object oriented data base and the control environment. The main idea is to design a hierarchy of data objects starting with low level I/O point data and then abstracting that data logically and repeatedly. The final abstraction may then represent an entire subsystem. Once the data base is designed, we use the configuration manager to input the hierarchy of data objects in RDB. This method of configuration management is working well and will remain an integral part of building and maintaining the application control system.

Object Allocation. The Object Allocation component of the system provides the program mechanisms for translating the RDB control system configuration data base into a run time data base in the form of a link loadable program section for a specified control system node. It takes the RDB data base defined by the Configuration Manager tool and builds a completely initialized run time format that can be linked with the target application programs. The format includes GROUP definition information, PROPERTY definition information, and the actual MEMBER data as extracted from RDB. Object Allocation also builds indexing structures as part of this run-time data base that allow fast access to GROUP, MEMBER, and PROPERTY information. The index structures are built using chained hash tables.

The Object Allocation subsystem outputs the contents of the configuration data base as a VAX-11 Macro file, which is assembled into an object program section. The object file is then linked with the rest of the application programs as a shared data segment. The final result is an executable image containing a fully initialized data base. The use of the macro intermediate file allows for symbolic names to be used in the definitions of the data base, which simplifies the job of data definition. The names are resolved by the linker, reducing the complexity of the object allocation subsystem.

The association of actions to the data is done with three other output files that are generated by

the Object Allocation Subsystem. The first is the action routine nomination module, which is a Pascal source module that has the calling structure for all data action routines defined for the GROUPs being allocated. As each group is defined in the macro file, a unique index is generated for all the routines associated with the properties. The index of the action routine to execute upon data access is stored in the run-time data base as part of the PROPERTY information for each GROUPTYPE. This index is used as an offset to the proper procedure call in this nomination module. Another of the output files generated by Object Allocation is the action routine definition file, which contains all of the included statements that bring the action routines in at system compile time. It also is a Pascal source module. The last output file of the Object Allocation system is a PASCAL type declaration module, which contains a set of PASCAL record definitions that map to the GROUPTYPEs defined in the system.

Object allocation is also important in terms of the distributed nature of our system. It allows us to build specific run time configurations for different nodes in the distributed control system. With the Configuration Manager, we can specify that a particular GROUP be in one or more nodes in the network. Object allocation takes care of distributing GROUPs across one or more run time configurations. In the section on the Network Server, we discuss more fully how this distribution of GROUPs is maintained and where the associated action routines reside.

## The Run Time System

The run time system consists of some node specific global data, the application specific action routines, a set of shared data access routines, and a process for maintaining the global integrity of the data base. The node specific global data consists of all of the groups allocated to the node. The action routines are associated with particular attributes of data object classes and contain the logic of HOW actions are performed on the data objects. The shared routines are called the Run Time Object Interface or RTOI and represent a controlled interface to the run time data base. The process that maintains the global integrity is called the Network Server. The run time system provides the real time information to the processes running in the control system.

Run Time Object Interface. The Run Time Object Interface (RTOI) is a set of shared routines that is linked into the processes and provides access to all the information in the data base that a node was configured to contain. That information can be pure data or can represent objects; the interface makes no distinction. This provides a consistent access method across the system. The interface provides a restricted relational view of the data with some enhancements for simplifying the accessing of individual objects in the system.

All information is accessed by symbolic name through the RTOI. The use of symbolic names does incur some run time penalties but increases the flexibility of the system dramatically. Using symbol names, a much more generic system can be built by encoding system specifics in the data of the system. While it is possible to resolve symbols before run time, this increases the complexity of the system support software and increases the coupling of the system. Results presented in the project status show that the performance is sufficient using symbolic references.

The relational operations that the RTOI provides are selection, which is a subset of the MEMBERs, and projection, which is a subset of the properties. These operations conveniently provide access to subsets of information contained in GROUPs, for both reading and writing. The join operation was not provided because of the potential impact at run time. This is not a severe limitation because the Object Allocation subsystem can build GROUPs that are joins of tables in the off-line configuration data base. Since the data accesses in a control system can be defined in advance, any join operations can be done at configuration time, significantly reducing the run time overhead.

With the PASCAL types generated by the Object Allocation component that map to all GROUPTYPEs, the control system processes can access the run time data base through the defined relational operations. Using the variant record that maps onto all the GROUPTYPEs, we eliminate the need to do run time binding of program variables to query statements. The RTOI can do field to field copies of the data into and from the variant record since the variant corresponding to the GROUP being accessed matches the structure of the MEMBER of that GROUP. This provides the applications with the ability to store and retrieve data from specific GROUPs in the run time data base using standard Pascal syntax and semantics.

Since the system does not distinguish between pure data and objects, it is very easy to perform operations on a set of objects. Performing operations such as opening a set of valves is easily accomplished. A selection criteria describing the set is defined. A projection specifying the COMMAND property is set up. The relational interface is then used to store the value OPEN in each of the valve objects specified. We believe this to be a powerful and very flexible control mechanism.

Access to individual objects is provided in the capability to access individual values in the global data base. Using the GROUP, MEMBER, PROPERTY the hashed indexed structures are quickly traversed for rapid access to the desired value. This access to individual values easily supports a data driven user interface. The GROUP, MEMBER, PROPERTY to be manipulated can be maintained symbolically in the data structures defining the screens that are presented to the user. The user interface can maintain a pointer into the data structures for referencing which set of names is associated with a field the user is currently interested in. Using the names and a value the user specifies, the interface software can access the data base. The object oriented data base then handles all the system specific functions so the user interface software is not tied to application.

Network Server. The Network Server process provides a communications center in each control system node. The primary purpose of the Network Server is to maintain the distributed data base.

This entails refreshing distributed copies of data throughout the network. To describe how this is accomplished, we first describe how groups are handled in the system.

In the system, a GROUP can exist in one of two forms. It can be a primary GROUP or it can be a ghost GROUP. A ghost GROUP is a read-only copy of a primary GROUP that exists on another node in the system. There are no actions associated with a ghost GROUP. Only the primary GROUP has the actions, if any. One primary GROUP in the system can have many ghost copies of it distributed around the network.

The primary GROUP is allocated to the node that has the highest update frequency to reduce network overhead. Only the primary GROUP is updated by the processes in the system. A data base update directed to a local ghost GROUP is intercepted by the RTOI and sent as an immediate message to the node containing the primary copy. This remote update will then be reflected in the local ghost copy in the next update sent by the remote network server. This prevents inconsistencies resulting from multiple simultaneous updates by insuring all copies reflect the primary copy. The latency introduced here amounts to a maximum of one cycle of the network server that is designed to be one second.

To maintain what changes have been made to the primary copies of groups on a node, a shared data structure is managed by the RTOI and the Network Server. The structure is a doubly linked list that is threaded by groups in one direction and nodes in the other. When a write is done to a primary group, the RTOI walks the structure through the group links, putting an entry at each node intersection describing the change that has been made. When the Network Server for a particular executor node does the periodic updating, it walks the structure through the node links, building a package for each remote node containing what needs to be updated from the executor node's primary groups. All changes directed to a remote node are packaged into one message. The message overhead for update information to ghost copies is low since data for many GROUPs can be packaged as one message that is sent periodically. This provides an efficient method for maintaining the distributed data base consistency.

The need for multiple groups of the same type comes from the desire to reduce the network overhead. By providing multiple groups of the same GROUPTYPE, the commonalty of the treatment of the data is preserved while the segmentation of the data is increased. The increased segmentation allows the distribution of data to only the nodes that are interested in that information, reducing the amount of updating in the system. As an example, if there were only one group in the system that represented all valves, then when any valve in the system changed, all copies of the group would have to be updated to maintain data consistency. So if a valve on node 1 opened, node 3 would have to be informed even though node 3 does not care about the valves on node 1. By segmenting the valves into groups related to the nodes, only the nodes that are interested in valves of node 1 would be updated when that valve opens.

In terms of a control architecture, we have a polled system. We contrast this to an interrupt-driven architecture where updates are pushed out as they occur. We believe a polled architecture accommodates a wider range of system activity in a more consistent manner. This is because of the relatively constant message traffic regardless of update frequency in the network. In an interrupt-driven system, each data base update generates one or more messages, and as the update frequency goes up, more and more CPU cycles must be devoted to message processing. Performance can then vary widely between peak periods of activity since the number of messages varies directly with system activity. In a polled architecture, where we package many updates into a single message, we avoid this performance variance at peak periods. This allows us to provide approximately the same performance over a wider range of system activity.

One of the features of this design is a rapid local read capability. All data base read operations are resolved in the local node without message traffic in the network. If a particular node needs information contained in primary GROUPs in another node, the table(s) containing the information are allocated at system generation time as ghost copies on the node requiring the data. Analysis shows that the majority of compute activity in a control system is for monitoring and displaying state information. Therefore, we have endeavored to make the reading of data base information as cheap as possible in the context of a node in a distributed process control environment.

Certain applications, such as sequences, require exclusive update capability for values in the data base. Sequences are a set of logical steps that take the state of the system from a current known state to a new desired state. We have implemented a data locking primitive in our distributed data base to ensure exclusive write access. Each process in the network has a unique identifier that is used as the lock key. Locks are implemented to allow locking at the field level within GROUPs. A lock allows only the owner to write to the field, but allows any and all read requests. Locking is particularly important when performing sequences. There may be many steps which depend on the outcome and steady state of previous steps. It is therefore necessary to lock devices in particular states such that the sequence can proceed knowing that previous steps will not be altered by other asynchronous processes in the network.

Another important function of the network server is the maintenance of the communication links. It is responsible for initiating and accepting communications with other interested nodes at system startup. For each logical link between two nodes, there is an acceptor and a requester node. The network server works using a link data base that describes each link that is maintained on a particular node. A node may accept connection with some node(s) and request connection with some other node(s) based on the definitions in the link data base.

The network server is also responsible for detecting when a communication link fails and for re-establishing that communication. Failure detection is simple since all nodes are in periodic

communciation in our polled architecture. The same logic for accepting or requesting a connection is followed. The attempt to make connection is repeated on a periodic basis until such time as the connection is re-established.

The network server is built around the use of VAXELN circuits, ports, names, and messages. These kernel data objects combined with the ETHERNET data link driver from the VAXELN Network Service. This Network Service uses the Phase IV DECnet Network Services Protocol (NSP) Version 4.0 and Session Control Protocol Version 1.0 to provide transparent application-level circuits to remote nodes. Each node in our control system network has a network server process that uses the VAXELN Network Service mechanisms to communicate with other cooperating control system nodes. We found the networking capabilities provided by VAXELN to be very useful in simplifying the design of our network server and the required distributed processing overall.

## PROJECT STATUS AND BENCHMARKS

The project is progressing on schedule at this point. The single node configuration is currently functioning and performing local control functions in several prototype environments. Data base access times were tested for the single value and relational access mechanisms. For a direct retrieval of a single data base value from one GROUP, the delivery time is approximately 0.67 milli seconds, which equates to 1500 data base values per second. Using the relational operators for retrieving mutiple PROPERTIES from mutiple MEMBERs from a single GROUP, we get single value delivery times of 0.17 milli seconds or 6000 values per second. These access times should remain relatively constant with different table sizes based on the hash table access method. Our table sizes are relatively small with the largest ones only around a few hundred MEMBERs. The above access times seem to be quite acceptable for the data base traffic we are observing.

In evaluating the above times, two points need to be considered. First is that these times do not include any accesses invoking actions. There is no reasonable way to estimate what an average action is so they were omitted from the timings. The second point is that in using an object oriented approach, we can significantly reduce the number of data accesses required. To perform an operation requires accessing only one value, which is equivalent to sending the member a message, and letting the member handle the details. This significantly reduces the overhead because the member can reference its properties with almost no overhead. This one access by an application can accomplish a task that would require many accesses with a non-object oriented system.

We have performed some benchmark testing of the run time data base to determine access times and cycle times for data acquisition. The data acquisition time for a single node configured for 300 I/O points is approximately one half second per scan cycle. This time includes the display of the information on a VT200 class video terminal. This affords us room to add on the network functions and still meet the one second cycle time requirement.

We have also done some network benchmark testing to measure network message processing speeds for VAXELN and VAX/VMS. We tested message passing between two dedicated MicroVAX II's running the VAX/VMS operating system. Here we observed full machine saturation at approximately 60 messages per second for message sizes of 512 bytes or less. The same test done when both nodes were using the VAXELN run time environment, yielded repeatable results of 200 messages per second. We, of course, feel much more comfortable using VAXELN because of its much lower message processing cost. We feel that we can process as many as 40 network messages per second in each node without severely impacting system performance. Our analysis shows that message traffic will usually be less than 20 messages per second worst case.

These benchmark results give us confidence that the design decisions were valid. They indicate that there is sufficient capacity to handle the requirements of our system with room to spare.

## SUMMARY

In summary, we have designed and are in the process of implementing a distributed data base for real time data acquisition and process control. We have selected hardware and software products from Digital Equipment Corporation that give us a good system base to build from. These include the VAXELN operating environment, VAX/Rdb VMS and FMS-11. The real time data base operation is based on the precepts of an object oriented system and a performance focused relational data base design. The object oriented features provide for the association of a set of operations as part of data definition and data access. The relational model provides well defined data normalization techniques and data independence. The overall approach facilitates a hierarchical system development and greatly enhances system modularity. The design also supports the consistent distribution of the data base across many processing nodes.

We are very pleased with progress to date and the empirical results, both in functionality and performance. The majority of the software is generic in nature and very amenable to different control applications. This is due to the fact that system operation is largely data driven. We have built several different single processor systems using the current implementation. The majority of work for new control applications is basically designing the data base and action routines.

We are anxiously proceeding with the design and implementation of the fully distributed Run Time Data Base. We believe the flexibility and elegance of the software architecture will allow us to address any problems that arise. When fully implemented, the Run Time Data Base will allow the development of application software without consideration of the distributed environment. The run time data base will manage all those details for the application. The application software will not and should not be aware of or burdened with the problems of dealing with a distributed process control environment.

ACKNOWLEDGEMENTS

We would like to thank all the members of the EDS control system development team for their efforts in the development of this system. Their help with the review of the analysis and design of the system helped remove many problems from the system early in the development life cycle.

REFERENCES

1. DEC, VAX, MicroVAX II, ELN, RDB, FMS-11, VAXELN, VAX/Rdb, DECnet and NSP are trademarks of the Digital Equipment Corporation, 12 Crosby Drive, Bedford MA 01730

2. A. Goldberg and D. Robson. Smalltalk-80, The Language and Its Implementation. (Addison-Wesley), 1983.

# MicroVAX II IMAGE PROCESSING TUTORIAL

John Molinari
Data Translation, Inc.
Marlboro, MA

Digital image processing is the technology of using computers to receive and interpret visual information from real-world scenes. Applications of this technology are growing fast in medicine, factory automation,robotics,and surveillance as specialized hardware for computers such as the MicroVAX II becomes more powerful and less expensive. Frame grabber,frame processor, and array processor products for the MicroVAX II exemplify this rapid development. This article introduces the basics of frame grabbing and frame processing for digital image processing.

The technology of using computers to receive and interpret visual information from real-life scenes is taking off. This technology, called digital image processing, allows doctors to enhance and examine diagnostic images from computed-axial-tomography (CAT) scanners, ultrasound, magnetic-resonance,and radiography devices; manufacturing engineers put robots to work, or employ "vision" systems for quality assurance inspection; and scientists monitor details on the earth's surface and voyage vicariously through billions of miles of space to seek new planets, moons, and stars.

Although the recent surge in the popularity of digital image processing is new, interest in the technology is not. Researchers and scientists have been interested since the early sixties when NASA was hard at work studying the surface of the moon in preparation for the Apollo program. And another government agency, the United States Geological Survey, years ago started a program called LANDSAT whose reconnaissance satellites capture digital images of earth, and most recently provided some of the images of the nuclear reactor mishap in Chernobyl.

Key to the growing popularity of digital image processing is the availability of low cost image data acquisition and processing boards and software for the MicroVAX II. The mostpopular of these, the frame grabber, can digitize a video signal, store an image in on-board memory for processing, and display the image at a real-time rate of 30 images per second. Auxiliary processing boards such as pipelined frame processors and array processors are also popular. These boards incorporate specialized hardware for executing lengthy arithmetic calculations typical of many image processing operations.

**Video**

Before a computer can process image data for manipulation and enhancement, the image data must be made available to the computer. Video cameras are a typical source of real-world image data and present this data in the form of an analog video signal. Ordinary VCRs are also a common video source since they provide an inexpensive and convenient means for storing real-world images. Most video devices, like video cameras and VCRs, conform to the RS-170 (CCIR, 50Hz systems) or NTSC (PAL, 50Hz systems) television standards. RS-170 contains lines of black-and-white (monochrome) video - visual data - and synchronization (sync) data. NTSC is basically identical to RS-170, except NTSC signals contain color visual data as well as monochrome video and sync data.

A single image frame from either of these signals contains two fields of lines, an odd field comprising all odd-numbered lines, and an even field comprising all even-numbered lines. Horizontal sync pulses separate lines of video from each other while vertical sync pulses separate entire fields from each other. Displaying one field after the other, at a rate of 60 fields/second, called interlacing, yields a complete image frame display at a real-time rate of 30 frames/second.

Some video sources, such as CAT scanners,scanning electron microscopes (SEMs), and slow-scan cameras, provide non-standard video signals which are not compatible with recognized video standards such as RS-170 and NTSC. To accommodate these signals, frame grabbers must provide separate sync inputs for setting variable frame, line, and pixel digitization rates.

**Capturing Images**

An analog-to-digital converter is required to digitize the video signal. All frame grabbers, to operate in real time, use a high speed flash A/D converter. The conversion procedure is analogous to placing a grid over the input image. This grid is typically 256 or 512 units across by 256 or 512 units high. The image sampling circuitry effectively looks at the contents of each unit in the grid, determines an analog voltage value corresponding to the average brightness level of that unit, and, using the A/D converter, changes that analog voltage into a binary value called a pixel.

Note that the proportional shape of the pixels, and thus the proportional shape of the entire image, may be square or rectangular. The ratio of the horizontal dimension of the image to the vertical dimension of the image is termed the aspect ratio. RS-170 signals have an aspect ratio of 4:3—their width is 1.3 times their height.

The number of pixels which make up a digitized frame determines the spatial resolution of the image. This may be thought of as the spatial granularity of the image—the number of light and dark dots which make up the image. Spatial resolution is analogous to the number of scan lines displayed in a television picture.

The spatial resolution of digitized frames is expressed as a matrix: the number of lines (rows) into which the image is divided by the number of pixels (columns) per line. Typically spatial resolutions are 256 lines by 256 pixels, or 512 lines by 512 pixels. A second kind of resolution is brightness or gray scale resolution. The possible range of brightness values or gray levels for a given pixel is determined by the resolution of the A/D converter: an 8-bit A/D converter produces a range of 256 possible gray level values; a 6-bit A/D converter produces a range of 64 possible gray level values.

Most frame grabbers use a special circuit, called a phase-locked loop (PLL), to align the internal timing of the board to the timing of the incoming video signal. The PLL is designed literally to "lock" onto a stable frequency; therefore, PLLs work well with video cameras since they produce stable horizontal and vertical sync pulses marking new lines and frames of video. To operate with less pristine signals, such as those produced by most VCRs which contain signal noise, noise spikes, and missing sync pulses, PLLs must be specifically designed for robustness.

Because most frame grabbers acquire images in real time (that is, in the 1/30 second RS-170 allows for each video frame), special video-speed A/D converters must be used. RS-170 allows 52.59 us for each line in the image to be sampled. This is called the active line time. In this time a 512 by 512 frame grabber, for example, must convert 512 pixels, or perform 512 A/D conversions. This requires an A/D converter with a throughput of almost 10MHz (52.59 us divided by 512 equals 103 ns; 103 ns is equivalent to 9.74 MHz). Thus, the advent of low-cost monolithic 6-bit and 8-bit A/D flash converters has greatly reduced the cost of frame grabber designs.

Frame grabbers, by definition, operate in real time. Of crucial importance to capturing and displaying - not to mention processing images - at this rate is on-board memory, called frame-store memory. The frame-store memory is usually dual-ported - both the host MicroVAX II CPU and the acquisition hardware have access to it. Frame grabbers designed for use with auxiliary frame processing boards ideally have additional I/O ports which connect the acquisition and processing devices using dedicated data lines. One synchronous or asynchronous input port is provided at the input to frame-store memory. A similar output port is provided at the output from memory.

The memory requirements of image processing systems are extraordinary: a 512 X 512 X 8 image requires 256 Kbytes of memory, while a 256 X 256 X 8 image requires 64 Kbytes. Add to this the requirement for high speed memory access (in order to provide real-time image acquisition and display), and the requirement for greatly increased processing speed imposed by only moderate increases in resolution (doubling the line and pixel resolution from 256 by 256 to 512 by 512 quadruples the number of data points in each image) and some architectural features of real-time frame grabbers become apparent. First, for real-time operation, high speed memory, mapped directly into the address space of the MicroVAX II CPU, for storing at least one complete frame should be provided on the board. Frame grabber boards with highly advanced memory architectures may feature more than one complete frame-store memory buffer for the parallel processing of multiple images at once. Second, the frame-store memory should provide input and output access to a dedicated high speed data bus

for access to an auxiliary frame processor. To speed operations which do use the host Q22 bus, BLOCK MODE DMA should be supported to speed bus throughput.

Careful consideration should be given to the selection of frame grabber resolution. Higher resolution boards, in addition to being more expensive to buy, place much more stringent demands on the host MicroVAX II system, particularly if they are used for real-time acquisition and display. Frame data from lower resolution devices requires much less memory space for frame storage and much less processing power than data from higher resolution hardware. Most frame grabbers which are compatible with RS-170 or CCIR video signals provide 512 by 512 spatial resolution, with 8-bit brightness resolution. A number of recent designs with 256 by 256 spatial resolution and 6-bit brightness resolution are available. While the higher resolution models can be used in a wider range of applications, many users will find the lower resolution frame grabbers entirely adequate for their applications.

Some video digitizer boards are built without on-board frame-store memory. Such devices typically can neither acquire nor display images in real time. Real-time acquisition or display of 512 X 512 X 8 frame data requires transfers of 10 million bytes per second, well beyond the maximum DMA (direct memory access) transfer speeds of the Q22 bus. These modest data transfer speeds necessarily make all kinds of pixel processing very much slower on boards without than on boards with frame-store memory.

**Processing Images**

Digital image processing is a general term applied to a range of operations which alter image data from a real scene in order to extract information about the real scene which is not otherwise readily perceptible. Common processing techniques allow images to be added together, or subtracted from one another; images can be offset by a constant or operated on by logic functions. More sophisticated operations enhance the edges of objects or remove signal noise. Still other operations can implement thresholds to eliminate gray levels from images, or can attribute false colors to gray level regions in images.

Digital image processing operations can be discussed in two broad categories, those which are performed on individual pixels (pixel point processing), and those which are performed on groups of pixels (pixel group processing). Pixel point processing divides further into operations performed on a single image, and operations performed on two or more images.

**One Pixel at a Time**

Single image pixel point processing operations change the gray level values of pixels in a single frame, one-at-a-time to produce new pixels with new gray level values whose spatial location in the image is the same as before the operation. Pixel point processing operations are important for effecting contrast enhancement. A multiplication operation, for instance, changes each pixel value in an image by multiplying each pixel value in an image by a constant. Multiplication increases the contrast of an image uniformly, and can be used to brighten a dark image. When applied to an image whose pixel values all fall in the lower half of the possible gray scale, frame multiplication can stretch the range of those values so that better use is made of the possible gray scale.

A complementary operation, division, causes pixel values to be divided by a constant. Division reduces the contrast of an image, and can be used to darken a bright or washed-out image.

An offset operation uniformly increases or decreases in pixel value in an image by a constant value. Offsetting does not alter the resolution of the image, but uniformly brightens or darkens the regions of the image which contain the most information, so the features of the image can be seen more clearly. The difference between the lowest and highest intensity value is not changed (as it would be if a multiplication had been used), but the range of intensities is shifted to become brighter or darker.

The principles of dual image pixel point operations are the same as single image pixel point operations, except two or more images are involved. Again, to alter entire frames the gray levels of individual pixels are operated upon on-at-a-time to produce new output pixels, one-at-a-time. The addition operation takes the pixel values of one frame and adds them to the corresponding pixel values of a second frame. This has the effect of combining two images, and can be used to apply graphic overlays, to superimpose one image on another, or to combine live images with animation. The subtraction operation, conversely, reduces all pixel values in a frame by the corresponding pixel values in another. Subtraction shows the difference between two frames, and can be used to detect product variations in automated inspection work, or to detect moving objects in surveillance or security systems.

Note that both single image and dual image pixel point processing affect only the intensity content of a frame - the frame can be made brighter or darker. The spatial content of the image is not changed.

**Groups of Pixels at a Time**

Pixel Group Processing, on the other hand, looks not just at individual pixels, but also at the region or neighborhood around each pixel to produce one output pixel. Group processing is most notably used for spatial filtering operations which operate on pixels, groups-of-pixels-at-a-time, to enhance edges (high pass filtering) or smooth image details (low pass filtering) by accentuating or attenuating frequency components in the image.

A classic group processing operation, the spatial convolution, is a mathematical method for calculating a weighted average of pixel inensities around and including a pixel point. Repeating this operation for every pixel in an image produces a convolved or filtered output image.

The actual region or neighborhood of pixels which is operated upon by a spatial convolution operation is called the convolution kernel. Usually, the convolution kernel is a square or rectangular-shaped array of pixels. One operation on one kernel of pixels produces on output pixel value which, usually, is placed at the same spatial location in the output image as the center pixel in the kernel matrix.

Kernel dimensions are specified as N x M, where N and M are the two dimensions of the kernel. For example, in a 3 X 3 convolution, the kernel consists of a square of nine pixels (three pixels high by three pixels wide), with the target pixel at the center of the square.

The spatial convolution operates by multiplying each pixel in the kernel by a coefficient, and summing all these values to produce one output pixel, a weighted average. Thus, for a 3 x 3 kernel, 3 x 3 or nine total coefficients must be defined. These coefficients comprise a matrix called a coefficient mask and may be all the same, or all different; their values determine the characteristics of the convolution. And since every output pixel in the output image is actually produced by as many multiplications and as many additions as there are pixels in the convolution kernel, spatial convolutions are extremely arithmetic-intensive. The larger the convolution kernel, the more arithmetic-intensive and time-consuming the entire operation.

Several types of filters can be implemented using spatial convolutions. High pass filters make high frequency information (that is, rapid changes in intensity, associated with image details, edges, or textures) more prominent. A coefficient mask like the one below is typically implemented for a 3 x 3 high pass filter:

-1 -1 -1
-1 -9 -1
-1 -1 -1

The high pass filter makes small changes in the image easier to see.

Conversely, low pass filters tend to attenuate high frequency information, and thus reduce detail and blur edges in the image:

1/10 1/10 1/10
1/10 1/10 1/10
1/10 1/10 1/10

Low pass filters can be useful in removing noise from an image, or in reducing details in one part of the image so the shape of the image as a whole can be seen more clearly.

Another form of pixel group processing is Laplacian edge enhancement, also implemented using a spatial convolution. In this method, high frequencies are made very prominent, and low frequencies are highly attenuated.

-1 -1 -1
-1 +8 -1
-1 -1 -1

Since edges within an image constitute high frequency information, Laplacian edge enhancement is especially good at locating boundaries between objects.

**Hardware Versus Software**

Digital image processing can be implemented either in software or in hardware. Software implementations use the host MicroVAX II computer system. Execution speeds for software implementations are typically much slower, especially for arithmetic-intensive operations like spatial convolution, but these implementations offer flexibility. Hardware implementations require the use of specialized image processing hardware which may reside on the frame grabber board or may connect directly to the frame grabber board over dedicated I/O ports. Hardware implementations offer one very important advantage over software implementions: speed.

Image processing hardware varies greatly in sophistication, from simple look-up tables to complete array processors. Depending on the type of processing hardware provided, all or most of the pixel processing can be performed without using the host MicroVAX II.

Although specialized image processing hardware incurs additional cost, it can operate much more quickly than can the host MicroVAX II. Single and multiple frame pixel point process operations can be performed in real-time. Group processing operations can oftentimes be performed in real-time, or near-real-time.

For hardware processing, some frame grabber architectures incorporate look-up tables (LUTs) between the A/D conversion hardware and the frame-store memory. The input look-up tables map any data value to any other value. Look-up tables are fast and inexpensive to implement, and can be used for thresholding and frame addition, subtraction, multiplication, and division.

Each input LUT uses the digital data value of the pixel as an input or index into the table. Each index value has a corresponding output value. The output values are determined at the time the LUT is defined. Note that there is no necessary correspondence between the input or index value and the output value. Increasing index values can map to decreasing output values; all index values can map to a single value; groups of four index values can map to a single value, which increments; and so on. The choice is determined by the function the input LUT is intended to serve.

Single image pixel point processes can be implemented using input LUTs: the input or index value to the LUT is the first element in the operation. For each index value, the user simply performs the desired operation and loads the result as the LUT's output value. For example, to perform a pixel multiplication by two, the LUT would be loaded as follows: index value 0 has an output value of 0 ( 0 times 2 is 0); index value 1 has an output value of 2 (1 times 2 is 2); index value 2, output value 4; index value n, output value 2n; and so on. Once the LUT has been loaded with the appropriate values, simply passing pixel values through it performs the operation.

Providing a feedback loop between the output of the frame grabber's frame-store memeory and the input of the input look-up tables adds considerable flexibility. Very advanced frame grabber architectures may even incorporate an arithmetic logic unit (ALU) between the A/D conversion hardware and the frame-store memory. Operating in conjuction with the feedback loop and the input LUTs, the ALU allow real-time pixel point processing on single or multiple frames to occur in real time.

More sophisticated operations like convolutions require more complex operations on frame data. An auxiliary frame processor board with a built-in arithmetic logic unit, a frame-store buffer, and the ability to handle more than one pixel value at a time (piplined architecture) greatly speeds these operations, and provides greater accuracy than a look-up table alone on operations involving more than eight bits of resolution on itermediate results. Some frame processing hardware supports only fixed convolution kernel sizes, like the popular 3 x 3 convolution. More flexible designs support N x M convolutions, in which the kernel dimensions can be any size or shape, even rectangular or star-shaped.

Convolutions typically place enormous demands on procesing hardware, and can be slow to perform unless specialized hardware is provided. A 3 X 3 convolution, for example, requires nine multiplications and nine additions for each pixel in the frame. For a 512 by 512 image, a 3 x 3 convolution requires 2,359,296 multiplications and 2,359,296 additions.

**Displaying Images**

Once an image has been acquired and processed, it will normally be viewed. This requires image display hardware, which converts the digital pixel data back into an RS-170 analog signal. In most systms, the image display hardware consists of several ouput look-up tables and three digital-to-analog converters (DACs).

The first step in displaying a frame of digital pixel data is to transform the data through an output look-up table. After transformation through the output LUT, the data passes to the output circuitry, which converts the digital data back into analog values for display on the image monitor.

As the first step in the display operation, the data from the frame-store memory is sent to the output LUTs. The pixel data from memory becomes an index into the table. Each output LUT is 2 to the n elements long, where n is the number of bits in each pixel value in the frame-store memory. For example, if the pixel data is eight bits wide, the LUT contains 256 elements because 2 to the eighth power is 256. Thus there is one LUT entry for every possible pixel value.

In our example, look-up table elements are numbered 0 to 255. Each location in the output LUT maps to a separate value, the number of bits in the mapped value being a function of the system architecture. Let us assume an output LUT which maps 8-bit index values to 24-bit data words. The 24-bit data words determine the color and intensity of the output signal displayed on the image monitor.

As with the input look-up tables, the output values of the output look-up tables are determined at the time the look-up tables are defined. Moreover, the output values do not bear any special relationship to the input or index values.

Typical output circuitry consists of three 8-bit digital-to-analog converters (DACs), which provide RS-170 (or CCIR, 50Hz systems) analog outputs to drive the red, green, and blue inputs of an RGB monitor. The 24-bit value from the output look-up table divides into three 8-bit values which determine the output of the three DACs. The relative output levels of these DACs determine the color and intensity of each output pixel.

Embedded in each DAC output are the vertical sync and horizontal sync signals required by RS-170. In addition, the voltage output range of the DAC conforms to RS-170.

As was the case with image acquisition hardware, image display hardware typically operates in real time - a complete frame must be written out in 1/30 second. To accomplish this, special video-speed converters must be used - for a 512 by 512 frame grabber, DACs with a throughput of almost 10MHz are required.

# REAL TIME THROUGHPUT OF MICROVAX II AND MICROVMS

By Richard K. Somes
Digital Equipment Corporation
Marlboro, Massachussetts

## ABSTRACT

This paper, the second paper in a series, presents new data on polled IO throughput, interrupt processing rates, and the DMA data rates which can be supported using single transfer and block mode protocols.

Real time applications may be characterized in terms of the sample rate or the control loop update rate limitations imposed by system interrupt latency. Other applications may depend on the maximum rate at which data can be moved continuously between I/O devices and physical memory. The contents of this paper seeks to address both sets of issues.

The author has elected to publish, in Appendix A, the overhead transparencies presented at the DECUS Symposium on October 8, 1986. Appendix B contains listings of one of the test programs, written in VAX-11 C, and the CIN code, written in MACRO-32, used to acquire some of the test data presented.

Summary of contents:

o APPENDIX A

* POLLED IO - UPDATE

* INTERRUPT PERFORMANCE
  - Structure of the CIN Driver
  - QIO Interface to the CIN Driver
  - Synchronization Techniques
  - Test Methodology
  - Results

* DMA THROUGHPUT
  - QIO Interface to a DMA device
  - Single transfer and Burst Mode
  - Block Mode

o APPENDIX B - LISTINGS

## APPENDIX A - DECUS Presentation of October 8, 1986

### Polled I/O Code

Called from main program using SYS$CMKRNL system service.

```
;MODULE TO ACQUIRE N (FIRST ARG OF CALL) POINTS FROM A DRQ11-C BY POLLED I/
;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 structurs
   $IODEF                       ; I/O function codes
   $CINDEF                      ; Connect-to-interrupt
   $CRBDEF                      ; CRB stuff
   $VECDEF                      ; more

   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
   ADDL3   #6, R2, R6               ; PUT ADDR OF DBR IN R6

   DSBINT  #30                      ; ELEVATE IPL TO 30
$1:        MOVW    #TRIGGER, (R2)           ; TRIGGER THE HOLDING REGISTER
$2:                                         ; TEST THE DATA AVAILABLE FLAG
   BBC     #11, (R2), $2            ; AND WAIT HERE UNTIL IT SETS
   MOVW    (R6), (R4)+              ; STORE THE DATA
   SOBGTR  R3,$1                    ; TEST THE LOOP COUNTER
   ENBINT                           ; RESTORE IPL
   MOVZWL  R2,R0
   RET                              ; RETURN

LOOP_END:: .LONG   0

.END
```

Polled IO Performance Data

Maximum guaranteed response latency (microseconds):

13.5 Microseconds

Response latency histogram:

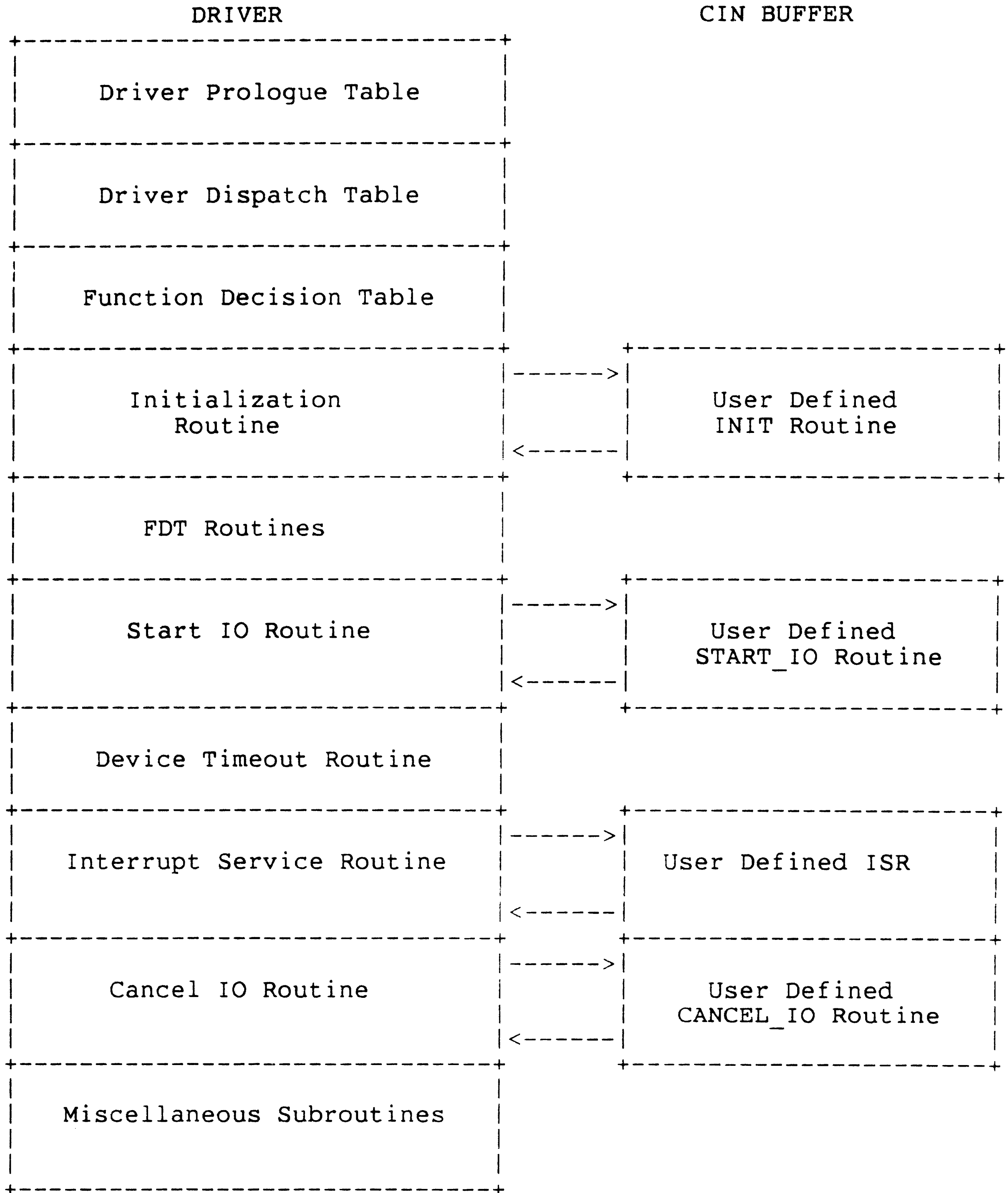
1024*1024 Data points

| | |
|---|---|
| 9.0 | 332497 |
| 9.5 | 604787 |
| 10.0 | 107195 |
| 10.5 | 625 |
| 11.0 | 2678 |
| 11.5 | 760 |
| 12.0 | 2 |
| 12.5 | 21 |
| 13.0 | 9 |
| 13.5 | 1 |

Note: When operated on the network, DEQNA DMA traffic can cause bus content which lengthens PIO times even when IPL is raised to IPL 30. The following data shows this effect.

| | |
|---|---|
| 9.0 | 333078 |
| 9.5 | 604400 |
| 10.0 | 106972 |
| 10.5 | 636 |
| 11.0 | 2634 |
| 11.5 | 794 |
| 12.0 | 2 |
| 12.5 | 30 |
| 13.0 | 10 |
| 13.5 | 1 |
| 14.0 | 2 |
| 15.5 | 1 |
| 16.0 | 2 |
| 16.5 | 2 |
| 17.0 | 1 |
| 17.5 | 1 |
| 18.5 | 1 |
| 18.5 | 1 |
| 20.0 | 2 |
| 20.5 | 1 |
| 21.0 | 3 |
| 23.5 | 1 |

STRUCTURE OF THE CIN DRIVER



## QIO INTERFACE TO THE CIN DRIVER

In C:

```
status = sys$qio(       efn,                -+
                        chan,                |
                        func,                | Device
                        iosb,                | Independent
                        astadr,              |
                        astprm,              |
                                            -+
                        p1,                  |
                        p2,                  | Device
                        p3,                  | Dependent
                        p4,                  |
                        p5,                  |
                        p6      );          -+
```

| | | |
|---|---|---|
| efn | - | VMS Local Event Flag Number |
| chan | - | Channel number, obtained from SYS$ASSIGN |
| func | - | Longword selecting desired FDT functions |
| iosb | - | Address of IO status block |
| astadr | - | Address of user defined IO completion AST |
| astprm | - | Longword parameter to be passed to AST |
| p1 | - | Address of CIN Buffer descriptor |
| p2 | - | Address of CIN entry point list |
| p3 | - | Longword containing flags |
| p4 | - | Address of user defined interrupt AST |
| p5 | - | Longword parameter to be passed to AST |
| p6 | - | Number of AST Control Blocks to pre-allocate |

Function Codes:

IO$_CONINTWRITE - CIN Buffer is writeable

IO$_CONINTREAD - CIN Buffer is read-only

Flags:

CIN$M_EFN

CIN$M_USECAL

CIN$M_REPEAT

CIN$M_INIDEV

CIN$M_START

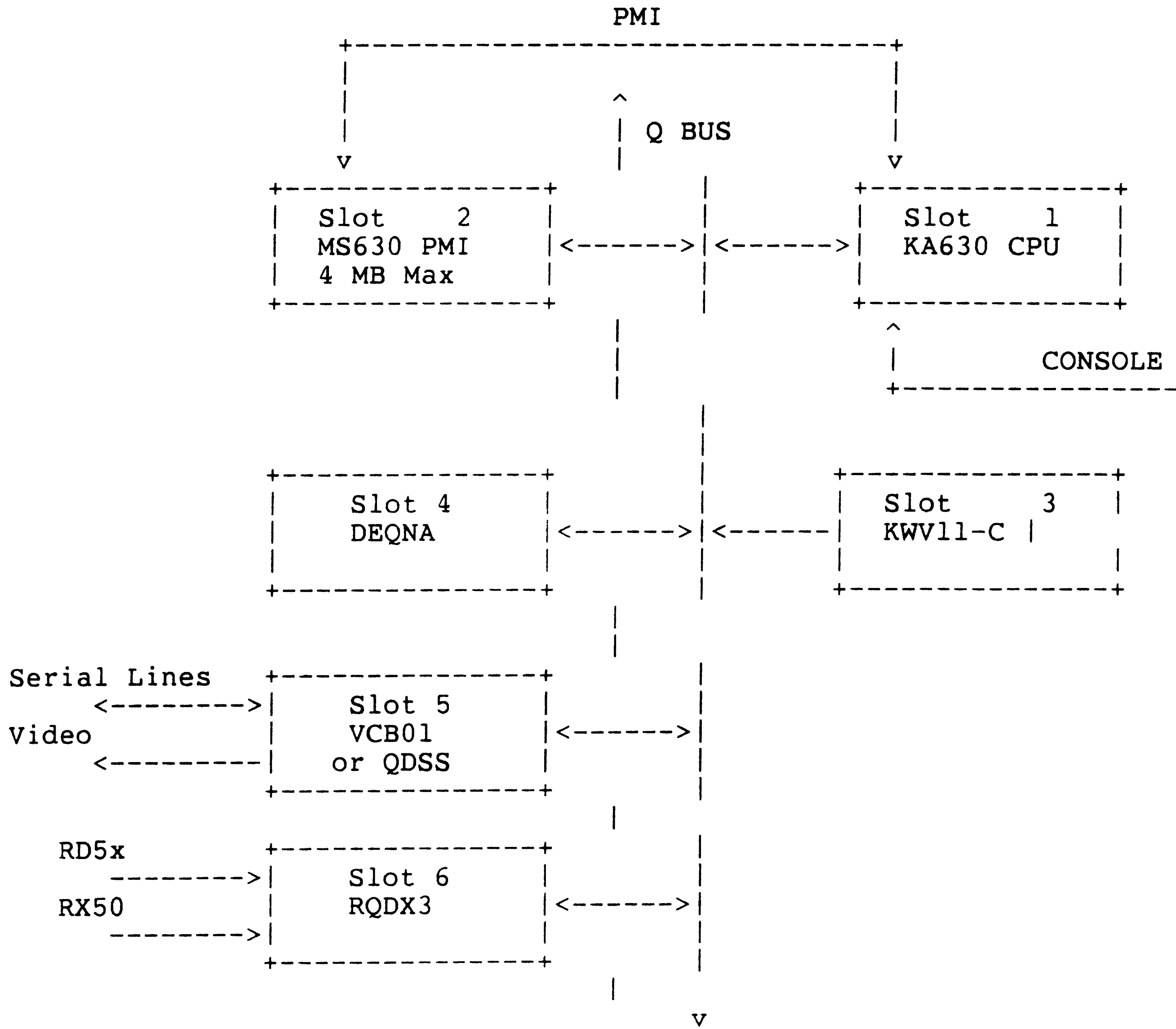
CIN$M_ISR

CIN$M_CANCEL

## SYNCHRONIZATION TECHNIQUES

- o Synchronous QIO - Not of interest
- o Hibernate/Wake
- o Wait on VMS LEF
- o Poll user defined flag set in AST
- o Poll on user defined flag set in ISR

MICROVAX II BASED PERFORMANCE TEST SYSTEM



Test Sequence

- o Map IO page into process space - SYS$CRMPSC
- o Initialize data structures
- o Top of test loop: Assign channel - SYS$ASSIGN
- o Clear counter
- o SYS$QIO to CIN Driver
- o Record Start IO time
- o Clear flag, UFLG, AFLG or LEF
- o Preset counter, start, and enable interrupts
- o Wait for flag to be set
  - ISR records latency time, sets UFLG
  - AST records latency, sets AFLG
- o Measure process latency, record ISR, AST, and Process times
- o Reset counter for Cancel time measurement
- o Cancel IO - SYS$CANCEL
- o If loop count = NLOOPS, exit and display results
  else, return to top of loop

## INTERRUPT RESPONSE DATA - NO LOAD

- o base IPL = 0
- o SW priority = 17
- o loop on user defined EF set in ISR (process is always CUR)
- o timer off
- o accounting disabled
- o no DEQNA
- o no QxSS
- o CONINTERR driver

65535 data points. Test Time = 1.25 hours

Latencies (microseconds)

| | min | mode | max |
|---|---|---|---|
| Start I/O | 736 | 755 | 774 |
| ISR | 40 | 42 | 44 |
| Process | 80 | 83 | 84 |

Maximum sustainable interrupt rate (Hz): 11 KHz

## INTERRUPT RESPONSE DATA - NO LOAD

- o base IPL = 0
- o SW priority = 17
- o loop on user defined EF set in AST (process is always CUR)
- o timer off
- o accounting disabled
- o no DEQNA
- o no QxSS
- o CONINTERR driver

655536 data points. Test Time = 1.25 hours

Latencies (microseconds)

| | min | mode | max |
|---|---|---|---|
| Start I/O | 831 | 835 | 855 |
| ISR | 39 | 41 | 41 |
| AST | 438 | 442 | 444 |
| Process | 561 | 565 | 568 |

Maximum sustainable interrupt rate (Hz): 1761 Hz

INTERRUPT RESPONSE DATA - NO LOAD

- o base IPL = 0
- o SW priority = 17
- o wait for VMS LEF set in AST (process is in LEF wait state)
- o timer off
- o accounting disabled
- o no DEQNA
- o no QxSS
- o CONINTERR driver

65536 data points. Test Time = 1.25 hours

Latencies (microseconds)

| | min | mode | max |
|---|---|---|---|
| Start I/O | 824 | 829 | 880 |
| ISR | 40 | 41 | 43 |
| AST | 610 | 613 | 623 |
| Process | 960 | 964 | 975 |

Maximum sustainable interrupt rate (Hz): 1025 Hz

## INTERRUPT RESPONSE DATA - NO LOAD

- o base IPL = 0
- o SW priority = 17
- o hibernate in main, wake in AST (process is in HIB or HIBO wait state)
- o timer off
- o accounting disabled
- o no DEQNA
- o no QxSS
- o CONINTERR driver

64*1024 data points. Test Time = 1.25 hours

Latencies (microseconds)

| | min | mode | max |
|---|---|---|---|
| Start I/O | 816 | 831 | 880 |
| ISR | 40 | 41 | 43 |
| AST | 615 | 620 | 625 |
| Process | 974 | 982 | 991 |

Maximum sustainable interrupt rate (Hz): 1009 Hz

## INTERRUPT RESPONSE DATA - COMPOUND LOAD

- o base IPL = 0
- o SW priority = 17
- o loop on VMS LEF set in AST code (process is in wait state until interrupt)
- * timer on
- o accounting disabled
- * DEQNA installed, no activity to/from host
- o no QxSS
- o CONINTERR driver

65535 data points. Test Time = 1.25 hours

Latencies (microseconds)

| | Min | Mode | 90 Pct | 99 Pct | 99.9 Pct | Max |
|---|---|---|---|---|---|---|
| Start | 816 | 830 | 833 | 961 | 2001 | 3242 |
| ISR | 37 | 40 | 40 | 42 | 71 | 123 |
| AST | 613 | 622 | 629 | 751 | 1905 | 4670 |
| Process | 967 | 978 | 1012 | 1109 | 2260 | 5024 |

Maximum sustainable interrupt rate (Hz): 199 Hz

INTERRUPT RESPONSE DATA - COMPOUND LOAD

- o base IPL = 0
- o SW priority = 17
- o loop on VMS LEF (process is in LEF wait state until interrupt)
- * timer on
- o accounting disabled
- o no DEQNA
- * QVSS installed, no screen activity
- o CONINTERR driver

65535 data points. Test Time = 1.25 hours

Latencies (microseconds)

| | Min | Mode | 90 Pct | 99 Pct | 99.9 Pct | Max |
|---|---|---|---|---|---|---|
| Start | 817 | 822 | 881 | 1005 | 1087 | 3501 |
| ISR | 34 | 37 | 37 | 121 | 177 | 221 |
| AST | 524 | 618 | 655 | 1900 | 3339 | 3926 |
| Process | 768 | 972 | 1033 | 2268 | 3696 | 4280 |

Maximum sustainable interrupt rate (Hz): 233 hz

INTERRUPT RESPONSE DATA - COMPOUND LOAD

o base IPL = 0
o SW priority = 17
o loop on VMS LEF (process is in LEF wait state until interrupt)
* timer on
o accounting disabled
o no DEQNA
* QVSS installed, scrolling in 2nd non-occluded window
o CONINTERR driver

65535 data points. Test Time = 1.25 hours

Latencies (microseconds)

| | Min | Mode | 90 Pct | 99 Pct | 99.9 Pct | Max |
|---|---|---|---|---|---|---|
| Start | 816 | 822 | 878 | 2000 | 2343 | 10046 |
| ISR | 35 | 38 | 45 | 689 | 1121 | 1211 |
| AST | 483 | 623 | 748 | 3585 | 5538 | 10748 |
| Process | 839 | 978 | 1112 | 4467 | 6974 | 11102 |

Maximum sustainable interrupt rate (Hz): 90 Hz

## CONCLUSIONS - POLLED IO AND INTERRUPT DRIVEN IO

- o SYS$CMKRNL provides mechanism for calling kernel mode polled IO code
  - Burst to memory at >60 KHz
- o CIN Driver provides flexible interface to VMS IO subsystem
- o Guaranteed ISR latency in well disciplined system configurations
  - Single device at any BR level
  - Single bus device at BR7
- o Guaranteed interrupt rates on dedicated systems
  - 10 KHz with ISR flag
  - 1700 Hz with AST flag
  - 1000 Hz with LEF or Hibernate/Wake
- o Interrupt latency can be probably guaranteed at <100 usec for a single BR7 device on loaded systems, assuming properly designed device drivers.
- o Interrupt rate is dependent on system loading and on application.
  - Network activity degrades process latency
  - Workstation activity degrades process latency
  - Worstcase process latency tends to govern maximum interrupt rate in control aplications
  - Average process latency tends to govern maximum data rate in data flow applications

## QIO INTERFACE TO A TYPICAL DMA DEVICE - DRx11-C

In C:

```
status = sys$qio(       efn,                -+
                        chan,                |
                        func,                | Device
                        iosb,                | Independent
                        astadr,              |
                        astprm,              |
                                            -+
                        p1,                  |
                        p2,                  | Device
                        p3,                  | Dependent
                        p4,                  |
                        p5,                  |
                        p6       );         -+
```

```
efn      -     VMS Local Event Flag Number
chan     -     Channel number, obtained from SYS$ASSIGN
func     -     Longword selecting desired FDT functions
iosb     -     Address of IO status block
astadr   -     Address of user defined IO completion AST
astprm   -     Longword parameter to be passed to AST
```

(Device dependent parameters for LBLK and VBLK IO operations only)

```
p1       -     Address of Buffer 1
p2       -     Buffer 1 wordcount
p3       -     Address of Buffer 2
p4       -     Buffer 2 wordcount
p5       -     Number of blocks to transfer
p6       -     Device timeout time in seconds
```

Function Codes:

```
IO$_WRITELBLK                  IO$_SETFUNC
IO$_WRITEVBLK                  IO$_READSTAT
IO$_READLBLK                   IO$_GETSTATUS
IO$_READVBLK                   IO$_SETMODE!(options)
                                       IO$M_BUFPATH
IO$_WRITEWORD                          IO$M_DIRPATH
                                       IO$M_ATTNAST
IO$_READWORD                   IO$_REALEASE!(option)
                                       IO$M_RESET
IO$_WRITELARGE
IO$_READLARGE

               IO$_NOTIFY
               IO$_BURST
               IO$_CHANGE
               IO$_LOOPBACK
```

SINGLE TRANSFER AND BURST MODE DMA PERFORMANCE

HISTOGRAM OF DMA TRANSFER TIMES

Data Summary - Large Buffer Mode, 32 buffers of 32K

| | MicroVAX II *********** | | VS II ***** | |
|---|---|---|---|---|
| Cycle Time usec | Single Cycle | Burst Mode | Single Cycle | Burst Mode |
| 2.0 | 844077 | 855491 | 689755 | 705863 |
| 2.333 | 200051 | 188635 | 353651 | 377505 |
| 2.667 | 95 | 91 | 53 | 68 |
| 3.0 | 1018 | 1047 | 777 | 840 |
| 3.333 | 1141 | 1103 | 954 | 1069 |
| 3.667 | 1290 | 1273 | 1480 | 1398 |
| 4.0 | 797 | 818 | 1296 | 1196 |
| 4.333 | 99 | 112 | 318 | 303 |
| 4.667 | 5 | 2 | 70 | 92 |
| 5.0 | | 1 | 46 | 49 |
| 5.333 | | | 54 | 37 |
| 5.667 | | | 3 | 7 |
| 6.0 | | | 15 | 21 |
| 6.333 | | | 44 | 63 |
| 6.667 | | | 30 | 36 |
| 7.0 | | | 20 | 21 |
| 7.333 | | | 7 | 5 |

## BLOCK MODE DMA PERFORMANCE
## EXPERIMENTAL PARALLEL IO MODULE

CONCURRENT DMA FROM MS630 TO MS630 THROUGH LOOPBACK

AVERAGE DATA RATE - 650,000 W/SEC FOR 8KW BUFFER

INTERPRETATION:

TWO INDEPENDENT DMA CHANNELS, EACH PERFORMING AT THIS RATE.

AGGREGATE DATA RATE = 2x650,000 W/SEC = 1,300,000 W/SEC = 2.6MB/SEC

DESIGN OF KA630 MEMORY CONTROLLER CONSTRAINS MEMORY READS AT 2.6 MB/SEC

EFFECT OF CONTENTION FOR MEMORY:

TOWERS OF HANOI BENCHMARK ACCESSES MEMORY AT MAX RATE

BENCHMARK RUNS IN 11.7 SEC IN ABSENCE OF DMA

BENCHMARK RUNS IN 13.8 SEC IN PRESENCE OF 640 KW/SEC DMA

## CONCLUSIONS - DMA THROUGHPUT

o MV II architecture minimizes contention between CPU and DMA devices

- Negligible difference between single transfer and burstmode performance
- 16% degradation in Hanoi benchmark under heavy block mode DMA load
- 2% degradation in blockmode throughput under Hanoi benchmark

o DMA Bandwidth is dependent on bus protocol used

- 1 MB/sec for DRx11-C devices in single or burst mode
- 2.6 MB/sec in block mode on experimental parallel IO board

APPENDIX B
Example program and CIN code

```
/*
Test of CIN Driver performance using VMS LEF as synchronization mechanism
Written by R. K. Somes, June 1986
Revised in September 1986 to lengthen histograms and to summarize them
*/


#include stdio
#include ssdef
#include errno
#include errnodef
#include iodef
#include timeb
#include secdef
#include descrip

#define    KWVOFFSET       010420
#define NULL               0
#define    KERNEL          3
#define USER               0

#define HISTO_LENGTH       2048

#define GO         1
#define MODE0              0
#define MODE1              2
#define MODE2              4
#define MODE3              6
#define RATE1M             8
#define RATE100K   16
#define RATE10K            24
#define RATE1K             32
#define RATE100H   40
#define RATEST1            48
#define RATELINE   56
#define MAINTST2   512
#define INTOV              64

globalref int ISRMON_BUF_DESC, ISRMON_CIN_ENTRY, ISRMON_CIN_MASK,
              ISRMON_CIN_FUNC;

globalref short int RNUM, TISR, TSTRT, TCNCL, UFLG, RQAST;

extern     char IOPAGE[8192];
static     long piopage[2];
static     long viopage[2];

struct
  {
  unsigned short int csr;
  short int bpr;
  } *kwv;
```

```
struct histogram
   {
   int npoints;
   unsigned short int basevalue;
   unsigned short int minvalue;
   unsigned short int maxvalue;
   unsigned short int outlyers;
   unsigned short int bins[HISTO_LENGTH];
   } isr_time;

struct large_histogram
   {
   int npoints;
   unsigned short int basevalue;
   unsigned short int minvalue;
   unsigned short int maxvalue;
   unsigned short int outlyers;
   unsigned short int bins[8*HISTO_LENGTH];
   } start_time, ast_time, ast_ret_time, cancel_time;

static $DESCRIPTOR(section_name, "IOPAGE");
static $DESCRIPTOR(device_name, "_KZA0:");

unsigned short int chan;

int efn, func, astadr, p1, p2, p3, p4, p5, p6, astparam[2], iosb[2];

int  status, acmode, flags, pagcnt, vbn, i, j, k;

int size;

FILE *file_id;
int *file_spec, *access_mode;
char rd_acs[] = "r";
char wr_acs[] = "w";
char file_name[] = "efn_data.dat";

int NLOOPS, timer_sw, user_ast(), ast_del_time, param, AFLG;
int lim90, lim99, lim999;

extern int TIMOFF(), TIMON();
main()
  {

/* Map the I/O page to the process */
  piopage[0] = IOPAGE;
  piopage[1] = IOPAGE+8191;
  acmode = KERNEL;
  flags = SEC$M_WRT+SEC$M_PFNMAP;
  pagcnt = 16;
  vbn = 0x100000;
  status = sys$crmpsc(     piopage,
                     viopage,
                     acmode,
                     flags,
```

```
                    &section_name,
                    NULL,
                    NULL,
                    NULL,
                    pagcnt,
                    vbn,
                    NULL,
                    NULL);

/* Prepare for test by soliciting the number of test measurements to be mad
and by initializing the histograms, where necessary */

  printf("MicroVAX II Interrupt Performance Data\n");
  printf("Complete test with AST and VMS Local EF delivery\n\n");

  printf("Number of test measurements? = ");
  scanf("%8d", &NLOOPS);
  printf("\nNumber of test measurements = %8d\n\n", NLOOPS);

  lim90 = NLOOPS-NLOOPS/10;
  lim99 = NLOOPS-NLOOPS/100;
  lim999 = NLOOPS-NLOOPS/1000;
/*Debug statment
  printf(" lim90 = %8d  lim 99 = %8d  lim999 = %8d\n\n",lim90, lim99, lim99
*/

  printf("Interval Timer (OFF = 0, ON = 1)? = ");
  scanf("%8d", &timer_sw);
  printf("\nInterval Timer = %8d\n\n", timer_sw);

  if(timer_sw == 0)
    {
    status = sys$cmkrnl(&TIMOFF, NULL);
    };

  start_time.npoints = 8*HISTO_LENGTH;
  start_time.basevalue = 500;
  start_time.maxvalue = 0;
  start_time.minvalue = 65535;

  isr_time.npoints = HISTO_LENGTH;
  isr_time.basevalue = 0;
  isr_time.maxvalue = 0;
  isr_time.minvalue = 65535;

  ast_time.npoints = 8*HISTO_LENGTH;
  ast_time.basevalue = 0;
  ast_time.maxvalue = 0;
  ast_time.minvalue = 65535;

  ast_ret_time.npoints = 8*HISTO_LENGTH;
  ast_ret_time.basevalue = 0;
  ast_ret_time.maxvalue = 0;
  ast_ret_time.minvalue = 65535;
```

```
  cancel_time.npoints = 8*HISTO_LENGTH;
  cancel_time.basevalue = 0;
  cancel_time.maxvalue = 0;
  cancel_time.minvalue = 65535;

/* Issue QIO to map and lock connect-interrupt buffer, and to start I/O.  C
is asynchronous, but does not modify the KWV11-C operating mode. User must
configure the option and enable interrupts by direct access to KWVCSR. */

  status = sys$assign( &device_name, &chan, NULL, NULL);

  p1 = &ISRMON_BUF_DESC;
  p2 = &ISRMON_CIN_ENTRY;
  p3 = ISRMON_CIN_MASK;
  p4 = &user_ast;
  p5 = &param;
  p6 = NULL;
  func = ISRMON_CIN_FUNC;

/* Clear the counter in preparation for the QIO call */

  kwv = IOPAGE + KWVOFFSET;

for(k=0; k < NLOOPS; k++)
  {

  kwv->csr = MODE0 + RATE1M;
  kwv->bpr = 0;
  kwv->csr = MODE2 + RATE1M;
  kwv->csr += GO;

  status = sys$qio(        NULL,
                 chan,
                 func,
                 NULL,
                 NULL,
                 NULL,
                 p1,
                 p2,
                 p3,
                 p4,
                 p5,
                 p6);


  kwv->csr = MAINTST2;

  status = bin_it(&start_time, TSTRT);

  RQAST = 1;
  UFLG = 0;

  status = sys$clref(efn);

  kwv->csr = MODE0 + RATE1M;
```

```
  kwv->bpr = 65436;
  kwv->csr = INTOV + MODE2 + RATE1M + GO;
  kwv->bpr = 0;

  status = sys$waitfr(efn);

  kwv->csr = MAINTST2 + MODE2 + RATE1M + GO;

  status = bin_it(&isr_time, TISR);

  status = bin_it(&ast_time, ast_del_time);

  status = bin_it(&ast_ret_time, kwv->bpr);

/*Measure I/O cancel time from zero base*/

  kwv->csr = MODE0 + RATE1M;
  kwv->bpr = 0;
  kwv->csr = MODE2 + RATE1M;
  kwv->csr += GO;

  status = sys$cancel( chan );

  kwv->csr = MAINTST2;

  status = bin_it(&cancel_time, kwv->bpr);

  } /* End of for loop */

  if(timer_sw == 0)
    {
    status = sys$cmkrnl(&TIMON, NULL);
    };

  file_id = fopen( file_name, wr_acs);

  size = sizeof(struct histogram);
  fwrite(&start_time, size, 1, file_id);
  fwrite(&isr_time, size, 1, file_id);
  fwrite(&ast_time, size, 1, file_id);
  fwrite(&ast_ret_time, size, 1, file_id);
  size = sizeof(struct large_histogram);
  fwrite(&cancel_time, size, 1, file_id);
  fclose(file_id);
  printf("                Min     Mode    90      99      99.9      Max  \n");
  printf("                                Pct     Pct     Pct            \n\n");
  printf("           +-------+-------+-------+-------+-------+-------+\n");
  printf("           |       |       |       |       |       |       |\n");
  printf("Start      |");
  show_it(&start_time);
  printf("           |       |       |       |       |       |       |\n");
  printf("           +-------+-------+-------+-------+-------+-------+\n");
  printf("           |       |       |       |       |       |       |\n");
  printf("ISR        |");
  show_it(&isr_time);
```

```
  printf("          |       |       |       |       |       |       |\n");
  printf("          +-------+-------+-------+-------+-------+-------+\n");
  printf("          |       |       |       |       |       |       |\n");
  printf("AST       |");
  show_it(&ast_time);
  printf("          |       |       |       |       |       |       |\n");
  printf("          +-------+-------+-------+-------+-------+-------+\n");
  printf("          |       |       |       |       |       |       |\n");
  printf("Process   |");
  show_it(&ast_ret_time);
  printf("          |       |       |       |       |       |       |\n");
  printf("          +-------+-------+-------+-------+-------+-------+\n");
  printf("          |       |       |       |       |       |       |\n");
  printf("Cancel    |");
  show_it(&cancel_time);
  printf("          |       |       |       |       |       |       |\n");
  printf("          +-------+-------+-------+-------+-------+-------+\n");

  }
bin_it( pointer, datapoint)
  struct histogram *pointer;
  unsigned short int datapoint;
    {
    if(datapoint >= pointer->basevalue)
      datapoint -= pointer->basevalue;
    else
      return(0);

    if(datapoint < pointer->npoints)
      pointer->bins[datapoint] += 1;
    else
      pointer->outlyers += 1;

    if(datapoint <= pointer->minvalue)
      pointer->minvalue = datapoint;

    if(datapoint >= pointer->maxvalue)
      pointer->maxvalue = datapoint;

    return(1);
    }

show_it( pointer)
  struct histogram *pointer;
    {
    int i, trials, modevalue, mode, time90, time99, time999, sum;

    modevalue = 0;
    sum = 0;
    time90 = 0;
    time99 = 0;
    time999 = 0;
    trials = pointer->maxvalue
    if ( pointer->outlyers > 0) trials = pointer->npoints;
    for(i=0; i <= trials; i++)
```

```
      {
      if(pointer->bins[i] > modevalue)
        {
        modevalue = pointer->bins[i];
        mode = i+pointer->basevalue;
        }
      sum += pointer->bins[i];
      if( sum <= lim90 ) time90 = i+pointer->basevalue;
      if( sum <= lim99 ) time99 = i+pointer->basevalue;
      if( sum <= lim999 ) time999 = i+pointer->basevalue;
      }
    printf("%6d |", pointer->basevalue+pointer->minvalue);
    printf("%6d |", mode);
    printf("%6d |", time90);
    printf("%6d |", time99);
    if (sum < lim999)
      printf("  **** |")
    else
      printf("%6d |", time999);
    printf("%6d |\n", pointer->basevalue+pointer->maxvalue);
}

user_ast( param )
   int *param;
   {
   kwv->csr +=MAINTST2;
   ast_del_time = kwv->bpr;
   status = sys$setef(efn);
   }
```

```
;=============================================================================
;
;USING THE KWV11-C FOR INTERRUPT LATENCY TESTING
;
;There is a way of using the KWV11-C for interrupt latency testing that is
;poorly documented but works well.  This method differs from the recommenda
;Bill Forbes in that it does not stop the clock, but accumulates time.
;
;1. Load the BPR with the 2's complement of the number of microsecs to wait
;   before an interrupt. This should be randomized.
;
;2. Set the CSR for INT OV, 1 MHz, mode 1, GO: ( MOVW #^O113,KWCSR )
;
;3. CLEAR THE BPR!
;
;4. On interrupt, hit the MAINT ST2 bit: (BISW #^O1000,KWCSR). Clock contin
;   to run so that next measurement can be made cumulatively.
;
;5. Read the BPR to get the number of microsecs between interrupt and hitti
;   the MAINT ST2 bit.
;
;=============================================================================

    .TITLE   ISRMON

    .LIBRARY /SYS$LIBRARY:LIB.MLB/

    $IDBDEF                  ; Definition for I/O drivers
    $UCBDEF                  ; Data structurs
    $IODEF                   ; I/O function codes
    $CINDEF                  ; Connect-to-interrupt
    $CRBDEF                  ; CRB stuff
    $VECDEF                  ; more


    .PSECT ISRMON_CIN        PIC,USR,CON,REL,LCL,NOSHR,EXE,RD,WRT


    .SBTTL  DATA STRUCTURES

; CIN_BUFFER       Data buffers for ISRMON


ISRMON_CIN_BEGIN:
RNUM::                     .BLKW   1        ;Randon number passed by user
TISR::                     .BLKW   1        ;ISR latency time
TSTRT::                    .BLKW   1        ;Start I/O latency
TCNCL::                    .BLKW   1        ;Cancel I/O latency
UFLG::                     .BLKW   1        ;User flag spin on
RQAST::                    .BLKW   1        ;Set by user to activate AST.
```

```
   .SBTTL  ISRMON_CIN_START, Start I/O routine

;++
; ISRMON_CIN_START - Starts the KWV
;
; 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)  - Process Virtual address of the CIN buffer (to be system mapped)
;  8(R2)  - Address of the IRP (I/O request packet )
;  12(R2) - Address of the device's CSR
;  16(R2) - Address of the UCB (Unit control block)
;
; Outputs:
;  none
;
;  The routine must preserve all registers except R0-R4.
;
;--
CIN_BUF_ADD = 4                              ;Address of CIN buffer
AST_PARM = 8                                 ;Offset to AST parmeter address
CIN_CSR_ADD = 12                     ;Address of CSR

ISRMON_CIN_START:
   MOVL    CIN_CSR_ADD(R2),R0        ;Get CSR address

   DSBINT
   BISW    #^O1000, (R0)             ; Hit MAINT ST2 bit to xfer counter
                                     ;   to BPR
   MOVW    2(R0), TSTRT              ; Xfer BPR content to TSTRT
   ENBINT

   MOVZWL  #SS$_NORMAL,R0            ; Load a success code into R0.
   RSB
```

```
    .SBTTL  ISRMON_CIN_INTERRUPT, Interrupt service routine

;++
; ISRMON_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's CSR
;  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
;
;--

ISRMON_CIN_INT:

    MOVL     CIN_CSR_ADD(R2), R0       ; Get KWV CSR address
    BISW     #^O1000, (R0)             ; Hit MAINT ST2 bit to xfer counter
                                       ;   to BPR
    MOVW     2(R0), TISR               ; Xfer BPR content to TISR
    MOVW     #1, UFLG                  ; Set user defined flag
    MOVZWL   RQAST, R0                 ; 1 = queue the AST, 0 = don't
10$:         RSB
```

```
   .SBTTL  ISRMON_CIN_CANCEL, Cancel I/O routine

;++
; ISRMON_CIN_CANCEL, Cancels an I/O operation in progress
;
; Functional description:
;
;  This routine turns off the KWV
;
; Inputs:
;
;  JSB interface
;
;  R2       -  Negated value of the channel index number
;  R3       -  Addr of current IRP
;  R4       -  Addr of PCB of canceling process
;  R5       -  Addr of the UCB
;
;  CALL interface
;
;  0(AP)    Arg count 4
;  4(AP)    Addr of channel index number
;  8(AP)    Addr of IRP
;  12(AP)   Addr of PCB
;  16(AP)   Addr of UCB
;
; Outputs:
;
;  The routine must preserve all registers except R0,R3.
;
;
;--

ISRMON_CIN_CNCL:
   MOVL    UCB$L_CRB(R5),R0                 ; Get Address of the CRB
   MOVL    CRB$L_INTD+VEC$L_IDB(R0),R0      ; Address of the IDB
   MOVL    IDB$L_CSR(R0),R0                 ; Get addr of KWV
   BISW    #^O1000, (R0)            ; Hit MAINT ST2 bit to xfer counter
                                    ;   to BPR
   MOVW    2(R0), TCNCL             ; Xfer BPR content to TCNCL
   MOVZWL  #SS$_NORMAL,R0           ; Load a success code into R0.
   RSB
```

```
    .SBTTL  ISRMON_CIN_INIT ; Dummy dev intialization routine
 ;
 ;  JSB interface
 ;
 ;  R0 -     Addr of UCB
 ;  R4 -     Addr of CSR
 ;  R5 -     Addr of IDB
 ;  R6 -     Addr of DDB
 ;  R8 -     Addr of CRB
 ;
 ;  CALL interface
 ;
 ;  0(AP)    Arg count 5
 ;  4(AP)    Addr of CSR
 ;  8(AP)    Addr of IDB
 ;  12(AP)   Addr of DDB
 ;  16(AP)   Addr of CRB
 ;  20(AP)   Addr of UCB

 ISRMON_CIN_INIT:
    RSB


   .SBTTL  ISRMON_CIN_END, End of Module

;++
; Label that marks the end of the module
;--
ISRMON_CIN_END:                        ; Last location in module

;Data structures used by the QIO

ISRMON_BUF_DESC::                              ;Buffer descriptor for CIN

   .LONG    ISRMON_CIN_END - ISRMON_CIN_BEGIN          ;Buffer length
   .LONG    ISRMON_CIN_BEGIN                           ;Address of buffer

ISRMON_CIN_ENTRY::

   .LONG    ISRMON_CIN_INIT-ISRMON_CIN_BEGIN           ;Init code
   .LONG    ISRMON_CIN_START-ISRMON_CIN_BEGIN          ;Start code
   .LONG    ISRMON_CIN_INT-ISRMON_CIN_BEGIN            ;Interrupt service routi
   .LONG    ISRMON_CIN_CNCL-ISRMON_CIN_BEGIN           ;I/O cancel routine

ISRMON_CIN_MASK::

   .LONG    CIN$M_REPEAT!CIN$M_START!CIN$M_ISR!CIN$M_CANCEL!CIN$M_INIDEV

ISRMON_CIN_FUNC::

   .LONG    IO$_CONINTWRITE
.END
```

# SPEAKEASY: AN INTERACTIVE DATA ANALYSIS TOOL FOR THE RESEARCH SCIENTIST

David H. Saxe
Speakeasy Consultant
Auburn, NH 03032

## 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 elementary data analysis situations.

## INTRODUCTION

Speakeasy is a conversational computer language in wide use as an interactive problem solving tool on VAX and IBM mainframes and IBM PC 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 of the language 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. Since these operators are generally written as Fortran functions, compiled and optimized code is executed and parsing overhead is minimized. Thus, for example, matrix inversion in Speakeasy is as efficient as that performed in a Fortran program. A Fortran preprocessor is used to allow extensions to be ported across mainframe and PC versions of Speakeasy.

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 elementary data 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 statisical 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 occuring 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.

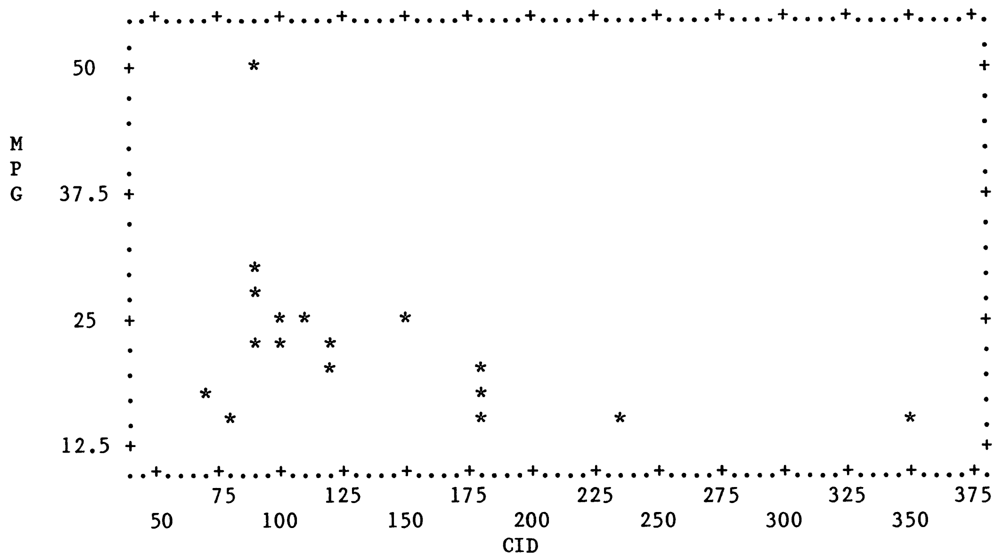
```
:_tabulate(mpg,carmodel,cid,cyl,trans,gears:title=head)

     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 X1/9 | 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
```

The interactive nature of Speakeasy becomes important when working with data to draw conclusions. After studying the following graph, variables in the above regression could be transformed to investigate log or inverse relationships between the dependent and independent data. Of course, Speakeasy supports many graphics devices, but the availability of reasonable "printer" graphics allows immediate presentation of a result without moving from a normal ASCII terminal.

```
:_graphz mpg cid
```

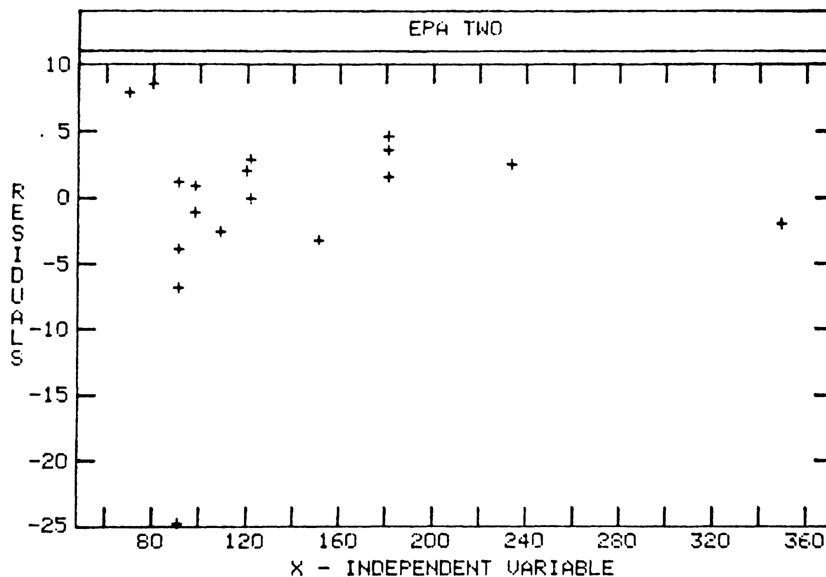



PROGRAM EXAMPLE

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
```




The plot above was done on a Tektronix terminal which has a movable cursor. The program could be easily modified to allow the interactive use of the cursor to identify and remove or otherwise study points from the regression.

## INTERACTION WITH A MODEL

Because Speakeasy is so mathematically oriented, it is very easy to create programs that allow the user to investigate a mathematical model. In the example shown below, the user's model is given on the left and the resulting Speakeasy program on the right. This initial program was written in a matter of minutes and drew a graph of the three functions using different line patterns with automatically scaled and labeled axes. The program was later expanded by adding about 20 lines to create the legends, axis titles and main titles. The resulting plot shown below was redrawn on a pen plotter in multiple colors for publication.

Original Handwritten Model Specification

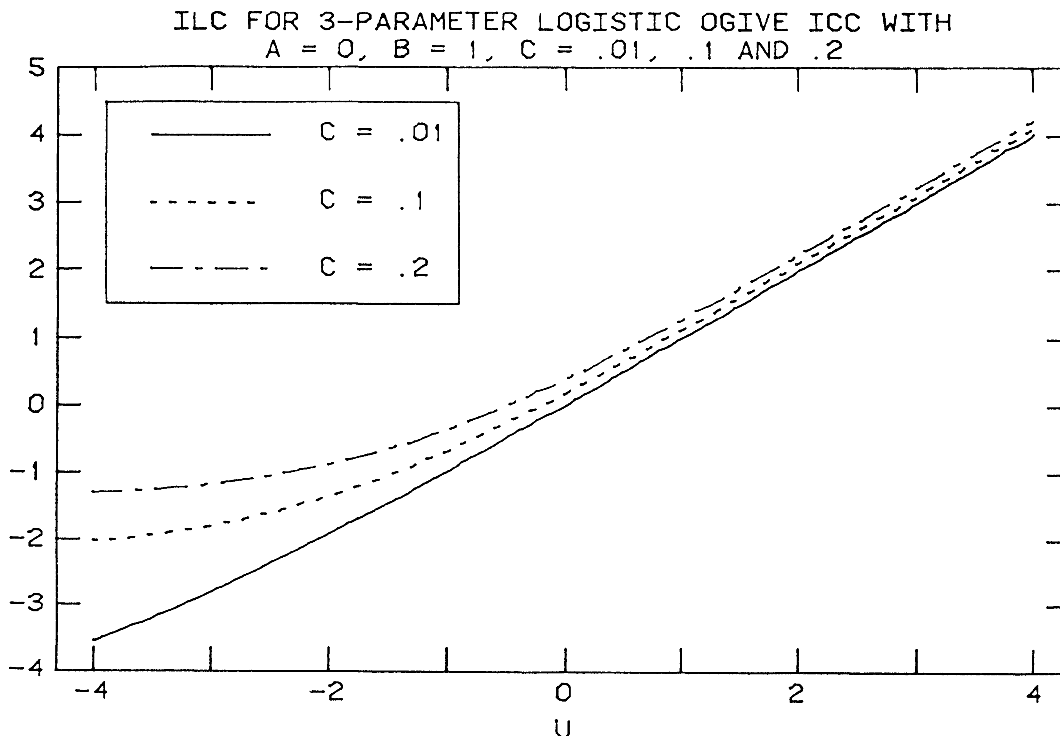
Plot $h_L(u,c) = \ln\left[\dfrac{c + (1-c)L(u)}{(1-c)(1-L(u))}\right]$

where $L(u) = e^u/(1+e^u)$

for $c = .01, .1, .2$

```
 1 PROGRAM
 2 X=GRID(-4,4)
 3 C=.01
 4 LOGOGIVE(HL,X,C)
 5 LINECODE=1
 6 GRAPH(HL:X)
 7 C=.1
 8 LOGOGIVE(HL,X,C)
 9 LINECODE=2
10 ADDGRAPH(HL:X)
11 C=.2
12 LOGOGIVE(HL,X,C)
13 LINECODE=3
14 ADDGRAPH(HL:X)
15 END
```

```
1 SUBROUTINE LOGOGIVE(L,X,C)
2 L=EXP(X)/(1+EXP(X))
3 N=C+(1-C)*L
4 D=(1-C)*(1-L)
5 L=LOG(N/D)
6 END
```



In a slightly more complicated example, Speakeasy was used to investigate a mathematical model of a liquid droplet surface. The formulation of the surface is done in a program that took only a very short time to create. Notice the correspondence between the formula and the Fortran-like program lines. The GRID operator causes PHI and subsequently derived variables to be arrays. The program was then used to experiment with differing values of epsilon and N. The model was derived from formulae using complex variables and could have been programmed in those terms since Speakeasy allows complex arithmetic.

Original Handwritten Model Specification

Plot projections onto the x-z plane of the surface given by:

$$r = 1 + \epsilon \sum_{j=0}^{n/2} \frac{(-1)^j \, n!}{(n-2j)!\,(2j)!} (\cos\theta)^{n-2j} (\sin\theta \cos\varphi)^{2j}$$

where n is an even positive integer and where

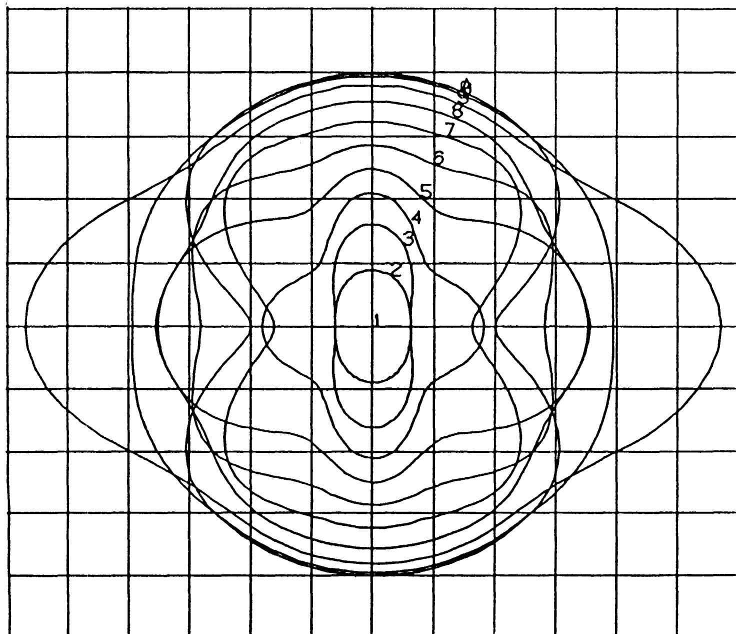
$0 \leq \theta \leq \pi/2$

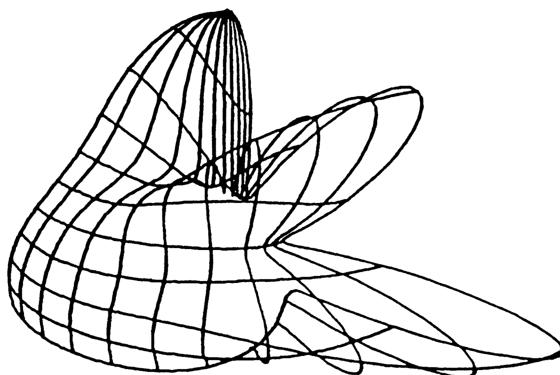
$0 < \varphi \leq 2\pi$

Try different values of $\epsilon$ and n.

```
 1 PROGRAM
 2 HENCEFORTH GRA IS GRAPH
 3 ALPHA="1234567890"
 4 VSCALE=-1.5, 1.5
 5 HSCALE=1023/780*VSCALE
 6 REQUEST N
 7 REQUEST EPS
 8 REQUEST DIV
 9 ZERO=.0001
10 PI=ACOS(-1)
11 PI2=PI*2
12 PI=PI/2
13 PHI=GRID(ZERO,PI2+ZERO)
14 CP=COS(PHI); SP=SIN(PHI)
15 DT=PI/DIV
16 FOR THETA = ZERO, PI+ZERO, DT
17 CT=COS(THETA); ST=SIN(THETA)
18 S = PHI - PHI
19 FOR J=0,N/2
20 DS=N!/(N-2*J)!/(2*J)!*(CT**(N-2*J))*((-1)**J)
21 DS=DS*((ST*CP)**(2*J))
22 S=S+DS
23 NEXT J
24 R=1+EPS*S
25 X=R*ST*CP
26 Z=R*ST*SP
27 GRA(Z:X)
28 TEXTPUT(ALPHA(IROW), Z(20), X(20))
29 HENCEFORTH GRA IS ADDGRAPH
30 NEXT THETA
31 X=GRID(VSCALE(1), VSCALE(2), .25)
32 LINES(X:X)
33 HARDCOPY
34 END
```

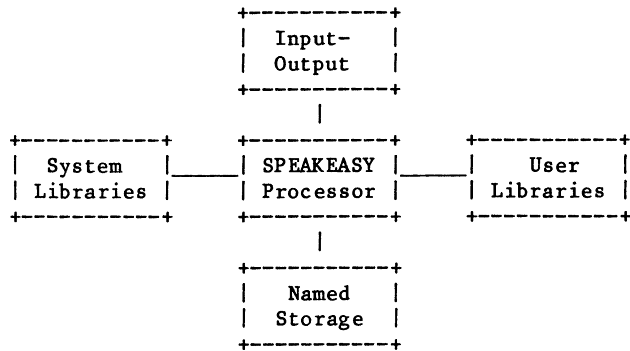
The graph on the left below shows the contours created by the program above. After determining epsilon and N values of interest, a second program was used to generate 3-space drawings of one octant of the surface. The drawing on the right below shows an octant of the surface for epsilon equal to .5 and N equal 10.





## 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 'FFFFFFFFFFFF7FFF'X and on IBM to Z7FFFFFFF 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 IV Reference Manual, 1984, Speakeasy Computing Corporation, Chicago, IL.

## ACKNOWLEGEMENTS

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 some of the the example sessions used in this paper. The ILC model is due to Paul Holland at Educational Testing Service. The liquid droplet model is due to Kerry Landman.

# Real Time Acquisition Using The C Language

**J-F Vibert**
Chu Saint-Antoine
Universite P. & M. Curie
Paris, France

### Abstract

The C language allows access to devices registers through pointers. Thus it is possible to write real-time programs using the C language instead of using MACRO-11. We have developed in C an acquisition program devoted to electrophysiological signals (evoked potentials) using an AD converter (AD11K), a programmable clock (KW11K) and a 16 bit parallel input/output board (DR11C) used to trigger external stimulators. Using extensively pointers manipulation to process the sampled data, it is possible to digitize the incoming signal each 50 microseconds on two channels and visualize simultaneously the data on a graphic display. Using C in real time environment allows rapid development of applications that would have took very long and tedious programming time using MACRO. Methods and examples are developed.

The C language allows access to devices registers through pointers. Thus it is possible to write real-time programs using the C language instead of using MACRO-11. Programs written in C can be as quick as those written in assembly language and much more easier to develop.

Our research field is related to neurosciences and we are involved in evoked potential signal processing. We have developed in C several acquisition programs devoted to the acquisition of electrophysiological signals. Two main type of signals had to be digitized and processed: acquisition of cortical evoked potentials using two channels with simultaneous graphic visualization and stimulator driving. In this case, the sampling interval ranged from 100 to 500 us on 2 multiplexed channels. The second was acquisition of unitary muscular potentials using one channel with also simultaneous graphic visualization and stimulator driving. Sampling interval ranged from 25 us to 100 us. Our hardware configuration comprised 2 computers on which these programs had to be run. The first was a PDP 11/34 with a floating point processor (FP11A), a dual programmable clock (KW11K), an analog digital converter (AD11K) and a general purpose parallel interface (DR11C). A French Tektronix like graphic processor Arinfo AF410, a RL01 and 2 RXO2 were the main peripherals. The second was a PDP 11/03 with a dual programmable clock (KW11K), an analog digital converter plus digital-analog converter (Analog-Device RTI 1250) AD11K) and a general purpose parallel interface (DRV11). A Tektronix oscilloscope served as graphic output, and 2 RXO2 were the main peripherals. Each of them runs RT11 V5.2.

These fully modular programs were entirely written in DECUS C using standards for portability. They are easy to modify for example to adapt to other machines and boards. In fact all these programs were children of a first one that derived from their parent for very little instructions and address values. They are fully documented in the code, and associated with an in-source documentation whose the getrno DECUS-C software tool reads and produces an UNIX like manual. Programs were developed and compiled under RSX11M V4.2 on a PDP-11/44, but linked and run under RT11, the target machines.

## Why C, a high level language for speedy acquisition?

Most of the usual high level languages does not permit to access device register, thus impairing the direct use of boards without drivers. C allowing access to physical addresses and thus to devices registers through pointers usage, it appears to be very suitable for real time acquisition, were AD/DA boards are to be used. Moreover, C allowing single bit operations, it provides an easy way to set or check the content of such board registers using masks and its set of logical bit operators.

One other advantage of the C language is represented by the fact that it produces an assembly code in readable form that can be checked and if necessary, manually optimized. This feature can be implementation dependent: DECUS C produces a *.S file that is then assembled by AS, while Whitesmith C produces a *.MAC then assembled by MACRO. The experience proved that this feature was only useful to verify that the produced assembly code was really optimized, and if not to understand how to use the C tips to improve it. The following examples will show

such an approach.

Since C is a high level structured language, it allows an easy way to program the man/machine interface that can be thought in terms of ergonometrics and explicit error recovery, that represents a tedious work when coding in assembly language. Last but not least, the source code developed in C is much easier to read than MACRO-11, and greatly facilitates the modification and maintenance of the program.

### Pointer usage

Pointers are the C features that allow a direct mapping of the program to the register board. Program 1 is given as an example of a function devoted to signal acquisition using AD11K and a KW11K boards. The comments will explain the process.

This piece of code demonstrates that programming in C for real time acquisition is very straightforward and the programs easy to read.

### Warning to compact C code

Reading the above code, the normal C programmer has probably remarked that compacting this code using the auto increment or decrement operators would produce better performance. Here was the major utility to look at the produced assembly code. In fact my first version of the same code was, as a long time C programmer, given as Program 2.

Those 10 lines of C code produced the following 24 instructions of assembly code:

```
mov $405,*_pkwcsr
.3:
;while(num_pts--)
mov r4,r0
dec r4
tst r0
beq .2
mov *-14(r5),*_padcsr
mov $_tab_can,-16(r5)
mov $2,-12(r5)

.5:
;while(n_can--)
mov -12(r5),r0
dec -12(r5)
tst r0
beq .4

.7:
;while(!(*padcsr & 0200));
bit *_padcsr,$200
beq .7

.6:
;*padcsr=p_can++
mov -16(r5),r0
add $2,-16(r5)
mov (r0),*_padcsr
;*p_don++=*padbuf
mov -20(r5),r0
add $2,-20(r5)
mov *_padbuf,(r0)
br .5

.4:
br .3
.2:
clr *_drout
mov $716,r4
.11:
```

while the code given in Program 3 give the following more compact 20 instructions of assembly code.

```
mov $405,*_pkwcsr
mov $4000,*_padcsr

.3:
tst r4
beq .2
mov *-14(r5),*_padcsr
mov $_tab_can,-16(r5)
mov $2,-12(r5)
.5:
tst -12(r5)
beq .4
add $2,-16(r5)
.7:
bit *_padcsr,$200
beq .7
.6:
mov *-16(r5),*_padcsr
mov *_padbuf,*-20(r5)
add $2,-20(r5)
dec -12(r5)
br .5
.4:
dec r4
br .3
.2:
clr *_drout
mov $716,r4
.11:
...
```

Thus, it is necessary to warn to the use of too compact C code, that necessitates temporary loading of registers and forward/backward movements of register values.

Another useful usage of pointers to board register mapping is the overlook to the keyboard status register, that allows to respond to any keystroke at any moment. During an acquisition process for example, it can be necessary to keep a possible control over the experiment in order to either stop or suspend it if something goes wrong

during the process. This could be done by a program such as given in program 4.

This mimics the INKEY$ instruction of some BASIC implementations, but is rarely possible so easily with other high level languages such as FORTRAN.

## Conclusion

Using C in real time environment allows rapid development of applications that would have taken long and tedious programming time using MACRO-11. Moreover, programs are much easier to modify and maintain.

Program 1

```
/*
 *
 *      DEMOACQ.C Demonstration function for acquisition with a
 *      prestimulus sampling , send of a stimulus, and post stimulus
 *      sampling
 *
 *      Author: JF Vibert
 *              CHU Saint Antoine
 *              Paris- France
 */

#define NPTS    512
#define NCAN    2
#define PRE_STIM        50

int     data[NPTS*NCAN],                /* data array */

        /* here are defined and initialized in octal
        the pointers mapping board registers */

        /* AD11K */

        *padcsr=0170400,                /* ADC csr address  */
        *padbuf=0170402,                /* ADC buffer address */

        /* KW11K */

        *pkwcsr=0170404,                /* clock csr address */
        *pkwpre=0170406,                /* clock buffer address */
        *pkwbuf=0170430,                /* clock counter */

        /* DR11C */

        *drcsr=0177530,                 /* DR11-C csr address */
        *drout=0177532,                 /* DR11-C output buffer address */

        /* DL11 */

        *pkbcsr=0177560,                /* keyboard csr address */
        *pkbbuf=0177562;                /* keyboard buffer address */

        /* Array defining the AD11K CSR contents for the 16 channels */

int tab_can[16]={040,0440,01040,01440,02040,02440,03040,03440,
                    04040,04440,05040,05440,06040,06440,07040,07440};
```

Program 1 (continued)

```
acquis()
{

        register        num_pts;            /* sample counter */

        int     pas,                        /* sampling interval */
                n_can,                      /* current channel */
                *p_chn,                     /* channel pointer */
                *p_can,                     /* pointer on the current channel */
                *p_don;                     /* pointer on sampled data */

        p_don=data;                         /* pointer on data array */
        p_chn=tab_can;                      /* start at the 1st channel */

                /* NB: a * before the pointer name signifies: the content
                of the location pointed by the pointer */

        *drout=02;                          /* set the DR11-C line 1 output to 1
                                             that silents our stimulator */
        *pkwpre=-(pas/10);                  /* time inter sample */
        *pkwcsr=0405;                       /* start the clock at 100 khz,
                                               this value comes from the
                                               KW11K user manual */
        /* prestimulus acquisition */

        num_pts=PRE_STIM;                   /* number of pts to sample prestim */
        while (num_pts)                     /* 1 sample by channel */
        {
                *padcsr=*p_chn;             /* sampled channel */
                p_can=tab_can;              /* initalise to the 1st channel */
                n_can=NCAN;                 /* max number of channel */

                /* in C, all non zero value is considered as true */

                while(n_can)                /* channels sequentially */
                {
                        p_can++;                    /* prepare next channel */

                        /* here is examplifified the use of a mask (0200)
                        and a logical bit operator (&): while the bit #7
                        is not set, we loop on the test. Note the semi-colon
                        just after the while, it represents an empty
                        instruction */

                        while(!(*padcsr & 0200));  /* 1st channel ok ? */

                        *padcsr=*p_can;             /* yes..next channel */
                        *p_don =*padbuf;            /* save sampled data */
                        p_don++;                    /* next data */
                        n_can--;                    /* next channel */
                }
                num_pts--;                          /* next sample... */
        }
```

Program 1 (continued)

```
*drout=00;                          /* bit 1 of DR11-C zeroed
                                    starts  stimulation */

/* post-stimulus acquisition */

num_pts=NPTS-PRE_STIM;                      /* number of samples after
                                            stimulation */
while (num_pts)
{
        /* Exactly the same the same things... */
}
*pkwcsr=012;                        /* stop clock */
*drout=02;                          /* reset the DR11-C output to 1 */
```

Program 2

```
...
        *pkwcsr=0405;                     /* start the clock at 100 khz*/

        /* here an auto decrement operator */

        while (num_pts--)                         /* 1 sample by channel */
        {
                *padcsr=*p_chn;                   /* sampled channel */
                p_can=tab_can;                    /* initalise to the 1st channel
                n_can=NCAN;                       /* max number of channel */

                /* here again an auto decrement operator */

                while(n_can--)                    /* channels sequentially */
                {
                        while(!(*padcsr & 0200));  /* 1st channel ok ? */

                        /* here auto increment operators */

                        *padcsr=*p_can++;         /* yes..next channel */
                        *p_don++ =*padbuf;        /* save sampled data */
                                                  /* next data */
                                                  /* next channel */
                }
                                                  /* next sample... */
        }
        *drout=00;                                /* bit 1 of DR11-C zeroed
                                          starts  stimulation */
...
```

Program 3

```
*pkwcsr=0405;                    /* start the clock at 100 khz*/
  while (num_pts)                        /* 1 sample by channel */
  {
    *padcsr=*p_chn;              /* sampled channel */
    p_can=tab_can;               /* initalise to the 1st channel */
    n_can=NCAN;                  /* max number of channel */
    while(n_can)                 /* channels sequentially */
    {
      p_can++;                   /* prepare next channel */
      while(!(*padcsr & 0200));   /* 1st channel ok ? */
      *padcsr=*p_can;            /* yes..next channel */
      *p_don =*padbuf;           /* save sampled data */
      p_don++;                   /* next data */
      n_can--;                   /* next channel */
    }
    num_pts--;                           /* next sample... */
  }
  *drout=00;                             /* bit 1 of DR11-C zeroed
                                         starts  stimulation */
```

Program 4

```
...
        for(;;)          /* infinite loop */
        {
                acquis();        /* here the acquisition function */

                if (!(*pkbcsr & 0200))   /* something typed */
                {
                        /* yes, test it in lower case */

                        switch(tolower(*pkbbuf))
                        {
                                case 'e': return;
                                case 's': suspend();
                        }
                }
        }
...
```

# An Investigation into the Use of ELN in a Multiprocessor Compute Engine

Tom Turano
Digital Equipment Corporation
Marlborough, Massachusetts

## ABSTRACT

As scientific computations become more complex, the time required for such computations on readily available computer systems becomes prohibitive. In an attempt to increase speed, computations are decomposed, where possible, into segments which can be computed in parallel. These segments are then allocated to separate processors. The increase in speed can be close to N times for N processors operating in parallel and requiring little or no communication.

ELN allows an ELN system image and application task to be downline loaded to diskless MicroVAXen. Each MicroVAX can act as an autonomous compute engine, passing data to another MicroVAX or to the host as required.

A "farm" of these MicroVAXen with ELN system images has been used to perform a Monte Carlo simulation of the movement of particles in a field-flow fractionation apparatus. This paper briefly describes the Monte Carlo simulation, and then it focuses on the configuration of the MicroVAX farm, the form of the decomposed problem, and the effect of the decomposition on the processing speed. Future experiments in this ongoing investigation are also discussed.

## 1 INTRODUCTION

As the complexity of scientific computing increases, the time required to complete these calculations also increases. This increase in time is not necessarily linear. A slight increase in problem complexity can result in the problem's becoming computationally prohibitive. The nature of science requires that computations become more complex, reflecting an attempt at a deeper understanding of the fine structure of the object under study.

In order to reconcile these two conficting goals - having a problem which is computable, while having enough detail to its calculation to answer the questions at hand, two distinct methods are employed. The first is to use a faster CPU. Although CPU speeds have consistently increased over the years, general purpose CPUs can not compete in terms of speed with special purpose CPUs. The distinguishing features of these special purpose computers is that they are very fast and very expensive.

The second method is to employ multiple CPUs and let all of them work on the problem in parallel (1,2,3). When a problem can be broken into several independent, or nearly independent, compute tasks, the method of choice is to partition the problem into these independent tasks and assign one of those tasks to each CPU of the multiprocessor system. In this way, the calculation speed can increase close to N times for a multiprocessor containing N CPUs. This second method will be discussed in this paper.

## 2 THE PROBLEM

A member of the LDP staff, Dr. Mark Schure, maintains an interest in a macromolecular separation technique called field-flow fractionation. Part of the research is modeling the behavior of groups of particles undergoing this separation (4). The model itself is a Monte Carlo simulation of the trajectories of particles moving under the fluid flow and centrifugal fields.

Specifically, the field-flow fractionation (FFF) apparatus consists of a tube positioned around the circumference of a centrifuge. Fluid is pumped through this tube. A mixture of particles of various sizes is introduced into the inlet of the tube and collected at the outlet (Figure 1). As the particles are carried through the tube, they are separated by size. This separation occurs because each particle experiences two forces as it passes though the tube.

The first is the force caused by net flow of liquid through the tube. The velocity of each particle depends upon how close the particle is to the walls of the tube. This is due to the parabolic form of the velocity profile of fluid moving

through the tube. The fluid closest to the tube walls moves the least, while the fluid at the center of the tube moves the fastest (Figure 2). Because of this velocity distribution, particles which spend more time at the center of the tube move through the tube most quickly.

The second force experienced by the particle is what is generally called centrifugal force. Although diffusion carries the particles both toward the center of the tube and toward the walls of the tube, the centrifugal effect (not a real force) causes the larger particles to experience a stronger tendency to move toward the tube walls. As a result, larger particles spend less time in the higher-velocity portion of the fluid flow, and so elute later than the smaller particles.

## 3 THE MONTE CARLO MODEL

The simulation of field-flow fractionation (FFF) uses a Monte Carlo technique whereby initially a random number is used to position each particle randomly within the velocity profile. To determine whether a particle is to diffuse randomly toward or away from the walls of the tube, another random number is used.

At each point in the simulation:

1. A direction is randomly selected (toward or away from the tube walls).
2. The effect on the particle trajectory of the velocity profile and centrifugal field at that point is determined.

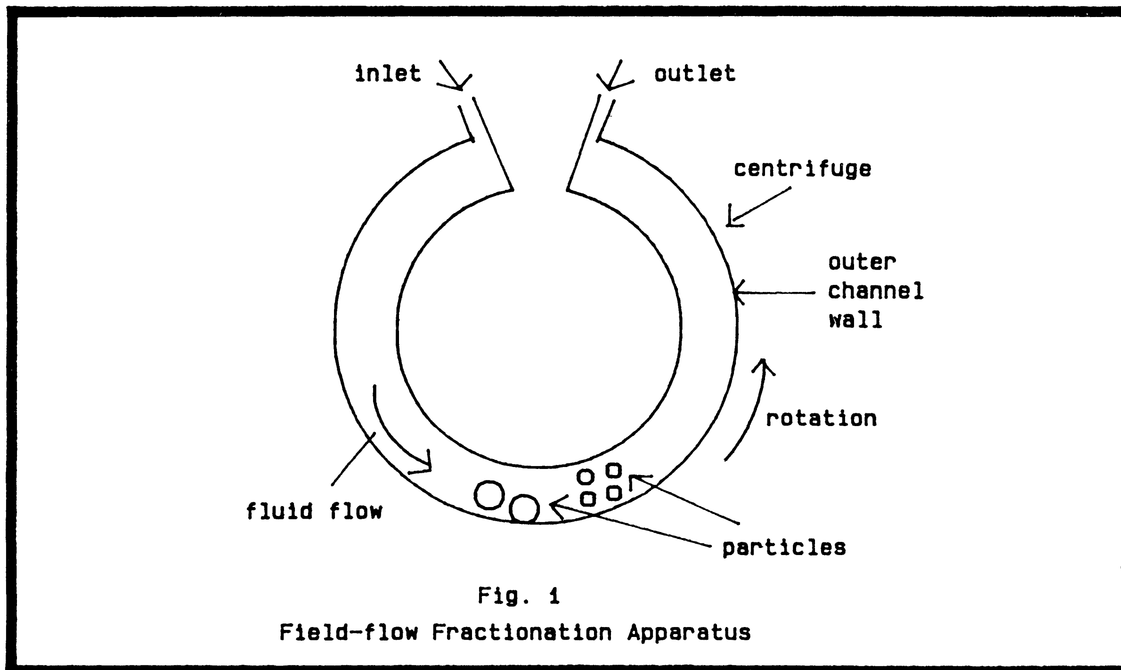



Fig. 1
Field-flow Fractionation Apparatus

3. The particle is moved to the new position dictated by the particle trajectory.

4. The steps are repeated.

Periodically, the time and location of the particle are written to disk.

## 4 SIMULATION ON A SINGLE CPU

The simulation was first run on a VAX 8600 computer which is capable of about a four-fold increase in compute speed over the VAX 780 (four VAX 780 equivalents). With the VAX 8600, approximately 3300 seconds were required to simulate 1500 seconds of FFF time or ~2.2 CPU seconds/FFF simulation second.

Therefore, to simulate 1000 particles (about 30 times the number simulated on the VAX 8600) would require 1.25 days of VAX 8600 CPU time. To achieve significant results for various parameter settings would require several weeks of time on a dedicated VAX 8600. It becomes obvious that a single processor of the VAX 8600 class would not be sufficient, and that the use of multiple processors might provide a reasonable solution to the problem.

## 5 MULTIPROCESSOR REQUIREMENTS

The application is ideal for a multiprocessor environment. First, each particle calculation is independent of any other particle calculation, so that the model can be completely and easily partitioned among the processors. Second,

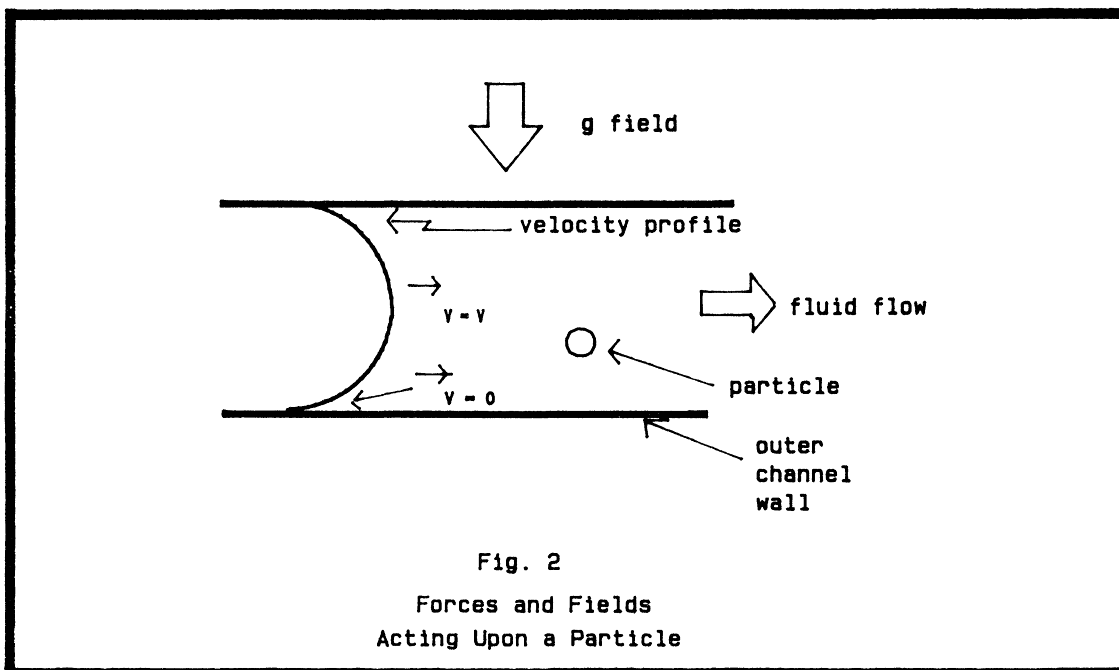



Fig. 2
Forces and Fields
Acting Upon a Particle

there is no requirement for communication between processors; and third, communication with the host is minimal.

With this type of application, an almost N-fold increase in performance can be realized by the partitioning of the problem across N processors. The increase is not strictly N times because the amount of communications required, although small, is still finite. As a result, there is a reduction in available compute cycles because of DECnet overhead. Further, as the number of processors accessing the ETHERNET increases, there will be increased contention for both the ETHERNET and the host disk.

## 6 THE VAXFARM

To investigate the possibility of using multiple MicroVAXen CPUs as a compute engine for scientific computation, we constructed a farm of six MicroVAX II class machines and two MicroVAX Is. We anticipated that this combination of CPUs would form a system which should operate at about 6.6 VAX 780 equivalents.

Each MicroVAX node consisted of a CPU, memory, and DEQNA. The amount of memory on each MicroVAX varied between one and four megabytes. The console panel for each MicroVAX was connected through a rotary switch to a single VT100 (Figure 3). In this way one VT100 could act as the console terminal for any MicroVAX simply by selecting the MicroVAX with the switch.

In an attempt to save physical laboratory space, we rack mounted four MicroVAXen BA23 boxes in each of two H9610 enclosures. The H9610 power supplies were connected to the same power controller so

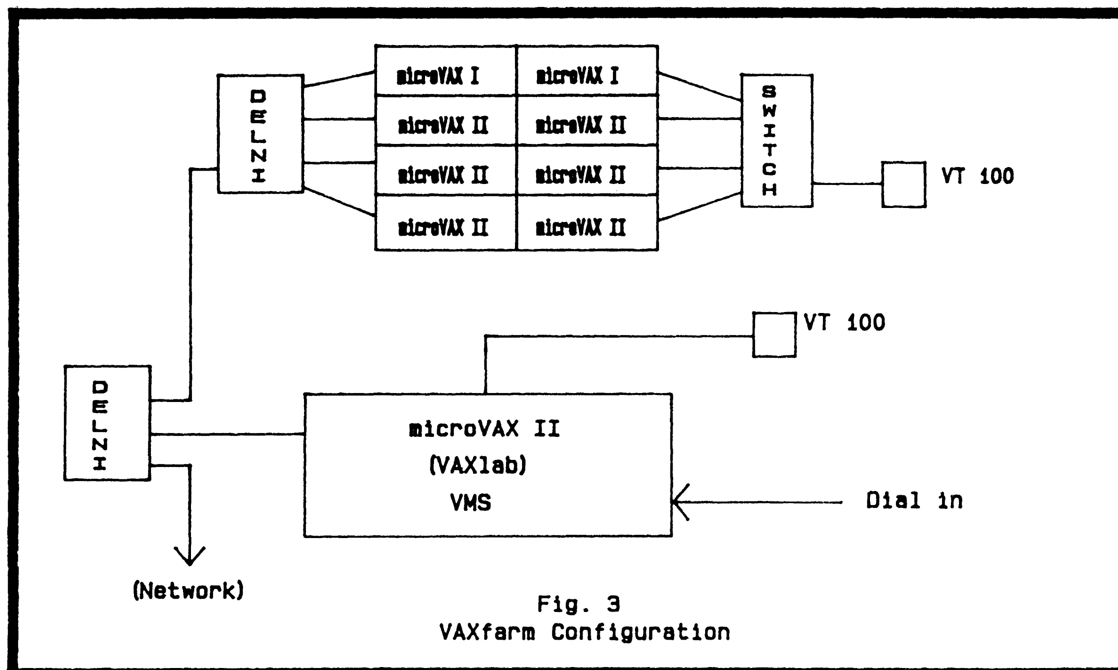


Fig. 3
VAXfarm Configuration

that the on/off switch mounted on one H9610 rack controlled the power supplies of both racks and permitted all six BA23 boxes to power up by pressing one switch.

The DEQNA of each MicroVAX was connected to one port of a DELNI. The output port of the DELNI, rather than connecting to an H4000 ETHERNET transceiver, was connected to an input port of a second DELNI. A second port on the second DELNI provided the ETHERNET link for the host system, a VAXlab. The host and the farm of MicroVAXen were thus interconnected by the equivalent of a private ETHERNET.

## 7 ELN

ELN is a subset of VMS which was originally constructed to provide real-time capabilities for the MicroVAX. Although an ELN image can be booted from disk, it does not require a disk. If no disk is present, ELN can be booted from ROM or across the ETHERNET. This means that the system image and the application must be memory resident. An ELN system image is built on the host machine, and the task to be run is imbedded within this image. The ELN image can be downline loaded to each node. In this way, a properly constructed application need only be built once to be used on each node of the VAXfarm. ELN also provides a remote debugger which runs on the host, but allows the program on the remote node to be debugged.

When each node of the VAXfarm attempts to boot, the node finds there is no disk or ROM load device and causes the DEQNA to issue a boot request across the ETHERNET. The network database on the host node is configured so that, when a boot request is received from a node address within the database, the host downline loads an ELN system image to that address and completes the boot.

## 8 THE SIMULATION

The simulation experiment consists of three programs. The first program is executed on the host and creates one parameter data file for each of the nodes of the VAXfarm. These files contain parameters which describe the system being simulated, such as particle density, centrifuge rotor speed, and fluid flow velocity These files also contain the random number seeds which will be used by the VAXfarm nodes to generate the random numbers used in the calculations. Each parameter data file has different random number seeds, so that no two nodes have an identical series of random numbers for their calculations. Once these files are created the host program terminates.

Upon power up, the second program of the experiment is downline loaded to each of the nodes of the VAXfarm. When this program executes, it first opens the parameter data file assigned to it on the host and reads the parameters for the

simulation, including the random number seeds. The program then closes the parameter data file.

Next, the VAXfarm node opens an output file on the host. The program begins its calculation. At certain points along the trajectory called breakpoints, the simulation time and the position within the tube are written to the output file on the host. When all the particles of the simulation experiment have completed their trajectories through the tube, the program closes the output file on the host and exits.

In case of irrecoverable error, the VAXfarm node opens, where possible, an error log file on the host. The VAXfarm node writes the cause of the fatal error to this file, and the program terminates. One reason there is no guarantee that an error file will be written is the possibility of a network file access failure. This error would also prevent the writing of an error log. The programs are written to terminate upon fatal error to allow the host to drop the DECnet links to nonproductive systems and thereby help reduce the traffic on the net.

Once all the VAXfarm nodes have completed their calculations, a third program on the host is used to analyze the data and produce elution profiles.

## 9 FUTURE PERFORMANCE EXPERIMENTS

Now that the system has been constructed and debugged, we intend to conduct several experiments to investigate its performance characteristics. First, we will run the simulation on from one to eight VAXfarm nodes and determine how the overall performance is affected as more VAXfarm nodes contend for the ETHERNET and the host disk. We will compare the overall performance of the VAXfarm against the same simulation executed on a VAX 750, VAX 780, and a VAX 8600. From this, we will determine how the system throughput deviates from that expected, and then attempt to determine where the bottlenecks exist.

## 10 CONCLUSIONS

A class of problems exist which exhibit the characteristics required for large granularity parallel computation. These characteristics include the problem's being naturally partitionable, requiring little communication between nodes and host, and not requiring the results of one node's calculations as the input for the calculations on another node. One of the problems in this class is the Monte Carlo simulation of particles under the influence of several fields.

We have constructed a loosely coupled group of general purpose processors which makes use of these characteristics to increase the computation speed of a Monte Carlo simulation. We do this by dividing the total number of particles to be simulated among the nodes of the group. In

this way a given node must perform fewer total calculations. Those which it does perform occur simultaneously with calculations performed on the other nodes of the group. The net result of such parallelism is an increase in the simulation speed without resorting to expensive high-speed processors.

## 11 ACKNOWLEDGEMENTS

I would like to take this opportunity to thank Susan Brown, the long suffering editor of my publications, Art Filz and Ivan Goddard, who actually constructed the farm, Jane Whitney who made the illustrations and Dr. Mark Schure who provided the incentive for building a parallel machine.

## 12 BIBLIOGRAPHY

1. Fast Special Purpose Computer for Monte Carlo Simulations in Statistical Physics, J.H. Condon and A.T. Ogielski, Review of Scientific Instruments, 56(9), September 1985, p. 1691.
2. A Special-Purpose Processor for the Monte Carlo Simulation of Ising Spin Systems, A. Hoogland et al. Journal of Computational Physics, 51, 1983, p 250.
3. A Fast Processor for Monte-Carlo Simulation, R.B. Pearson et al., Journal of Computational Physics, 51, 1983, p 241.
4. Separation of Coal Fly Ash Using Continuous Steric Field-Flow Fractionation, M.R. Schure et al., Environmental Science and Technology, Vol 19, 1985, p. 686.

# DATA MANAGEMENT SIG

# Data Normalization

Pamela A. Valentine
Senior Software Engineer

James M. Montgomery, Consulting Engineers, Inc.
P. O. Box 7009
Pasadena, California 91109-7009

## ABSTRACT

Normalization of data into physically separate and logical groups allows application developers to increase the integrity of their data, eliminate redundancy, reduce the size of the database and generally improve performance. Whether the application is of a business, scientific or engineering nature, normalized data is easily accessible thru programming languages and VAX Information Architecture products such as VAX-11 Datatrieve and Rdb/VMS. This paper is a tutorial on the process of normalization. By the use of practical examples, a step-by-step approach to normalization is shown. If you are an experienced application developer interested in improving the integrity of your data, or an inexperienced application developer just getting started with your database design, then this paper will offer some techniques to assist you.

### Introduction

While the actual steps of data normalization are fairly simple, obtaining the data that needs to be normalized is much more complex.

Looking at a lot of applications today (Figure 1) we find those that contain hundreds even thousands of files. The same data items appear in many files. This results in redundant and inconsistent data because the data resides in various stages of update. Also within these applications, many programs are virtually unmodifyable because of the lack of structure; the lack or absence of documentation; and because of their involvement with other programs within the application. A change to a program causes a dominos syndrome to the other programs within the application. This of course makes the programs very inflexible. If a manager wants a report presented in a different way, it is impossible without restructuring the files. Application reruns are also common because of application failures during update. Failure means starting the application over at the beginning because of the complexity of the updates to the files involved. To be successful in the management of data, computer files must be able to accommodate change with minimal impact.

These problems impact more than just the applications themselves. There are many corporations spending 80% of their programming budget in maintenance activities and only 20% in new development. Backlogs of work are growing to all time highs. Requirements keep increasing. Management wants better information for decision making. They want to play "what if" games with their data. They want greater productivity, faster response and of course increased profits. Government requests too are increasing for better audit procedures, more security and privacy controls.

Then there are the users with their many silent expectations. Of course they expect the applications to be fast and easy to build, but they also expect computer applications to be flexible. They do not understand why, after paying thousands of dollars for an application, they can't get the information they want. They also expect rapid results once a system is built. They cannot comprehend why a simple change to a report will take two weeks.

### Database Requirements

If Databases are a tool toward greater productivity, faster development and lower maintenance costs as advertised. If its true that with them comes better information for managers, up-to-date information any way its needed and of course faster service for the end-users then how do we get to this miraculous environment?

It is safe to say that a database will only be as good as its design. The construction of a database needs to be planned for the same reasons that the construction of an office building is planned. All of the pieces need to identified, documented and designed before construction starts.

Also the management of the data needs to be separated from the functions using the data. In most corporations, this will not be a simple task. Because information crosses political boundaries it needs to be managed from the top. Management needs to recognize its data as a corporate resource just like its people and equipment. It also needs to recognize that rivalries over data will result in the loss of profits.

Corporate organization charts do not show the complex relationships between managers and departments. It is vital for Data Processing personnel and Management to recognize up front that more databases have failed due to human problems and corporate politics than from the lack of available technology.

To migrate to a database environment we must have standards. Not only programming but data standards. It is essential to know that any piece of data will be the same throughout the database. The data needs to be independent. Changes to the data should not cause programs to change. The database needs to have data integrity both in recovery and accuracy. Its not only important to know that a piece of data is the right data but also if the system crashes that we can get back to where we were when the system crashed. The database must provide sharable data to the expert and novice alike and allow for varying formats in the data so that a change to a report will take minutes rather than weeks. Performance is a key issue today in the database environment. Not only in response time, but in the reliability of the hardware and the software. If either is unstable, the other is of little value.

## Data Modeling

The data modelling effort can be divided into two parts (Figure 2). First the physical or the way in which the data are physically stored in the computer and second the logical or the way that the end-user will see the same data.

Separation of the physical storage allows the database to be machine independent so that the database designer can select the best model for the application. The three most popular physical formats are the Hierarchical; Codasyl or Network; and the Relational. This by no means implies an all or nothing choice for the database designer. It is certainly possible and sometimes desirable to split the database into various formats to obtain the desired results. Each physical format has strengths and weaknesses and it is important for the database designer to evaluate all the physical models and select the one(s) best suited for the particular application.

The Hierarchical model (Figure 3) orders the data items in a top-down structure. In this model each record in the hierarchy is linked with one record in the next higher level of the hierarchy. This makes searching for data relatively fast but modifications to the database have to take into consideration the lines of connection.

In the Codasyl or Network model (Figure 4) records are grouped into sets and a record can be part of more than one set. This model permits more complex links between data items however modifications to this type of database can be even more complex than the hierarchical format.

The Relational model (Figure 5) does not use a pointer structure. Instead all the data are represented as rows (records) and columns (fields). The records contain fields that allow associations with the other fields in that record. The relational model is the easiest to understand and maintain because of its tabular form and its lack of pointers. The order of the records however are arbitrary and can be slow if performing sequential searches thru the records.

Thousands of each physical type currently exist the hot debates over which is best continue. While the differences can be confusing, they are advantageous to the designer in selecting the best model for the application.

The second part of the modeling effort is the logical or make believe record structures. The logical model will involve defining, organizing and documenting hundreds, even thousands, of data items. This will often be a very difficult and time consuming task because of the way the data has been treated in the past. The same data item will have been defined differently in different places, even given different names. The data administrator will need much help from the users in clearing up the confusion.

Involving the end-user in defining the data elements and their associations before the implementation of the database will greatly improve the final product. Because they know their data better than anyone else, they need to be involved from the bottom up and the Data Processing group needs to make it easy for them to help. Tools are needed to allow effective communication between the end-user and data processing.

There are many available methods for use in getting data descriptions from users and bubble charts are only one way. Using them, however, provides a way of looking at the data that is independent of hardware and software and they are easily understood by the end-users. In addition to being easy to understand, this method produces three very important results. It will give the database designer the primary key, the second key and all the attributes of the entities within the database.

The first step is to define the units of data (Figure 6). Reduce the items to their lowest form or the atom of data in that the data item cannot be divided into smaller data items and retain any meaning. For example, the data item "salary" is defined to be a monthly dollar amount paid to an employee. We cannot divide this data item into smaller data items which by themselves are meaningful. On the other hand, the data item "name" is defined to be the full name of an employee. This item can and should be divided into first name, middle name and last name data items.

Any single data element by itself is not very interesting. Only when it is associated with another data element does it becomes useful. For example, the data item "Salary" is only interesting when it is associated with a particular employee (Figure 7).

There are only three type of data associations; one to one; one to many; and many to many. These associations can be represented using two types of symbols (Figure 8). First the single arrow. Drawing that arrow from one data element to another represents a one to one relationship. In other words, any one employee number has only one salary associated with it. And second, the double arrow. Shown from one to another represents a one to many relationship. In this example any one employee can work on zero to many projects. The double arrow drawn in both directions will represent the many to many relationship.

For the database, the designer needs to combine all the users' views together. If we use the two previous examples as two user views of their data, one interested in salaries and one interested in projects. Their views can be combined together eliminating the duplicates and redundancies. When complete, the element looks like Figure 9. The redundant employee number data item has been thrown out and the remaining employee number data item points to both the salary and projects data items. This of course needs to be done for all the views of the data.

There are data items that map in both directions. Figure 10 shows that an employee can belong to only one department but many employees can belong to any one department. To illustrate this, draw the single arrow from employee to department and the double arrow from department to employee. If knowing how many employees work on a given project will be important to the database then a double arrow needs to be drawn between employee number and project number.

Combining all the user views together will result in many groups or entities of data items that look like Figure 11. From this entity the keys can be determined. A primary key will be a bubble with one or more single-headed arrows leaving it. A secondary key will be a bubble with one or more double-headed arrows leaving it and a nonprime attribute will be a bubble with no arrows leaving it.

From these data element groups we can derive the logical (make believe) record structures that will be used by the programs.

**Normalization**

Once the data element groups are understood the steps to normalization are fairly simple. First we begin with the unnormalized data structure. The structure is put into first normal form, then into second normal form and finally into third normal form.

The problems encountered with unnormalized files are numerous. Figure 12 shows a record definition as it would be written based on the record structure shown in Figure 13. This type of structure limits the project data to a set number of occurrences. Increasing this number of occurrences requires a file restructure. Also modifications would require searching compound keys to find the right record. Both the employee number and project number would be required to find the right record before the modification could take place. This design also requires redundant data to be carried along. The project title and the project engineer is duplicated in every employee's record working on that project. In addition to wasting file space, modifications to the project title or the project engineer for a project requires searching thru the entire file for all the occurrences of the data.

These problems can be solved thru normalization. Begin by putting the data into first normal form (Figure 14). To do this, very simply remove the repeating groups of fields into another structure and carry along the keys. The one structure therefore becomes two.

The problems with first normal form files are in additions, deletions and modifications. With this design, a project cannot exist without an employee. This would impose a requirement on the database of having an employee working on a project before the data could be entered into the database. Also an employee termination can result in the loss of the project data if the terminated employee happens to be the last employee working on that project. Modifications require searching part of the compound key. In this example, both the employee number and project number need to be searched before modifying the record.

Next put the data into second normal form by checking for functional dependencies. A data item (A) will be functionally dependent on another data item (B) if at every moment of time A has only one value in B associated with it in a record (R). Saying that B is functionally dependent on A is the same as saying that A identifies B. If we know the value of A we can find the value of B. In figure 15 we see that the project number data item is functionally dependent on employee number because to find out which project an employee is working on would be accessed via the employee number. Employee number however is not functionally dependent on project number because more than one employee could be working on the same project. Project Title and Project Engineer however are functionally dependent on project number so we remove those fields that are not functionally dependent on the primary key into another structure (Figure 15) and again carry along the keys.

Still problems exist with second normal form files (Figure 16). Department data cannot exist without an employee. The termination of the last employee in a department deletes the department data. Also changes to department information requires searching every record. Changing the name of a department from Administration to Corporate Administration for example requires searching every employee's record for the department name.

To solution to this problem lies in the removal of these transitive dependencies. In Figure 17 we have a data item (Department Number) that itself identifies two other data items (Department Name and Department Manager). To remove this transitive dependency we split the record in two

as shown. The department number item is left in the original record in order to tie the records together when needed.

Using third normal form files will provide data groups that are the least likely to cause maintenance problems or application programs to be rewritten.

While its true that other forms of normalization are possible they are not normally required. Also it should be noted that there will be rare times when normalization of data causes performance problems or when putting data into another normalized form is best for the particular application. When deviations are required, it is best to design in third normal form, investigate the deviations and completely document the exceptions.

**Data Dictionary**

Another kind of normalization that should be considered for the database is the use of a data dictionary for the data descriptions. A great deal of effort is being spent on making sure that the actual data is not redundant, but the actual data descriptions or record definitions are being repeated in hundreds of programs.

Using a data dictionary such as the Common Data Dictionary (CDD) on the VAX can provide tremendous benefits in defining, documenting and organizing the data descriptions for the database. Using the CDD will enforce standards for field and record names; field sizes will be set and consistent within programs and changes to the size of a field will require only a recompile of programs rather than editing each program and then recompiling. In addition to size, the CDD provides other standards for field definitions. By defining fields as separate objects in the dictionary, they can be used in all record definitions using the "Copy" statement provided by the Common Data Dictionary Language (CDDL). These field definitions can provide standard Column Headers for reports; standard picture strings, default values, edit strings and condition names for various programming languages; and documenting text inside of the fields and records.

Data dictionaries also provide an additional level of security in the form of Access Control Lists that are separate from the Access Control Lists found in Digital's Command Language (DCL).

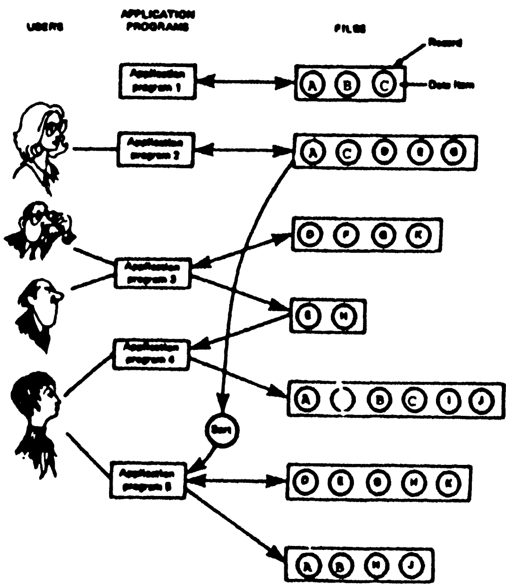
The CDDL has its own compiler. This will provide history lists of the programs using the records. The CDDL also provides a recompile qualifier that will force updates to the program from the new field fragments.

As the number of databases grow, the number of applications using them can grow disproportionately. In many cases it will become easier to rewrite old applications using the databases than trying to maintain them. The value of the stored data therefore will increase over time.

The percentage of time spent in maintenance will be reduced; the productivity and the rate of application development increases with the installation of the databases; and most importantly profits are increased.

Backlogs will be reduced because it takes less time for the application programmer to get the information needed. The end-users dependency on programmers is reduced because of their direct access to the data. Also as the knowledge of the end-user grows and as their involvement in the data processing environment increases, their understanding of the true effort involved in building a database will increase thereby making the next database effort with them much easier.

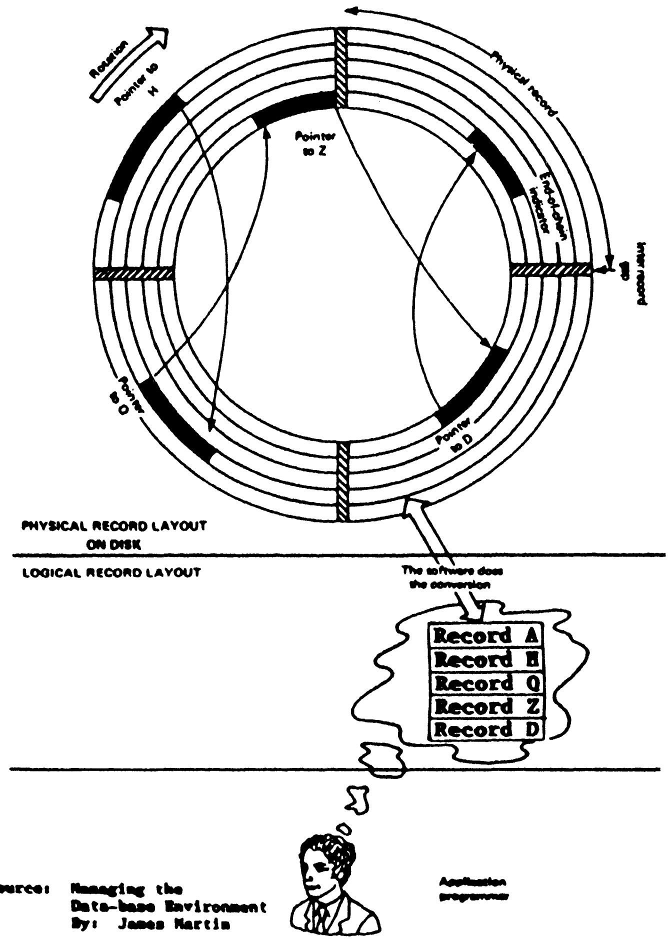
Successful Data Management isn't easy. It presents new management challenges and it requires the commitment of top management, end-users and data processing working together toward the common goal.




**Source:** Managing the
Data-base Environment
By: James Martin

Figure 1

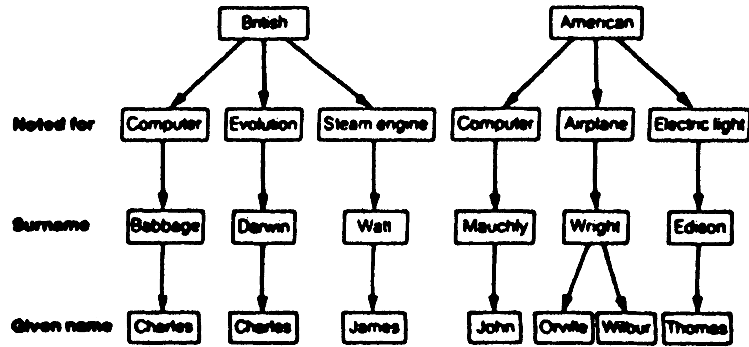
**Data Modeling**




**Source:** Managing the
Data-base Environment
By: James Martin

Figure 2

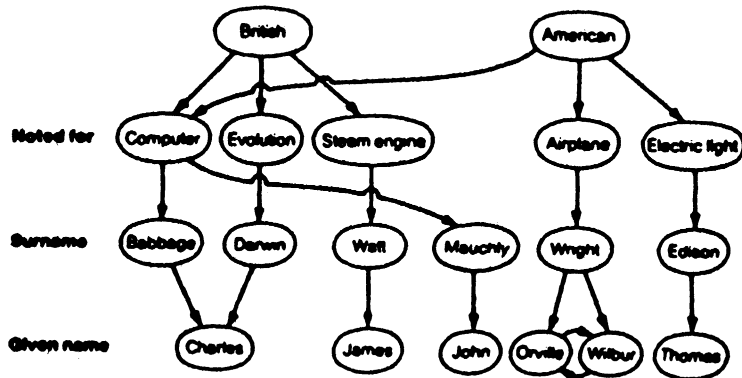
## Hierarchical




**Source: High Technology Magazine**
**December 1984**

Figure 3

## Codasyl or Network




**Source: High Technology Magazine**
**December 1984**

Figure 4

## Relational

| Surname | Given name | Nationality | Noted for |
|---|---|---|---|
| Edison | Thomas | American | Electric light |
| Babbage | Charles | British | Computer |
| Wright | Orville | American | Airplane |
| Mauchly | John | American | Computer |
| Darwin | Charles | British | Evolution |
| Watt | James | British | Steam engine |
| Wright | Wilbur | American | Airplane |

Compound key

**Source: High Technology Magazine**
**December 1984**

Figure 5

# Data Elements

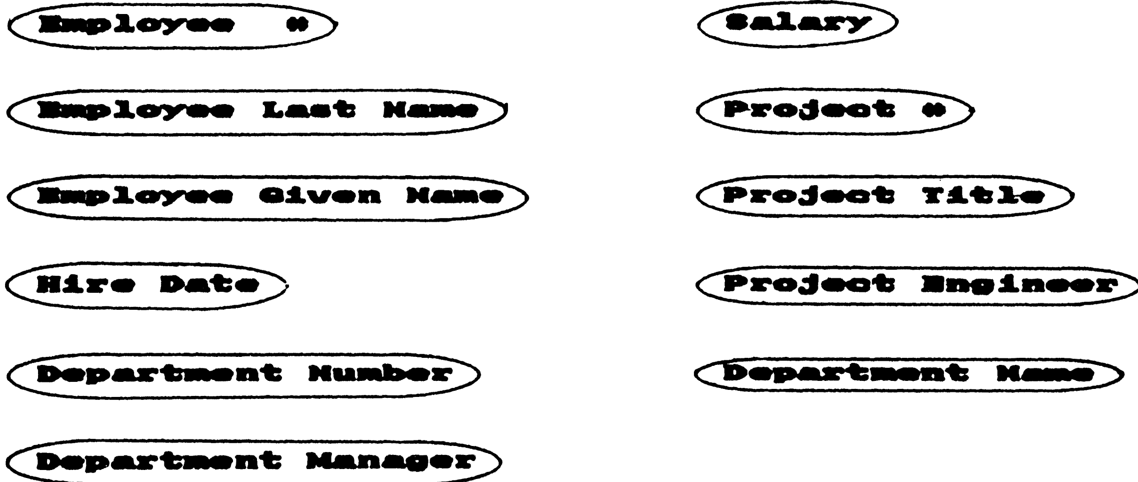



Figure 6

# Data Element Associations

Figure 7

# Data Element Links

Single Arrow

Double Arrow

Figure 8

## Combining Elements

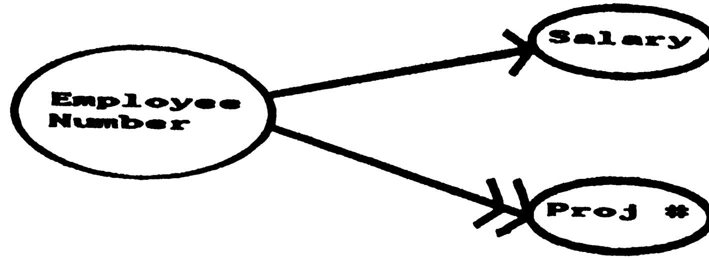



Figure 9

## Reverse Associations

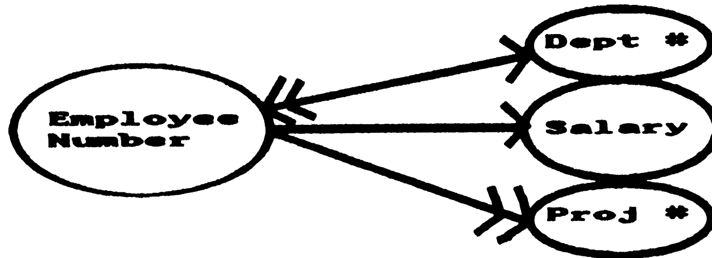



Figure 10

### Keys

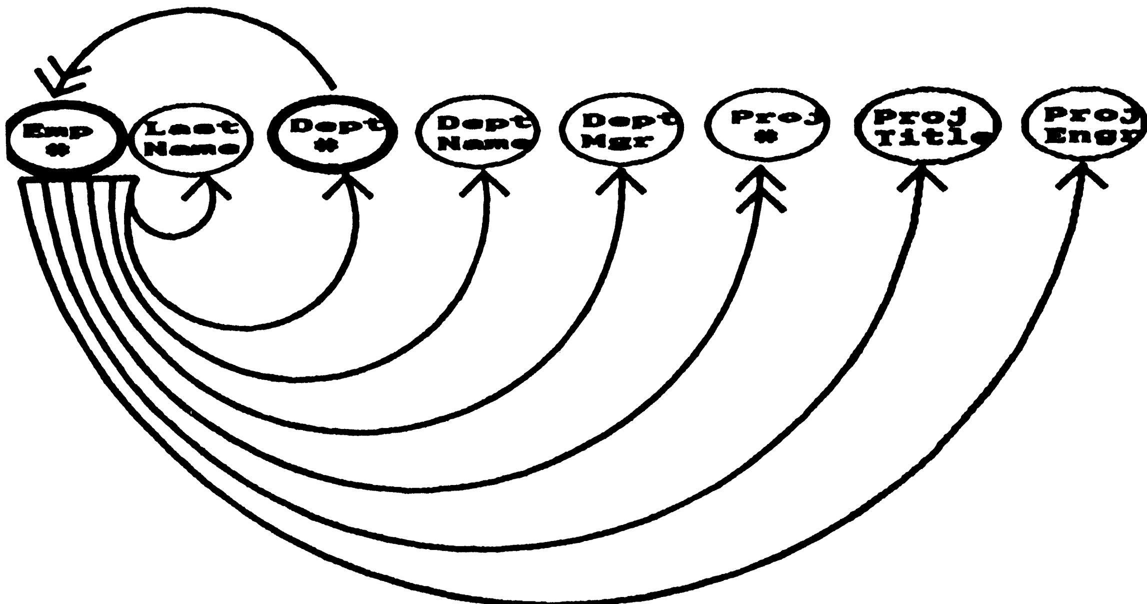



Figure 11

**Record Definition**

```
01    EMPLOYEE-RECORD.
      03    EMPLOYEE-NUMBER PIC 9(5).
      03    LAST-NAME PIC X(20).
      03    DEPARTMENT-NUMBER PIC 9(6).
      03    DEPARTMENT-NAME PIC X(60).
      03    DEPARTMENT-MANAGER PIC 9(5).
      03    PROJECTS OCCURS 3 TIMES
            05    PROJECT-NUMBER PIC 9(9).
            05    PROJECT-TITLE PIC X(60).
            05    PROJECT-ENGINEER PIC 9(5).
```

Figure 12

**Unnormalized**

Figure 13

**First Normal Form - Eliminate Repeating Fields**

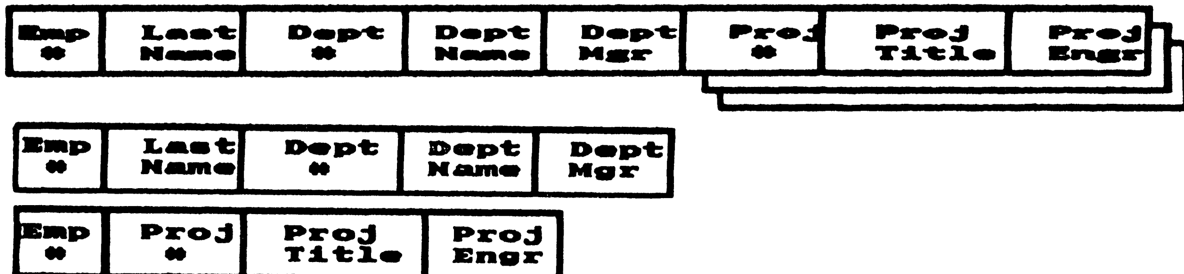



Figure 14

**Second Normal Form**
**Check Functional Dependencies**

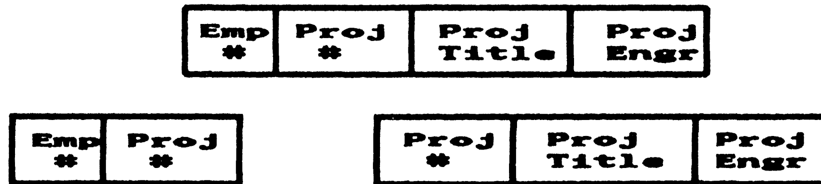



Figure 15

**Second Normal Form**

Figure 16

**Third Normal Form**
**Remove Transitive Dependencies**

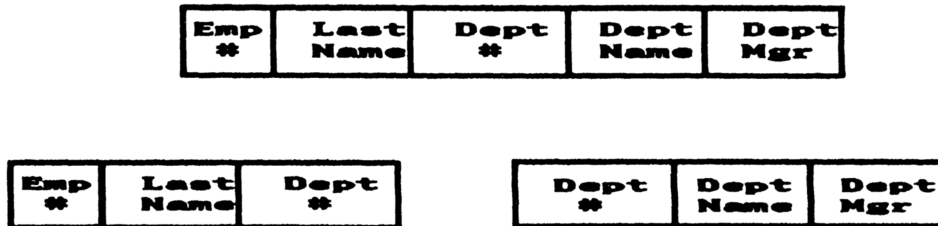



Figure 17

# DATA ENGINEERING

James R. Yoder
Sandia National Laboratories
Albuquerque, New Mexico 87185

ABSTRACT

Data Engineering can be defined as the application of computing, science, mathematics, communications, and other engineering disciplines to the transformation of data into information. A Data Engineer is a member of a group devoted to data engineering or is one who is trained in, or follows as a profession, the practice of data engineering. This paper describes the application of data engineering to the development and operation of a large VAXcluster based product validation data system. The system has many sources spread over the country, hundreds of users, and several important constraints. Data engineering is required to assure continuity of data acquisition, transmission, storage, retrieval, and analysis. A large volume of data must be reduced and analyzed with user friendly, interactive tools that provide broad capabilities in signal processing, statistics, and graphics. The use of artificial intelligence (expert systems and pattern recognition) will be a significant aid to improving system utility.

## INTRODUCTION

A substantial body of literature under the generic heading of Data Engineering exists to define, from an engineering point of view, the technology and facilities required to manage information [1,2]. Much of the motivation for the relational database model stems from problems presented by the manipulation and analysis of scientific and engineering data [3,4]. In contrast to an exposition of theory or technique, this paper is intended to describe some of the underlying professional staff requirements necessary for the construction, integration, and implementation of large technical data systems. The professional staff requirements for the development of administrative databases are reasonably well known. However, the development of technical data systems requires academic backgrounds and experience that are not typically available in administrative database projects. Ultimately, this paper will argue that the extended spectrum of background and talent required to develop and integrate such data systems can be enveloped into an embryonic discipline called Data Engineering and that those who practice Data Engineering should be called professional Data Engineers (although no "legal" certification is implied). A paradox will result when we find that no single academic track will provide an adequate background for an effective career in Data Engineering. The concepts will be illustrated by an example: the design, development, and operation of the Product Test Data (PTD) system in a VAXcluster environment at Sandia National Laboratories.

* Work supported by U. S. Department of Energy under Contract DE-AC04-76DP00789.

### A Note on Terminology

In general, we will use the term "data system" to include the entire process of acquiring, storing, and manipulating data. A data system might include a "database" - a structured repository for the data. A Database Management System (DBMS) is a program (perhaps acquired from commercial sources) that might be used to handle the database.

### Administrative vs. Technical Data Systems

In order to develop the thesis of this paper, we should draw an ad-hoc distinction between two general classes of computer based facilities: administrative databases and technical data systems. Administrative databases are those operated in support of the management and administration of an organization. For this paper, we will define technical data systems to be computer based information systems operated in support of scientific or engineering projects. Although most technical data systems share similar requirements and constraints, we will, by example, further restrict our domain to non-graphic (i.e., non-CAD), large scale, analysis-oriented engineering data systems (e.g., test data, environmental stimuli, statistical quality control, material characteristics).

The essential value of corporate databases is clearly recognized by modern management. Information is critical to the function of most industries and, for a growing number, information is their sole product. For the most part, the technology required to develop and operate corporate databases (personnel, finance, library, etc.) is well-in-hand and the supporting talent, although often in short supply, can be clearly identified. On the other hand, technical databases require contributions from several

disciplines that are not often associated with traditional database development or administrative computing projects. The thesis of this paper is that a new discipline, data engineering, is required to supplement the effort of computer scientists, programers, and hardware engineers to bring about the successful development of data systems that support scientific or engineering endeavors.

At the risk of over-simplification, one could state that administrative databases require the acquisition, transmission, storage, and retrieval of source data. The data are often amalgamated and restructured for analysis and reporting, but there is rarely a requirement that data be transformed in order to be understood. In contrast, it is very often necessary in engineering data systems to apply some kind of transformation so that data can have meaning. Further, the product of administrative data systems is typically a standard form, report, or transaction (e.g. a paycheck) based upon well understood conventional algorithms while the product of an engineering data system is most often a judgement based upon manipulation of data in a problem driven, ad-hoc manner. In addition, the final presentation of results can require the application of esoteric, narrowly understood mathematical transformations or computations.

We will explore these requirements and their effect upon the selection and development of the staff required to create and operate engineering data systems. Next, an example of such a system will be given along with the staff and management considerations. Finally, we will conclude that a newly defined discipline, Data Engineering, is a necessary and vital profession, essential to the successful application of computers to the problems of science and engineering.

## ENGINEERING DATA SYSTEM REQUIREMENTS

Before considering the professional requirements for the development and operation of engineering data systems, lets examine the nature of these systems in a little more detail.

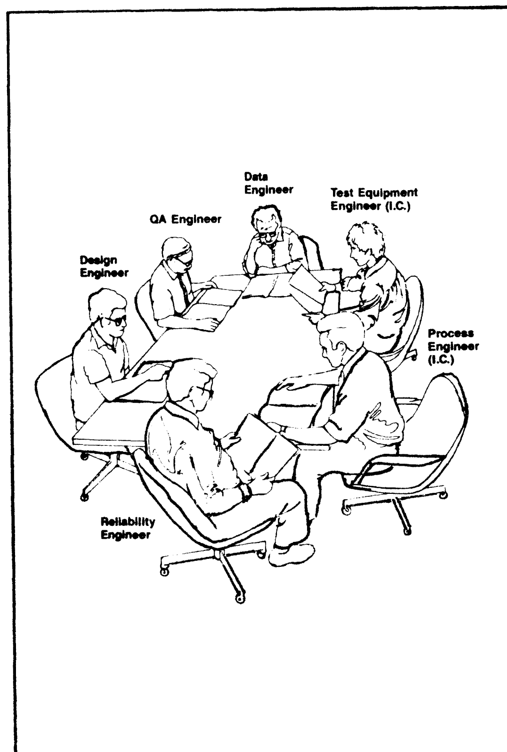
### Data Determination

Although scientific data systems may be ad-hoc in nature, engineering systems tend to be archival, corporate resources. This requirement leads to the use of the same data by several users and to use of the same data for many purposes. Often the individual responsible for generating the data is not the same person that needs the information. In many cases, the use of the data may be somewhat speculative and in the distant future. It is necessary, in these instances, to assure that all potential needs for information are considered as the data system for a given project is formulated. In the event that the cost of data acquisition is high (e.g., data from one-shot destructive tests), or when the data have a long term value, this a-priori consideration, which we might term "data determination", should be the product of multiple judgements under the guidance of an experienced data czar.

### Data Acquisition Systems

The requirements for engineering data systems often include the acquisition of data from measuring systems whose construction and operation, by itself, is somewhat complicated. The data acquisition system might be operated in a severe, hostile environment (e.g., flight test, reactors). The difficulty of acquiring any data at all in these situations may be compounded by the sometimes subtle corruption of the data by a data acquisition system or by the transmission channel. A well known first step in any engineering data analysis problem is the reduction or elimination of "noise" and/or the treatment of missing values [2, pp. 58-66]. Although it is sometimes possible to reduce the noise at the data source, it is the responsibility of the overall data system designer to minimize the corruption that may occur at any point in the information delivery path.

**DATA DETERMINATION**




### Data Communications

In nearly every case, data must be transported from the data acquisition facility to the computer used to manage the database. Often, the transmission must take place over long distances and between non-homogeneous computers. The data transmission is often asynchronous and bursty, i.e. one doesn't know when to expect the data nor how much he will receive. Large scale engineering data systems must be able to handle many different kinds of traffic in modes that may range from real-time to hand-entered keyed transactions. The resulting need is for the consideration of the many problems of computer-to-computer data communications, protocols, and, not infrequently, the potential variety of data formats. It is here that the problem of data transformation is first encountered. For example, one may record a signal via telemetry using pulse code modulation and need to transform the data into ASCII files. Perhaps one records a long time sampled waveform via real-time digitizing but needs to apply a sampling frequency reduction to fit the computer input buffer or the analysis program.

### The Engineering Database

Once captured, the data must be stored for eventual

use. As with any database problem, the database organization should be determined by the expected retrieval and analysis requirements. It is not always possible, however, to provide even a basic characterization of the ultimate use of engineering data. For example, a digital waveform would certainly be subject to simple time or frequency domain displays. However, one often needs to derive, from a set of such waveforms, a vector of random variables for statistical analysis. These considerations give rise to orthogonal database organization requirements - a file structure for the original waveforms and, perhaps, a relational database structure for the scalar variables.

In order to achieve consistent analysis of engineering objects, data in engineering applications must be manipulated in an integrated manner [5]. However, the attributes described by different tests of an object, by tests of different objects, or as derived from data transformations, may not be in any way homogeneous. Yet, if one is to analyze the data in an integrated system, the organization of the database must result in a coherent technique for retrieval. This and similar problems in engineering database organizations make it difficult to acquire commercial software (i.e. DMBS) for the archival storage of engineering data [6].

### Engineering Data Analysis

Engineering data may be displayed in simple reports, plotted in many forms, input to mathematical computations, or used to drive simulations. Very often, the end-user, typically a practicing hardware engineer or analyst, is not a frequent computer user nor especially interested in the underlying mathematical or computational methodology used for the required analysis. The analytical tools provided for the user must be well defined, robust, friendly, and convenient. In order for a computer based engineering information system to be effective, it must be easier to use than to avoid. On the other hand, the tools should not provide to an untutored user a facility with which to easily hang himself. For example, a non-statistically trained engineer can easily make incorrect decisions based upon the misuse of certain inferential statistics or by improperly ignoring the underlying assumptions governing use of the tools.

## NEED FOR DATA ENGINEERS

As we list the characteristics of a large scale engineering data system, we see that the development of such a system requires talent not usually associated with typical database development projects. Ideally, one would have a team composed of computer scientists, database administrators, electrical engineers, programers, mathematicians, statisticians, and other analysts. Should the project be sufficiently large (or the company sufficiently rich with people), that is precisely the recommended formula. However, most projects don't require the full time, life long contributions from each of these disciplines and, given the essential need for an integrated system, we see the need for a professional designer to put it all together. The solution is apparent: develop individuals with a substantial grasp of all of the required technology and call them Data Engineers. The task is made much easier if one were to start with people trained in an engineering discipline, both because the engineering point of view is important and because it is much easier to teach an engineer some of the concepts of computer science than to teach a computer scientist engineering. Of course, an individual or team of Data Engineers should be supported by dedicated professionals as required by the needs of the project.

Once the system is established, there is a need for similar professionals to keep the system going. Since engineering data systems are linked to the "real" world and, given that the real world is always changing, such systems are never really "finished". Frequently, the required changes are fundamental.

## EXAMPLE: PRODUCT TEST DATA (PTD) SYSTEM

Sandia National Laboratories has developed and oeprates a large scale engineering data system called the Product Test Data system. The PTD system is operated in support of the design, development, and quality assurance engineers associated with Sandia's role in the development and stockpile surveillance of the ordinance components of nuclear weapons. Nuclear weapons are complex systems with very high reliability requirements. They must remain safe and operate under extreme environmental conditions. The PTD system provides both a working tool for development engineers and weapon system analysts and a database intended to characterize the functional performance and safety of stockpiled weapons. As such, it is typical of the engineering data systems described above.

### Computing Environment

The system is composed of 6 Gigabytes of test data related to over 1650 different types of components. Test data are obtained during all phases of component development, production, and stockpile testing. The sources of data range from local laboratory tests to flight tests conducted at remote ranges with a substantial volume of data obtained from production at sites located throughout the country. Data types include scalar variables data, digital waveforms, attributes, and associated descriptive information. The data are stored in a single database located on a VAXcluster consisting of one VAX 11/780 and two VAX 8600's operated under VMS. End-user access to the system is via a single, interactive, menu-driven program, which is itself a system of over 100 functional routines.

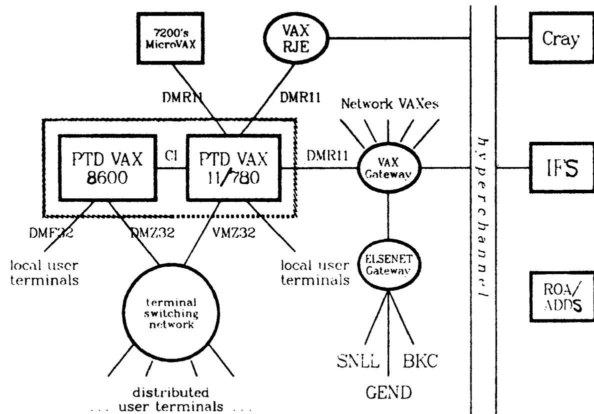
The organization of the database presents a particular problem due to the variety of record types and, especially, to the fact that no two component types share the same data descriptions. The data for each component must, therefore, be defined independently of any other. Since digital waveforms are retrieved and analyzed differently from scalar variables, each data type for any component is stored separately from the other.

### Data Determination

Data Determination, as described above, is conducted on an individual component or system basis. The several downstream uses of the data from any test program are considered during the design of the component data system. It would be undesirable, for example, to ignore the need for baseline characteristics related to stockpile surveillance while measuring component performance during a production acceptance test. Additionally, since data are

invisible, adequate external documentation of the database contents must be developed for each component or weapon system along with documentation that defines the entire data acquisition, transmission, and storage path.

Communications Diagram for the PTD VAXcluster




## Analysis

Data analysis can take on many forms and is driven both by the type of data to be analyzed and by the problem at hand. For the most part, the end-users of the system are graduate engineers who have little or no computer training or experience. It is necessary to provide a data retrieval, sub-setting, noise-reduction, and analysis system in a single, user friendly package. As noted above, the end-user must have a data utilization tool that is robust, friendly, and convenient. A complete, menu driven, interactive system, called MIRACLE, has been developed for the PTD system and is in use by engineers that are, at least somewhat, computer literate. The ultimate goal, to be achieved in the next couple of years, is to improve the user interface so that the system can be used directly by completely untutored and, perhaps, infrequent users. As an aide to the use of statistics by such engineers, the system will incorporate a Statistical Expert System. The expert system will provide the user with a strategy for data analysis and will automatically check the data and procedure for conformance to an appropriate set of assumptions. Work is also underway on the development of a waveform pattern recognition facility so that large numbers of digital waveforms can be selected on the basis of user-described shape features.

To the extent possible, commercial software has been incorporated into the design of the analysis system. To date, a relational database system for ad-hoc data manipulation, two statistical analysis packages, a preliminary statistical data analysis advisor (expert system), and a waveform analysis system have been acquired for PTD analysis.

## The PTD Data Engineer

Essentially, individual data systems are created for each engineering object represented in the PTD system. The individual data systems are then folded into the overall PTD system. The entire process, from data acquisition to analysis, is conducted under the control of a PTD Data Engineer assigned to the individual (or several) data systems. The Data Engineer is, typically, a graduate engineer, scientist, or senior technician with training or experience in the design or development of computer based data systems. The Data Engineer is responsible for the design and specification of the individual data systems and for their integration into the PTD database. A Data Engineer has a peer relationship with engineers in design, development, or quality assurance and these groups look to the Data Engineer for solutions to ad-hoc data analysis problems as well as for the construction of data systems.

The PTD Data Engineer must, then, be first an engineer as well as a part-time programmer, database administrator, data communications expert, and analyst or statistician. Clearly, Data Engineers are not born nor are they graduated. They develop their capabilities as they pursue their careers. Most find the experience to be unique, challenging, and rewarding.

## CONCLUSION

The design, development, and operation of engineering data systems requires professional contributions from those not typically associated with administrative database development projects. Often, an engineering viewpoint is necessary in order to define the data systems required for engineering decisions. These conditions result in the requirement for a professional Data Engineer who combines the discipline of the traditional hardware engineer with the expertise of certain software professions.

## REFERENCES


1. Ramamoorthy, C.V., Proceedings of IEEE International Conference on Data Engineering, IEEE Computer Society, Los Angeles, CA 1984, Forward, p. iii.

2. Proceedings of the Second IEEE International Conference on Data Engineering, IEEE Computer Soc., Los Angeles, CA, 1986.

3. Codd, E.F., "A Relational Model of Data for Large Shared Data Banks," Co-m. ACM, Vol 13, No. 6, June 1970, pp. 377-397.

4. Jones, S.E. and Ries, D.R., "A Relational Data Base Management System for Scientific Data," Proc. Sixth International CODATA Conf., 1978, Santa Flavia, Sicily, Italy. (Lawrence Livermore Laboratory Preprint UCLR-80769)

5. Udagawa, Y. and Mizoguchi, "An Advanced Data-Base System ADAM --- Towards Integrated Management of Engineering Data," in ref. [2], pp. 3-11.

6. Benkovitz, C.M. and Tichler, J.L., "Scientific Data Bases on a VAX 11/780 Running VMS," Proc. of the Digital Equipment Computer Users Soc. (DECUS), Fall 1980, DECUS U.S. Chapter, Marlboro, MA, pp. 55-63.

# DATATRIEVE SIG

# Advanced DATATRIEVE Record Definitions

**B. Z. Lederman**
ITT World Communications
New York, NY

**Abstract**

This session is intended to illustrate some of the more advanced features of DATATRIEVE record definitions. Lower case text indicates commands typed in by a user, upper case is printed by DTR or is material stored in the CDD. Please keep in mind that most examples are "stripped down", showing only the fields necessary to illustrate the principles being demonstrated: "real" applications would require additional fields, and in most cases more descriptive field names. Most of these examples use advanced features found in VAX-DTR and DTR-20, and unfortunately will not work in DTR-11 (or PRO-DTR).

Reading a file whose records differ in length and field layout is a common problem. In the following sample file, there are records whose total length is not given directly by a field in the record.

```
01 10 bytes.
02 15 byte record
01 10 bytes.
03 This is 20 bytes...
02 15 byte record
04 This is 25 bytes long...
01 10 bytes.
03 This is 20 bytes...
02 15 byte record
01 10 bytes.
04 This is 25 bytes long...
01 10 bytes.
```

You can just define a text field the length of the longest record, but you get "Record too Short..." error messages, and the short records are padded with blanks or zeroes. Also, it would be hard to look at the individual data items within each record. A first try at a better record definition could be:

```
DTR> show var-rec

RECORD VAR_REC
01 VAR_REC.
    10 TYPE PIC 99 EDIT_STRING Z9.
    10 REC_LEN COMPUTED BY TYPE
       VIA VAR_LEN_TAB.
    10 TOP.
      15 VARIABLE OCCURS 0 TO 30 TIMES
         DEPENDING ON REC_LEN.
        20 VTEXT PIC X.
    10 A REDEFINES TOP.
        20 FILLER PIC X.
        20 NBRA PIC 99 EDIT_STRING Z9.
        20 FILLER PIC X.
        20 TXTA PIC X(6).
    10 B REDEFINES TOP.
        20 FILLER PIC X.
        20 NBRB PIC 99 EDIT_STRING Z9.
        20 FILLER PIC X.
        20 TXTB PIC X(11).
    10 C REDEFINES TOP.
        20 FILLER PIC X.
        20 TXTC1 PIC X(7).
        20 FILLER PIC X.
        20 NBRC PIC 99 EDIT_STRING Z9.
        20 FILLER PIC X.
        20 TXTC2 PIC X(8).
    10 D REDEFINES TOP.
        20 FILLER PIC X.
        20 TXTD1 PIC X(7).
        20 FILLER PIC X.
        20 NBRD PIC 99 EDIT_STRING Z9.
        20 FILLER PIC X.
        20 TXTD2 PIC X(13).
;
```

This depends upon a table that converts the record type to a record length. This happens to be in a domain table, in this example, but could also be in a dictionary table.

```
DTR> show var-tab-rec

RECORD VAR_TAB_REC
01 VAR_TAB_REC.
    10 TYPE PIC 99 EDIT_STRING Z9.
    10 LENGTH PIC 99 EDIT_STRING Z9.
;
```

```
DTR> show var-len-tab

TABLE VAR_LEN_TAB FROM VAR_TAB_DOM
USING TYPE : LENGTH
END_TABLE

DTR> print var-tab-dom

TYPE LENGTH

  1     10
  2     15
  3     20
  4     25
```

If you print this domain, you get the first field by default.

```
DTR> print var

    REC
TYPE LEN VTEXT

  1  10
          1
          0

          b
          y
          t
          e
          s
          .
  2  15
          1
          5

          b
          y
          t
          e

          r
          e
          c
          o
          r
          d
  1  10
          1
          0

          b
          y
          t
          e
          s
          .
  3  20
          T
          h
          i
          s

          i
          s

          2
          0

          b
          y
          t
          e
          s
          .
          .
          .
```

etc. This is very useful in cases where you want to get each character in the record separately, such as for "parsing" data, and you get the length of the text without having to add an FN$STR_LENGTH function to DTR. However, if you want all of the data in a single field:

```
DTR> for var print a

NBRA  TXTA

 10  bytes.
 15  byte r
 10  bytes.
Illegal ASCII numeric "Th".
  0  s is 2
 15  byte r
Illegal ASCII numeric "Th".
  0  s is 2
 10  bytes.
Illegal ASCII numeric "Th".
  0  s is 2
 15  byte r
 10  bytes.
Illegal ASCII numeric "Th".
  0  s is 2
 10  bytes.
```

and the same happens for all other REDEFINEd fields, because the numeric fields don't "line up". One alternative is to use a CHOICE statement in a procedure to get the proper field to print out. (You can also use IF-THEN-ELSE statements to accomplish the same result, and that approach will also work with DTR-11, but CHOICE is more compact.)

```
DTR> show var-print

PROCEDURE VAR_PRINT
```

```
FOR VAR BEGIN
    PRINT TYPE, CHOICE OF
        TYPE = 1 THEN A
        TYPE = 2 THEN B
        TYPE = 3 THEN C
        TYPE = 4 THEN D
        ELSE " "
    END_CHOICE
END
END_PROCEDURE
```

```
DTR> :var-print

TYPE

  1   10 bytes.
  2   15 byte record
  1   10 bytes.
  3   This is 20 bytes...
  2   15 byte record
  4   This is 25 bytes long...
  1   10 bytes.
  3   This is 20 bytes...
  2   15 byte record
  1   10 bytes.
  4   This is 25 bytes long...
  1   10 bytes.
```

This suits many applications, but is sometimes inconvenient. An alternative is a record definition (actually a VIEW) that will print out the proper fields by default. (See figure 1.)

```
DOMAIN VARI OF VAR USING
01 VARIX OCCURS FOR VAR.
    10 TYPE FROM VAR.
    10 REC_LEN FROM VAR.
 02 AV OCCURS FOR VAR WITH TYPE = 1.
    10 NBRA FROM VAR.
    10 TXTA FROM VAR.
 02 BV OCCURS FOR VAR WITH TYPE = 2.
    10 NBRB FROM VAR.
    10 TXTB FROM VAR.
 02 CV OCCURS FOR VAR WITH TYPE = 3.
    10 TXTC1 FROM VAR.
    10 NBRC FROM VAR.
    10 TXTC2 FROM VAR.
 02 DV OCCURS FOR VAR WITH TYPE = 4.
    10 TXTD1 FROM VAR.
    10 NBRD FROM VAR.
    10 TXTD2 FROM VAR.
```

This has the slight drawback that, since there is nothing which identifies unique records in this example, all records of a given type are obtained for each record in the view. In cases where there was an additional field with a unique key, this would not be a problem: in this case, however, some additional work can solve the problem. (See figure 2.)

```
DTR> show print-first-vari

PROCEDURE PRINT_FIRST_VARI
DECLARE N PIC 9.
N = 1
FOR VARI BEGIN
    WHILE N = 1 BEGIN
        N = N + 1
        PRINT
    END
END
END_PROCEDURE
```

This is one way to one complete set of records. Another method is:

```
DTR> find vari

[12 records found]
```

The following is not quite what we want.

```
DTR> for current print av

NBRA  TXTA

 10  bytes.
 10  bytes.
 10  bytes.
 10  bytes.
 10  bytes.
 10  bytes.
 10  bytes.
 10  bytes.
 10  bytes.
 10  bytes.
 10  bytes.
 10  bytes.
 10  bytes.
 10  bytes.
 10  bytes.
 10  bytes.
 10  bytes.
 10  bytes.
 10  bytes.
 10  bytes.
 10  bytes.
 10  bytes.

Execution terminated by operator.
```

but you can do this:

```
DTR> select first
```

now, you can do some interesting things, like separating the different groups of similar records.

```
DTR> print av
```

```
DTR> print vari

     REC
TYPE LEN NBRA  TXTA   NBRB     TXTB       TXTC1   NBRC  TXTC2     TXTD1   NBRD      TX
  1  10   10  bytes.  15   byte record This is  20   bytes... This is  25   bytes
          10  bytes.  15   byte record This is  20   bytes... This is  25   bytes
          10  bytes.  15   byte record
          10  bytes.
          10  bytes.
  2  15   10  bytes.  15   byte record This is  20   bytes... This is  25   bytes
          10  bytes.  15   byte record This is  20   bytes... This is  25   bytes
          10  bytes.  15   byte record
          10  bytes.
          10  bytes.
  1  10   10  bytes.  15   byte record This is  20   bytes... This is  25   bytes
          10  bytes.  15   byte record This is  20   bytes... This is  25   bytes
          10  bytes.  15   byte record
          10  bytes.
          10  bytes.

Execution terminated by operator.
```

Figure 1

```
DTR> :print-first-vari

     REC
TYPE LEN NBRA  TXTA   NBRB     TXTB       TXTC1   NBRC  TXTC2     TXTD1   NBRD      TX
  1  10   10  bytes.  15   byte record This is  20   bytes... This is  25   bytes
          10  bytes.  15   byte record This is  20   bytes... This is  25   bytes
          10  bytes.  15   byte record
          10  bytes.
          10  bytes.
```

Figure 2

```
NBRA  TXTA

 10  bytes.
 10  bytes.
 10  bytes.
 10  bytes.
 10  bytes.

DTR> print bv

NBRB    TXTB

 15  byte record
 15  byte record
 15  byte record

DTR> print cv

 TXTC1  NBRC  TXTC2

This is  20  bytes...
This is  20  bytes...

 TXTD1  NBRD     TXTD2

This is  25  bytes long...
This is  25  bytes long...
```

Normally I would discourage the use of FIND and SELECT, but in this case it can be used to sort and separate all records of a given type.

Still, this is not quite what we were looking for. If you can put a CHOICE statement into a procedure, why not put it into the record definition.

```
DTR> show cvar_rec

RECORD CVAR_REC USING
01 CVAR_REC.
    10 TYPE PIC 99 EDIT_STRING Z9.
    10 REC_LEN COMPUTED BY TYPE
       VIA VAR_LEN_TAB.
    10 FILLER PIC X.
    10 TOP.
      15 VARIABLE OCCURS 0 TO 30 TIMES
         DEPENDING ON REC_LEN.
        20 FILLER PIC X.
    10 A REDEFINES TOP.
        20 ANBR PIC 99.
        20 FILLER PIC X.
        20 ATXT PIC X(6).
    10 B REDEFINES TOP.
        20 BNBR PIC 99.
        20 FILLER PIC X.
        20 BTXT PIC X(11).
    10 C REDEFINES TOP.
        20 CTXT1 PIC X(8).
        20 CNBR PIC 99.
        20 CTXT2 PIC X(9).
    10 D REDEFINES TOP.
        20 DTXT1 PIC X(8).
        20 DNBR PIC 99.
        20 DTXT2 PIC X(14).
    10 TEXT COMPUTED BY CHOICE OF
        TYPE = 1 THEN ATXT
        TYPE = 2 THEN BTXT
        TYPE = 3 THEN CTXT1 ||| CTXT2
        TYPE = 4 THEN DTXT1 ||| DTXT2
        ELSE ""
        END_CHOICE.
    10 NUMBER EDIT_STRING Z9
       COMPUTED BY CHOICE OF
        TYPE = 1 THEN ANBR
        TYPE = 2 THEN BNBR
        TYPE = 3 THEN CNBR
        TYPE = 4 THEN DNBR
        ELSE 0
        END_CHOICE.
;

DTR> print cvar

     REC
TYPE LEN          TEXT            NUMBER

  1  10  bytes.                     10
  2  15  byte record                15
  1  10  bytes.                     10
  3  20  This is  bytes...          20
  2  15  byte record                15
  4  25  This is  bytes long...     25
  1  10  bytes.                     10
  3  20  This is  bytes...          20
  2  15  byte record                15
  1  10  bytes.                     10
  4  25  This is  bytes long...     25
  1  10  bytes.                     10
```

Just to prove that NUMBER is really numeric

```
DTR> for cvar print fn$log10(number)

  1.000
  1.176
  1.000
  1.301
  1.176
  1.398
  1.000
  1.301
  1.176
  1.000
  1.398
  1.000
```

Now we finally have the data in the form we want. Something which is not visible when you look at this printout on paper is that the field TEXT always prints out the

length of the actual field: it does not pad short records with spaces or zeroes which is what would happen if you just defined one field of 25 bytes (you also don't get the "Record too Short" error messages).

There are a number of applications where data validation in the record definition is desired. In this example, the employee number contains a sort of "check sum", where the last two digits are the sum of the first two. This sort of thing is sometimes done to verify that the data does not contains errors (I'd rather depend on the operating system facilities, but some people would prefer this). This particular check sum is a bit crude, and done only to demonstrate the methods which may be used. If you were going to do this a lot, it would be worthwhile to define a new FN$— function to do the computation, especially if the check method was more complicated such as some sort of "rule of 11", but not everyone wants to add functions to DTR. The interesting part is that you can define a VALID IF clause to work on parts of the same field it validates, and that the fields used can be defined after the VALID IF clause.

```
DTR>show empno-rec

RECORD EMPNO_REC
01 EMPNO_REC.
    10 EMPLOYEE_NUMBER PIC 99999
        VALID IF CK = (N1 + N2 + N3).
    10 NBRS REDEFINES EMPLOYEE_NUMBER.
        20 N1 PIC 9.
        20 N2 PIC 9.
        20 N3 PIC 9.
        20 CK PIC 99.


DTR> print empno

EMPLOYEE
 NUMBER

 12306
 65617
 98724
 11002
 00101
 32308
```

Something that users don't always realize is that a COMPUTED BY field can be anywhere in the record definition, and does not have to be computed from fields that come "ahead" of it in the definition. DTR will read and parse the entire record definition to resolve all field names before doing anything with the record: thus, a field can, in some cases, even be computed by itself.

With this definition, you can prevent invalid numbers from being stored.

```
DTR> store empno

Enter EMPLOYEE_NUMBER: 32301

Validation error for field EMPLOYEE_NUMBER.
Re-enter EMPLOYEE_NUMBER: 32308
```

You can also find out if all the numbers currently in the domain are still valid (something which a normal VALID IF won't do):

```
DTR> for empno print ck, (n1 + n2 + n3)

CK

06   6
17  17
24  24
02   2
01   1
08   8
08   8
```

Now look at what happens if an invalid number is present in the domain.

```
DTR> print empno

EMPLOYEE
 NUMBER

 12306
 65617
 98724
 11002
 00101
 32301 [this number is invalid]

DTR> print empno with ck ne (n1 + n2 + n3)

EMPLOYEE
 NUMBER

 32301
```

We can use DTR to go in and fix any checksums. (I would advise looking at the data first to be certain it really is valid, unless you want to do something like this to add checksums to data that was stored previously without checksums.)

```
DTR> ready empno modify
DTR> for empno with ck ne (n1 + n2 + n3)
     begin
CON>   modify empno using ck=n1+n2+n3
CON> end

DTR> print empno

EMPLOYEE
 NUMBER
```

```
12306
65617
98724
11002
00101
32308
```

While thinking up stuff for this presentation, I came up with the following example which, quite frankly, I didn't think would work.

```
DTR> show sci_rec

RECORD SCI_REC
01 SCI_REC.
    10 SCI_NOT USAGE REAL EDIT_STRING 99.99.
    10 N2 COMPUTED BY *."N2".
;
```

Depending upon how you access the domain, you can be prompted for N2 once per record (might be used to make the system pause during loops), once per domain, or not at all.

```
DTR> for sci print sci_not

 SCI_NOT

00.01
00.88
01.20
09.80
23.40

DTR> print sci
Enter N2: 30

 SCI_NOT  N2

00.01 30
00.88 30
01.20 30
09.80 30
23.40 30

DTR> for sci print sci_rec

 SCI
 NOT      N2

Enter N2: 30

00.01 30
Enter N2: 20

00.88 20
Enter N2: 10

01.20 10
Enter N2: 1

09.80 1
Enter N2: 0

23.40 0
```

Having done this, I'm not at all sure what I would use it for, but I'm sure someone will think of something someday. Knowledge never goes to waste.

Some COMPUTED BY fields are more useful than others. For example, if several departments share a data base and you want to make sure that each department enters the correct sequence of numbers (this example assumes a valid range of numbers for each department, just to make it more difficult):

```
DTR> show po_rec

RECORD PO_REC
01 PO_REC.
  10 DEPT PIC XXX.
  10 PO_NUMBER PIC 99999 VALID
     IF 1 = CHOICE OF
        (DEPT = "AAA" AND PO_CHECK
         BETWEEN 01 AND 20) THEN 1;
        (DEPT = "BBB" AND PO_CHECK
         BETWEEN 21 AND 40) THEN 1;
        (DEPT = "CCC" AND PO_CHECK
         BETWEEN 41 AND 60) THEN 1;
      ELSE 0
      END_CHOICE.
   10 PO_CHECK REDEFINES PO_NUMBER.
      20 DEPT_NO PIC 99.
;

DTR> print po

       PO
DEPT NUMBER

AAA  01001
BBB  21001

DTR> store po
Enter DEPT: AAA
Enter PO_NUMBER: 01002
DTR> store po
Enter DEPT: BBB
Enter PO_NUMBER: 01003

Validation error for field PO_NUMBER.
Re-enter PO_NUMBER: 21002
```

This isn't bad, but it could be better. Why store the department number and verify it, when you could change the record definition and force it to always be correct? (This time I'm assuming one prefix per department.)

```
DTR> show po-rec
```

```
RECORD PO_REC
01 PO_REC.
    10 DEPT PIC XXX VALID
       IF DEPT = "AAA", "BBB", "CCC".
    10 HIDEIT.
        20 FILLER PIC 999.
    10 REAL_STUFF REDEFINES HIDEIT.
        20 DEPT_SEQ PIC 999.
    10 PO_NUMBER PIC 99999 COMPUTED
       BY CHOICE OF
        DEPT = "AAA" THEN DEPT_SEQ + 01000
        DEPT = "BBB" THEN DEPT_SEQ + 02000
        DEPT = "CCC" THEN DEPT_SEQ + 03000
        ELSE "00000"
       END_CHOICE.
;
```

We can also force the sequence number to be correct.

```
DTR> show store-po

PROCEDURE STORE_PO
DECLARE MAXSEQ PIC 999.
DECLARE TMPDEP PIC XXX.
TMPDEP = FN$UPCASE(*."Department")
MAXSEQ = MAX(DEPT_SEQ) OF PO
         WITH DEPT = TMPDEP
STORE PO USING BEGIN
    DEPT = TMPDEP
    DEPT_SEQ = MAXSEQ + 1
END
END_PROCEDURE

DTR> :store-po
Enter Department: bbb
DTR> print po

       PO
DEPT NUMBER

BBB  02001
BBB  02002
AAA  01001
BBB  02003

DTR> :store-po
Enter Department: bbb
DTR> print po

       PO
DEPT NUMBER

BBB  02001
BBB  02002
AAA  01001
BBB  02003
BBB  02004
```

If you try to store a department which has no records yet, you get an error message, but you also get the correct result anyway

```
DTR> :store-po
Enter Department: ccc

Can't take MAX, MIN, or AVERAGE
   of zero objects.
DTR> print po

       PO
DEPT NUMBER

BBB  02001
BBB  02002
AAA  01001
BBB  02003
BBB  02004
CCC  03001
```

Something which I have run into, and which others have asked for at past Q and A sessions, is how to get non-VMS date strings into the VMS/DTR date type, especially when you are not able to restructure the data. The following very non-standard date and time is the type of data I've actually encountered.

```
$ type date.seq

86:01:02 1003A
85:03:14  120P
86:09:29 1100P
86:11:11  332A

DTR> show date-rec

RECORD DATE_REC
01 DATE_REC.
    10 INPUT.
        20 I_YEAR PIC 99.
        20 FILLER PIC X.
        20 I_MONTH PIC XX.
        20 FILLER PIC X.
        20 I_DAY PIC 99.
        20 FILLER PIC X.
        20 T_HOUR PIC XX.
        20 I_HOUR PIC 99
           COMPUTED BY T_HOUR.
        20 I_MINUIT PIC 99.
        20 I_AP PIC X.
    10 O_DATE COMPUTED BY
        FN$DATE(I_DAY | "-" | I_MONTH VIA
        MONTH_TABLE | "-19" | I_YEAR).
;
```

The date part is easy: you just need a table to turn the numeric month into an upper case alphanumeric month.

```
DTR> show month_table

TABLE MONTH_TABLE
01 : "JAN",
```

```
02 : "FEB",
03 : "MAR",
04 : "APR",
05 : "MAY",
06 : "JUN",
07 : "JUL",
08 : "AUG",
09 : "SEP",
10 : "OCT",
11 : "NOV",
12 : "DEC"
END_TABLE
```

```
DTR> print datei

 I   I    I    T    I     I    I      O
YR MNTH DAY HOUR HOUR MINIT AP     DATE

86 01   02   10   10    03   A   2-Jan-1986
85 03   14    1   01    20   P  14-Mar-1985
86 09   29   11   11    00   P  29-Sep-1986
86 11   11    3   03    32   A  11-Nov-1986
```

Not too bad: but when you have to add the time things get a little bit more complicated. I've shown only the hour and minuit here, but you can add seconds and fractions of a second as well. Note that I'm also using FILLER to hide the input fields, so by default only the wanted fields print.

```
DTR> show date-rec

RECORD DATE_REC
01 DATE_REC.
   10 HIDEIT.
      20 FILLER PIC X(14).
   10 INPUT REDEFINES HIDEIT.
      20 I_YEAR PIC 99.
      20 FILLER PIC X.
      20 I_MONTH PIC XX.
      20 FILLER PIC X.
      20 I_DAY PIC 99.
      20 FILLER PIC X.
      20 T_HOUR PIC XX.
      20 I_HOUR PIC 99
         COMPUTED BY T_HOUR.
      20 I_MINUIT PIC 99.
      20 I_AP PIC X.
      20 A_HOUR COMPUTED BY CHOICE OF
          (I_AP = "A" AND T_HOUR = 12)
             THEN 00
          (I_AP = "P" AND T_HOUR < 12)
             THEN T_HOUR + 12
          ELSE T_HOUR
          END_CHOICE.
      20 B_TIME COMPUTED BY
             ((A_HOUR * 60) + I_MINUIT) *
             600000000.
   10 O_DATE COMPUTED BY
      FN$DATE(I_DAY | "-" | I_MONTH VIA
      MONTH_TABLE | "-19" |
      I_YEAR ||| FN$TIME(B_TIME)).
;
```

The hard part is converting the AM/PM time to a 24 hour time, then getting it to print in the proper format. There are a number of ways it might be done depending upon the exact input format: in this case I convert the hour and minute to "clunks", then use FN$TIME to put it back to characters long enough to use FN$DATE to put the date and time back to clunks. This might seem a bit "clunky", but it's actually the easiest way to get it to work every time. The alternative is to make all of the fields "print" in the FN$DATE function the way the day and year do. (It is sometimes also possible to do this sort of thing in DTR-11: though there are no FN$— functions, DTR-11 will handle dates with embedded times in clunks.)

```
DTR> print datei

  O_DATE

 2-Jan-1986
14-Mar-1985
29-Sep-1986
11-Nov-1986
 7-Aug-1986
 4-Jul-1976
```

```
DTR> for datei print i_hour,
i_minuit, i_ap, fn$time(o_date)

 I      I     I
HOUR MINUIT AP    FN$TIME

 10    03    A  10:03:00.00
 01    20    P  13:20:00.00
 11    00    P  23:00:00.00
 03    32    A  03:32:00.00
 12    01    A  00:01:00.00
 12    58    P  12:58:00.00
```

The net result is that O_DATE now contains the complete date and time in VMS format, and all of the normal DTR boolean comparisons will work.

# Managing All-In-1 with Datatrieve

**B. Z. Lederman**
ITT World Communications
New York, NY

There are basically two types of uses for Datatrieve in managing All-In-1: the first one I will present is manipulating the All-In-1 environment. The following is a record definition for PROFILE.DAT, the file All-In-1 uses to the user profile information.

NOTE: The Datatrieve structures in this paper, with additional comments and documentation, and other structures relevant to All-In-1, are available from the DECUS library in the Datatrieve Library collection (also submitted to the VAX and RSX SIG tapes), and the older CORPHONE submission.

```
REDEFINE RECORD AI1_PROFILE_REC USING
01 AI1_PROFILE_REC.

    10 USER PIC X(30).
    10 USER_INFO.
        20 VMSNAM PIC X(12).
        20 FULNAM PIC X(32).
        20 TITLE PIC X(30).
        20 DEPART PIC X(24).
        20 STATUS PIC X(68)
           EDIT_STRING T(24).
        20 PASWRD PIC X(31).
        20 PHONE PIC X(20).

    10 RESERVED PIC X(15).
    10 PRIV.
        20 DCL PIC X.
        20 SUP PIC X.
        20 ERR PIC X.
        20 CMD PIC X.
        20 SRC PIC X.
        20 CPHD PIC X.
        20 LOG PIC X.
        20 MULTI_NODE PIC X.
        20 RSVD_FOR_TCS PIC X.

    10 ADDRESS.
        20 ADDR1 PIC X(30).
        20 ADDR2 PIC X(30).
        20 ADDR3 PIC X(30).
        20 ADDR4 PIC X(30).
        20 ZIPCOD PIC X(15).

    10 NOTIFY.
        20 NOTICE PIC X.
        20 BATCH_NOT PIC X.
        20 PRINT_NOT PIC X.
        20 MAIL_READ_REC PIC X.
        20 TICKLER PIC X.
        20 ACTITEM PIC X.

    10 DIRECTORY PIC X(68)
       EDIT_STRING T(32).
    10 FORMLIB PIC X(68)
       EDIT_STRING T(24).
    10 INIT_FORM PIC X(30).
    10 EDITOR PIC X(10).
    10 PRINTER PIC X(15).
    10 NODE PIC X(13).
    10 PRINT_PORT PIC X.
    10 TERM_MODE PIC X.
    10 MAIL.
        20 MAIL_FORWARD PIC X(66)
           EDIT_STRING T(24).
        20 MAIL_REPLY PIC X(31).
        20 MAIL_MENU PIC X(10).
        20 MAIDES PIC X(10).

    10 CALENDAR.
        20 CALTIMEING PIC X(5).
        20 SETUSR PIC X.
        20 YESDAYS PIC X.
        20 STARTD PIC X.
        20 ENDD PIC X.
        20 STARTH PIC X(7).
        20 ENDH PIC X(7).
        20 MEALS PIC X(7).
        20 MEALE PIC X(7).
        20 CALDAY PIC X.

    10 UFLAG.
        20 UFLAG1 PIC X.
        20 UFLAG2 PIC X.
        20 UFLAG3 PIC X.
        20 UFLAG4 PIC X.
        20 UFLAG5 PIC X.
        20 UFLAG6 PIC X.
        20 UFLAG7 PIC X.
        20 UFLAG8 PIC X.
        20 UFLAG9 PIC X.
        20 UFLAG10 PIC X.
        20 CLASS PIC X(10).

    10 LANGUAGE PIC X(20).
    10 END PIC X(138) EDIT_STRING T(24).
```

If you compare this definition with the screens All-In-1 gives you for user profile addition, modification, etc., you will find the fields match (more or less). Since All-In-1 gives you these fields, why would you use Datatrieve? The answer is that All-In-1 management is oriented to processing one user at a time. If you want to find out, for example, which users have DCL access enabled, you have to go through several menus and screens to get an index of users, write down their names, then examine them all one at a time to find the ones with DCL. With Datatrieve, you can ready the PROFILE domain and say 'PRINT PROFILE WITH DCL = "Y"' to find all such users. Similarly, operations on large numbers of users such as turning DCL or Logging on or off for everyone, or finding the users whose accounts point to certain disks and/or re-assigning them to other disks, are easier in Datatrieve than in All-In-1 as presently supplied. You can also use Datatrieve to produce nice formatted reports of all users or groups of users, and you can select which information fields are printed in that report.

Because All-In-1 manipulates several files for user profiles, it is NOT a good idea to add or delete user profiles using Datatrieve, though it could be done in emergencies.

A similar function is to examine the document database using Datatrieve (this also works for WPS-Plus/VMS).

```
REDEFINE RECORD DOCDB_REC USING
01 DOCDB_REC.
    10 FOLDER PIC X(30).
    10 N1 PIC 9(6).
    10 N2 PIC 9(6).
    10 SHORT PIC X(6).
    10 FILE PIC X(64).
    10 P PIC X.
    10 TITLE PIC X(72).
    10 PERSON PIC X(30).
    10 TYPE PIC X(15).
    10 A PIC X(15).
    10 B PIC X(15).
    10 C PIC X(50).
    10 D1 PIC X(8).
    10 D2 PIC X(8).
    10 OUT_N PIC X(6).
    10 F3 PIC X(30).
    10 MAIL_STATUS PIC X(8).
    10 F2 PIC X(12).
    10 NUMBER PIC 9(6) EDIT_STRING Z(6).
    10 READ PIC X.
    10 SENDABLE PIC X.
    10 N3 PIC 9(16).
    10 F4 PIC X.
    10 N4 PIC 9(16).
    10 F5 PIC X(16).
    10 CONTENT PIC X(16).
    10 F6 PIC X(45).
;
```

This definition is still undergoing development, so some of the fields are not as fully defined as they might be. However, there is one very important field which is included here which is not usually obtainable within All-In-1, and that is the VMS file which contains the document (the field FILE). It happens (a little too often) that the DOCDB.DAT file becomes corrupted, and then it is necessary to try to coordinate the VMS files with the documents: this record definition allows you to obtain a listing of all documents All-In-1 (or WPS) knows about and compare it with a directory listing to find missing documents or files. I also find that it is faster to use Datatrieve to obtain an index listing of all documents than it is to use All-In-1.

```
REDEFINE PROCEDURE DOCDB_REPORT
!
! Fast report of all documents
! with their VMS file name.
!
! B. Z. Lederman
!
REPORT DOCDB ON *."file specification"
SET COLUMNS_PAGE = 132
SET LINES_PAGE = 42
PRINT NUMBER, FOLDER,
  TITLE USING T(48), FILE USING T(24)
END_REPORT
END-PROCEDURE
```

Once again, with Datatrieve it is possible to easily change the report for the format you want. I have even put a Datatrieve procedure like this one into a form and included it as an All-In-1 application so that users can simply type one command and have a complete index of their documents sent to their default printer.

As in the case of PROFILE.DAT, I do not recommend adding or deleting documents with Datatrieve, except in emergencies (such as recovering from a corrupted DOCDB.DAT).

One final such application, still under development, is manipulation of the Time Management (Calendar) files. With the tools currently supplied with All-In-1, it is difficult to remove past appointments. Worse still: when a user leaves and you use All-In-1 to remove that users' profile and/or account, All-In-1 does NOT search through the calendar to remove that user's appointments. Using Datatrieve to search the calendar files, I have been able to purge out old appointments, and appointments for persons who are no longer users, much more easily than from within All-In-1. This is a case where using Datatrieve to delete records is justified.

There is another type of application using Datatrieve to manage All-In-1, and that is application usage. All-In-1 has the ability to generate a logging file, but there are no utilities or facilities supplied to do anything with the information generated. DATATRIEVE is a good tool for reading and analyzing this information.

```
REDEFINE RECORD AI1_LOG_REC
```

```
!
! This will read the logging
! file produced by ALL-IN-1 when
! logging is turned on.
!
! B. Z. Lederman
!
01 AI1_LOG_REC.
   05 ID.
    10 FACIL_ID USAGE LONG.
    10 MSG_ID USAGE LONG.
    10 PROC_ID USAGE LONG.
   05 TIMES.
    10 SYS_DATE USAGE DATE.
    10 SYS_TIME PIC X(12)
       COMPUTED BY FN$TIME(SYS_DATE).
    10 ELAP_TIME USAGE LONG.
   05 MESSAGE.
    10 INPUT PIC X(10).
    10 FUNCTION PIC X(16).
    10 TEXT PIC X(200) EDIT_STRING T(32).
    10 TX REDEFINES TEXT.
        20 FORM_NAME PIC X(32).
;
```

To make this information more useful, Datatrieve can be used to process the information to extract just form and do script usage, and match it up to the library where the form or script is stored. See Program 1 for the procedure AI1_NORMALIZE, which processes All-In-1 logging files so that the form names are extarcted and normalized.

There are a number of interesting bits of information here, but first it might be asked why use Datatrieve to process the information when a procedure as long as the one given above is needed? The answer is, if you compare it with the code needed for a "normal" 3rd generation language, you would have to write just as much processing code, plus a lot more for the file open and record handling work. In processing one serial file to another as we are here, Datatrieve is going to go just about as fast as any other program doing the same amount of work.

There are a number of things one can obtain from this data. One may determine how many users access All-In-1, when they use it, and the sequence of commands entered. I concentrated on what forms and do scripts were being used, with the intention of optimizing All-In-1 performance by putting the most used forms in MEMRES and the most used scripts in the TXL, and moving the lesser used forms and scripts elsewhere. With Datatrieve, it's quite easy to get summary information:

```
REDEFINE PROCEDURE AI1_NORM_RPT
!
! Report summarized use of forms and scripts
!
! B. Z. Lederman
!
READY AI1NORM
REPORT AI1NORM WITH LIBRARY NE " " SORTED BY
    FUNCTION, NAME ON *."TT or file name"
AT BOTTOM OF NAME PRINT FUNCTION, SPACE 1,
    LIBRARY, SPACE 1, COUNT, SPACE 1, NAME
AT BOTTOM OF FUNCTION PRINT NEW_PAGE
END_REPORT
END-PROCEDURE
```

```
FUNCTION    LIBRARY   COUNT   NAME

FORM        MEMRES        3 AUTO
FORM        OAFORM        1 DISPREMINDER
FORM        OAFORM        4 ENC
FORM        OAFORM        1 EMHEAD
FORM        MEMRES        4 MAIN
FORM        MEMRES        1 OA$EDIT
FORM        MEMRES        2 OA$LIST
FORM        OAFORM        1 WP
FORM        OAFORM        2 WPINDX
```

This is just a short example, but I actually went through data from a much larger sample and rearranged MEMRES and OAFORM to match. If one reads the All-In-1 manager's manual, it would appear that putting the most used forms in MEMRES, where they are compiled and linked into the All-In-1 executable task image itself, would improve performance: and, I've been told, the All-In-1 and WPS developers themselves thought so as well. It was not until I presented this material at the symposium that I was given additional information from an independent testing group within DEC that putting more forms in MEMRES not only does not improve run-time performance, it slows down task initialization. The more forms there are in MEMRES, the larger the task image and the longer it takes to start up All-In-1. If you install MEMRES and OAFORM in memory (the normal way All-In-1 is run), then the first time a form is used it is stored in memory, and from then on it doesn't matter much which library it was in.

When I first went through this exercise, I did rearrange the two libraries, and saw improvements: upon re-examining the libraries involved, I found that, after checking what forms my users actually needed, I ended up with fewer forms in MEMRES than the DEC distributed libraries, which probably accounted for the improvement. Also, I was working on a system which was drastically short of memory, so there was a potential for improvement in not installing any libraries in memory (something which may also be done if you want to run All-In-1 and WPS-Plus/VMS on the same system), and under these circumstances access to a compiled form in MEMRES might be an improvement. Additional testing will have to be done to obtain definitive answers.

This does not mean the information is not useful: far from it. Do scripts also account for a significant portion of the work done by All-In-1 (though it happens that none show in the very short examples in this paper), and putting the most used scripts into the TXL does improve performance: moving unused scripts out conserves memory.

In addition, this logging information is the best (perhaps only) method of finding out what your users are actually doing in All-In-1, where the most effort is being expended, and where efforts to improve performance or ease of use will yield the greatest results, and even how you might tune your system or adjust quotas to match the work being done.

As noted, all of the procedures (and additional information) is available through the DECUS library, but for convenience the procedure which creates the domain table containing the forms and do scripts, and the DCL command file which goes with it, are appended to this paper.

PROGRAM 1

```
REDEFINE PROCEDURE AI1_NORMALIZE
;
! Process AI1 logging file so that the form names are extracted
! and normalized.  This allows looking them up in a table to
! find out which library they are in, and to allow summation
! for statistics on use.
!
! B. Z. Lederman
!
DEFINE FILE FOR AI1NORM
READY AI1NORM WRITE
READY AI1LOG
!
! need a few working variables
!
DECLARE A_FORM PIC X(24).
DECLARE B_FORM PIC X(24).
DECLARE E1 PIC 99 EDIT_STRING Z9.
DECLARE E2 PIC 99 EDIT_STRING Z9.
!
! Go through the logging file and pick out uses of forms and
! scrips
!
FOR AI1LOG WITH FUNCTION = "FORM", "DO" BEGIN
!
! Now comes the fun part.  We want to extract only the form (or
! script) name and normalize it.
!
    E1 = 0                          ! initialize end of string
    E2 = 0                          !   position counters
    A_FORM = FN$UPCASE (FORM_NAME)    ! force upper case
    E1 = FN$STR_LOC (A_FORM, " ")     ! look for end of form name
    E2 = FN$STR_LOC (A_FORM, "/")     ! may have command attached
    IF E1 > 0 E1 = E1 - 1             ! want last character
    IF E2 > 0 E2 = E2 - 1             ! not search character
!
! take out the form name if it is not null: the form either ends
! with a null or space or with a slash if there was a command
! attached.
!
    IF ( (E2 > 0) AND ( (E2 < E1) OR (E1 = 0) ) ) THEN
        B_FORM = FN$STR_EXTRACT (A_FORM, 1, E2) ELSE
        B_FORM = FN$STR_EXTRACT (A_FORM, 1, E1)
!
! If we can find this form (script) in the domain table we
! created which lists the library each form is in, use that
! libraries' name, otherwise use a blank space.
!
    IF (B_FORM IN FORM_TABLE) THEN
        A_FORM = B_FORM VIA FORM_TABLE ELSE A_FORM = " "
```

```
!
! we now have nicely normalized data: store it.
!
    STORE AI1NORM USING BEGIN
        FACIL_ID  = FACIL_ID
        MSG_ID    = MSG_ID
        PROC_ID   = PROC_ID
        SYS_DATE  = SYS_DATE
        ELAP_TIME = ELAP_TIME
        FUNCTION  = FUNCTION
        NAME      = B_FORM
        LIBRARY   = A_FORM
    END
END
FINISH
!
RELEASE A_FORM
RELEASE B_FORM
RELEASE E1
RELEASE E2
END-PROCEDURE
```

```
REDEFINE PROCEDURE NEW_FORM
!
! Read the forms in a number of edited FMS library listings and
! scripts in directory listings and put them into a domain so
! we can look them up.
!
! B. Z. Lederman
!
! You should only have to use this procedure to rebuild your
! data base when you change the arrangement of forms and
! scripts used in AI1.
!
DEFINE FILE FOR AI1FORM KEY = FORM(DUP), KEY = LIBRARY(DUP),
    ALLOCATION = 55
READY AI1FORM WRITE
DELETE T_FORM;
!
! Get forms in OAFORM
!
DEFINE DOMAIN T_FORM USING F_T_REC ON OAFORM.SEQ;
READY T_FORM
PRINT "Storing OAFORM"
FOR T_FORM STORE AI1FORM USING BEGIN
    FORM = FORM
    LIBRARY = "OAFORM"
END
FINISH T_FORM
DELETE T_FORM;
!
! Get forms in MEMRES
!
DEFINE DOMAIN T_FORM USING F_T_REC ON MEMRES.SEQ;
READY T_FORM
PRINT "Storing MEMRES"
FOR T_FORM STORE AI1FORM USING BEGIN
    FORM = FORM
    LIBRARY = "MEMRES"
END
FINISH T_FORM
DELETE T_FORM;
!
! Get forms in MANAGER
!
DEFINE DOMAIN T_FORM USING F_T_REC ON MANAGER.SEQ;
READY T_FORM
PRINT "Storing MANAGER"
FOR T_FORM STORE AI1FORM USING BEGIN
    FORM = FORM
    LIBRARY = "MANAGER"
END
FINISH T_FORM
!
DELETE T_FORM;
!
! Get scripts in OA$DO
!
DEFINE DOMAIN T_FORM USING F_T_REC ON OADO.SEQ;
READY T_FORM
PRINT "Storing OA$DO"
FOR T_FORM STORE AI1FORM USING BEGIN
```

```
    FORM = FORM
    LIBRARY = "OA$DO"
END
FINISH T_FORM
!
DELETE T_FORM;
!
! Get scripts in OA$LIB
!
DEFINE DOMAIN T_FORM USING F_T_REC ON OALIB.SEQ;
READY T_FORM
PRINT "Storing OA$LIB"
FOR T_FORM STORE AI1FORM USING BEGIN
    FORM = FORM
    LIBRARY = "OA$LIB"
END
FINISH T_FORM
FINISH AI1FORM
END-PROCEDURE
```

```
$! AI1FORMDATA.COM
$!
$! Create data files with AI1 forms listed
$! for processing with DATATRIEVE to create a domain table.
$!
$! B. Z. Lederman
$!
$! This requires the TRIM utility supplied (supposedly) with AI1.
$! You should be in whatever account you normally use for AI1
$! system management when you run this (or the same account you
$! will use when in DATATRIEVE to analyze this data).
$!
$ SET COMMAND OA$LIB:TRIM
$!
$! The idea is quite simple.  Get a listing (either of a forms
$! library or a script directory), use TRIM to strip out
$! everything except the raw data, sort it (partially to make
$! it look nice and partially because it should load faster and
$! more cleanly into an indexed file), and purge old copies.
$! A DATATRIEVE procedure reads the sequential files created
$! here to load the domain (table).
$!
$ FMS/DIR/OUT=OAFORM.SEQ OA$LIB:OAFORM
$ TRIM/FIRST=11 OAFORM.SEQ
$ SORT OAFORM.SEQ OAFORM.SEQ
$ PURGE OAFORM.SEQ
$!
$ FMS/DIR/OUT=MEMRES.SEQ OA$LIB:MEMRES
$ TRIM/FIRST=11 MEMRES.SEQ
$ SORT MEMRES.SEQ MEMRES.SEQ
$ PURGE MEMRES.SEQ
$!
$ FMS/DIR/OUT=MANAGER.SEQ OA$LIB:MANAGER
$ TRIM/FIRST=11 MANAGER.SEQ
$ SORT MANAGER.SEQ MANAGER.SEQ
$ PURGE MANAGER.SEQ
$!
$ DIR/COL=1/NOSIZE/NODATE/NOTRAIL/OUT=OADO.SEQ OA$DO:*.SCP
$ TRIM/FIRST=3/FINAL="." OADO.SEQ
$ PURGE OADO.SEQ
$!
$ DIR/COL=1/NOSIZE/NODATE/NOTRAIL/OUT=OALIB.SEQ OA$LIB:*.SCP
$ TRIM/FIRST=3/FINAL="." OALIB.SEQ
$ PURGE OALIB.SEQ
$!
$ EXIT
```

For those who wish to experiment with the time management data, I append preliminary record definitions for ATTENDEE.DAT and MEETING.DAT

```
DELETE ATTENDEE_REC;
REDEFINE RECORD ATTENDEE_REC
01 ATTENDEE_REC.
    10 NAME1 PIC X(30) EDIT_STRING T(15).
    10 MATCH1.
        20 YEAR PIC X(4).
        20 MONTH PIC X(2).
        20 DAY PIC X(2).
        20 TIME PIC X(4).
        20 LENGTH PIC X(4).
    10 NAME2 PIC X(30) EDIT_STRING T(15).
    10 FLAG PIC X.
    10 YES_NO PIC X(3).
    10 MESSAGE PIC X(56) EDIT_STRING T(24).
    10 APPOINT OCCURS 6 TIMES.
       15 MATCH.
        20 YEAR PIC X(4).
        20 MONTH PIC X(2).
        20 DAY PIC X(2).
        20 TIME PIC X(4).
        20 LENGTH PIC X(4).
    10 END PIC X(19).
;

DELETE MEETING_REC;
REDEFINE RECORD MEETING_REC
01 MEETING_REC.
    10 NAME1 PIC X(30) EDIT_STRING T(15).
    10 MATCH1.
        20 YEAR PIC X(4).
        20 MONTH PIC X(2).
        20 DAY PIC X(2).
        20 TIME PIC X(4).
        20 LENGTH PIC X(4).
    10 MATCH2.
        20 YEAR PIC X(4).
        20 MONTH PIC X(2).
        20 DAY PIC X(2).
        20 TIME PIC X(4).
        20 LENGTH PIC X(3).
    10 MESSAGE PIC X(60) EDIT_STRING T(15).
    10 LOCATION PIC X(32) EDIT_STRING T(16).
    10 AGENT PIC XX.
    10 FILLER PIC X(31).
    10 NAME2 PIC X(30) EDIT_STRING T(15).
    10 FILLER PIC X(60).
;
```

# USES OF ACCENT R

Winston Tellis
Fairfield University
Fairfield, Connecticut

ABSTRACT

A look at the applications of ACCENT R in a university environment, with a brief history of the selection process. A brief introduction to the data base system.

It would be useful to present some information about the background about Fairfield University as well as the way in which the choice was made to obtain ACCENT R as our data base language.

Fairfield University has about 3000 full-time and approximately 3500 part-time undergraduate students as well as about 1000 graduate school students. There is also a high school on the campus. In 1968 we leased an IBM 1130 based educational system called the 1500. In 1980 we purchased a Decsystem 2060 and finally in January 1986, a VAX 8600.

Just about the time that the announcement was made to discontinue the DEC 10 and 20 systems, we were in the process of investigating data base systems. We therefore thought it prudent to add to our list of criteria an item that was to become important - products that run on both systems DEC 20 and VAX with little or no change.

The criteria we used to evaluate the products was as follows:

a) User friendliness
b) File description flexibility
c) Programmer productivity
d) DEC 20/VAX similarity
e) Performance

We made a judgement that the University's next system would be a VAX of some kind - the 8600 was not on the boards then. We thus planned that we would start converting each existing software application to this new data base so that when we switched to the next system, there would be minimal conversion problems. On this basis alone, we chose ACCENT R because the syntax is identical on the two systems. However all our other criteria was met to our satisfaction.

a) User Friendliness. The language and communication process in the data base system should be understandable without the need to go to manuals. A HELP facility is highly desirable. The syntax of the access language should be as close to natural language as possible so as to encourage non-technical users to feel comfortable.

b) Programmer Productivity. We feel that the FORTRAN/COBOL approach to applications programming is time consuming considering the usual backlog we all seem to have. We want this language to dramatically reduce the time needed to code an application, be it small or large, with programmer tools where necessary.

c) File Description Flexibility. We would like to avoid having to create duplicate copies of our data files, just so that we may use the data base system. This language should be able to describe any file organization and support it. This turns out to be a very important factor if you have, as we do, data files being accessed by multiple languages and being modified by them all.

d) DEC 20/VAX syntax similarity. Our goal is to move all our software to the VAX with little or no conversion. Consequently, the software developed on the DEC 20 should run on the VAX essentially unaltered.

e) Performance. We estimate that a significant number of users will be accessing some files concurrently and they could be large files. We do not expect a performance level worse than the current FORTRAN/COBOL applications that are being replaced. We actually want a performance superior to the current.

Our choice of ACCENT R was particularly good in the areas of critical importance to us. The syntax is identical - in fact we have migrated to a VAX 8600 and the only software we did not have to change at all was our ACCENT R code. The file description flexibility on the DEC 20 was outstanding. On the VAX the RMS file support is rapidly becoming available. This feature has saved us untold amounts of disk space. The typical time taken to complete an application is better than FORTRAN/COBOL by a factor of 10 to 1 or even 20 to 1 depending on the application and ACCENT R has routinely run about 3 times faster than FORTRAN. We are pleased with our choice and the product keeps improving.

There are several areas in which we have used ACCENT R either completely or to complement the current development. All new applications are done in ACCENT R.

## ACADEMIC

We have a system design course where ACCENT R is taught for second half of the course. The students design and develop applications after a rather brief introduction. It is interesting to observe how quickly the students become familiar with the language and how quickly they become proficient enough to develop their own applications.

Some of the faculty use it to retrieve information and do some statistical work.

ADMINISTRATION

ACCOUNTS RECEIVABLE

There is an existing sequence of FORTRAN and COBOL programs. However all but one of the programs has been rewritten - the exception being the Ledger cash totalling, master file updates etc., are all handled in ACCENT R. The programmers feel that they can do the routine maintenance chores much more productively and in a fraction of the time if they use ACCENT R, and at the same time it is much easier to develop a quality user friendly program or module with very little effort.

There are 6 schools involved, including the high school all supported by this system. ACCENT R was particularly useful in dealing with the slight differences that exist between the schools.

STUDENT RECORDS

This is the most ambitious project to date. It was developed entirely in ACCENT R and in fact we would not even consider doing it otherwise, taking 2 people working full-time about 2 months. The system includes the usual - biographical maintenance, grading - including printing optical scanning grade sheets, transcripts, course registration, class lists and housing information (actually a subsystem). All of the above are available for all authorized users at the screen or on hardcopy. This information is shared sometimes simultaneously, by the offices of admissions, financial aid, accounting, the registrar, the president, the deans, security and devlopment. All except the last office access the data through ACCENT R and the entire project was designed and developed in a very short period of time that actually overlapped with the conversion from the Dec 20 to the VAX.

The system is initiated by the entry of data set with the teacher information. Once this is in place and verified by the Registrar's Office, the 'add-drop' process can proceed. The students have used opscan forms that already have their ID and other pertinent information printed and all they do is blacken the dots with the course code number(s) of their choice. When the scanner has transmitted this data to the VAX, it is processed through a series of FORTRAN prograns until the file containing the scheduled courses are ready for inclusion in the 'courses' data set in ACCENT R. Incidentally the reason for the FORTRAN programs is that they already existed before we obtained ACCENT R and we have not had the time or the courage to change what works fine!

This data becomes part of an already extensive scheme to make available to a variety of users any information that they are entitled to and need. There is a data set containing all the courses the student has taken at this institution along with the grades and the semester. Other containing courses transferred to Fairfield, with the course(s) equivalent at Fairfield, is entered and maintained by the Dean charged with this responsibility, but available to affected officers at any school at the University. The student's current schedule is available on demand as is the dorm residence date maintained by the Housing Office.

The transcripts are displayed and/or printed on a laser printer either at the main Computer Center or at the Registrar's Office. This step has taken a great clerical burden from the staff and they merely enter the changes and carry on without the need to recompute the student's average or rank.

FINANCIAL AID

Completely designed in ACCENT R this system was written in about 2 weeks of full-time equivalent. It includes complicated screens for data input and update. Aid calculations and awards are made, the letters written by the system through a laser printer. Letters are generated on missing documents and various management and government reports are prepared.

The system begins with tapes arriving from The College Scholarship Service and being stored through a COBOL program into a sequential data set. An ACCENT R program then takes over and adds these names to the Financial Aid data set for processing. Screens allow the employees to call up any applicant and check the record and make any change needed. This is where the routines come in that have the user automatically send letters to the applicants who have incomplete applications. When all the decisions are made by Admissions Office, this office goes into high gear running the simulation that makes the awards for the Financial Aid Office to inspect. When they have make all the necessary adjustments, award letters are sent out with all the categories awarded clearly laid out. The same procedure is followed for the returning upper classmen.

PURCHASING

This is a system that ran in RMS indexed on the DEC 20 system, mainly because of the numerous views of the data needed by the office. However, there never was the flexibility and ease of response to change that a major application demands. Hence knowing that the VAX was ordered we went about redesigning the application and did it entirely in ACCENT R in about 2 months from start to finish with just one part time senior programmer.

Using a Decmate for input, the user enters all the purchase order information and saves it on the diskette while getting immediate printout on the Decmate printer. This diskette is periodically transmitted to the VAX, where a FORTRAN program fixes the input, clearing it of the extraneous prompts associated with the Decmate and readies it for ACCENT R.

At this stage, the staff merely tracks the purchase order through it's life, making full or partial payments etc. until it is closed and ready for the Accounting Office to pay the bill. A program makes a itemized list by vendor for the Accounting Office by due date for payment. Meanwhile any item not closed at the end of the month is automatically included in a report that becomes a batch in the University's Expense Reporting System as 'Open Purchase Orders'. The latter system is entirely written in FORTRAN, and ACCENT R has absolutely no problem interfacing with this system.

ADMISSIONS

This is a very high pressure system with a great need for both accuracy and quick response to a great volume of requests in a short period of time. The system was originally exclusively in FORTRAN and later in combination with data entry screens written in COBOL. As applications are received, the clerical staff enters it as fast as possible. Another screen allows change to any field in the system.

Once the data is entered, it is available to the numerous statistical and reporting programs that are constantly in demand during the period of decision making.

We are now at the point that only the very complex statistical programs are still in FORTRAN, because we have not had the courage to delve into them. Everything else is in ACCENT R some of it even generated by the user. Where the Computer Center was repeatedly in receipt of urgent requests for service, we now seldom intervene.

All the reporting programs were developed in a few hours with or by the user and a simple menu now presents the user a choice of reports and they have become virtually self-sufficient. It took a short period of time to describe the file ACCENT R after which the screen programs were developed with existing tools. We were able to go a step further using ACCENT R by offering them the service of using the laser printer to print the various letters on their own letterheads but automatically generated in ACCENT R, based on the major or residence status or other varying criterion.

The net result has been a vastly improved product as well as throughput for a very important system to the University.

A relatively simple but useful system was also developed to analyze a survey mailed out to all accepted applicants regarding their choice and general impressions and attitudes. The data entry and survey analysis is done with existing ACCENT R statistical statements and a minimum of programming. Once again the speed with which this information becomes available is of some importance to the institution.

PERSONNEL

This is a particularly sensitive system for obvious reasons. It was developed on the basic model of the Admissions system with specific modifications to reflect the nature of this office. As in that system, there are screen entry programs and numerous reporting programs of varying complexity.

When we obtained ACCENT R, we promptly described the existing data set in ACCENT R and began to make use of the tools to generate new reports with very little effort. Reports that used to take 20 minutes to program and produce, now were complete in 2 minutes. The long range plan is to convert the entire system to ACCENT R.

SECURITY

This is the latest addition to the family of ACCENT R applications. In essence it had to wait till several other aspects of our development

fell into place, because the system requires access to all the data sets that have students or employees stored in them. With ACCENT R we are able to search any kind of file match the ID and take the name and address from there.

It is a system where the user enters the ID and the make of the vehicle and some other information at the time the person registers the vehicle. Only when reports are required do we have to match the ID against some other file to find the name and address. We also produce tickets as a by-product of this system, the ticket being processed through to accounting if a student caused the infraction. It automatically links to the accounting system, preparing an input file ready for use.

ALUMNI

This system is no longer in use although it was one of the first and largest in terms of volume and number of records. The system was a hybrid of FORTRAN, COBOL and ACCENT R. When we found out the vastly superior performance of ACCENT R over the other two, we used the Host Language Interface in those programs to improve the throughput.

The system consisted of data entry and update using screen programs. There was a link to the undergraduate information system and there was complete transaction processing. At critical times, they would have numerous people doing entry simultaneously. We found it much easier to deal with the special purpose reporting that seems to be peculiar to Development Offices. When the VAX was obtained, this system was replaced by third party software.

ALUMNI-REUNION

An urgent request to develop a system to handle the reservations for residence and the various events associated with a class reunion. The system was developed in about 6 hours work, that included 2 screens to handle the event reservation and another for the overall registration. Several reporting programs were developed to list the various event rosters as well as print badges and the usual incidentals.

We would not have ever considered such a project in this time frame were it not for ACCENT R.

FACILITIES INVENTORY

This is a simple system consisting of one screen developed in 3 minutes to allow the Management Information Office to do a space inventory of the entire institution. In addition there are a few reporting programs developed using the QREP utility. The total development time was about 1 week.

RESERVATIONS

The Campus Center Office makes reservations for all non-academic events, and this system allows central control over all the alloted space, checking for conflicts and printing reports for the weekly and/or daily event schedule. While the overall system is extremely simple in concept and implementation, the conflict checking became quite involved. The system was developed in about two weeks.

FUTURE

Future plans include completing all the pieces of the accounting system so that a General Ledger System may be developed in ACCENT R and now that the student information system has been integrated into one in ACCENT R it is possible to address that problem.

OVERVIEW OF ACCENT R

At this point I would like to indulge in a very brief introduction to the syntax of ACCENT R so that you might be able to make a judgement about it's features, strengths and weaknesses. The system forces a consistent standard of structured code that will last beyond the person presently doing the coding. The program has sections that have built in logic relieving the programmer of most of the mundane tasks of titling and totalling etc. and with no need to define input/output statements.

All the items are stored in a Data Base Library (DBL) which is created once the first time and then repeatedly 'used'. In this DBL is stored the Data Definition (SD) the Data Files (DS) Data Indexes (DI) and even Programs (PM) and a few other items. The data is truly separated from the description thereof. In other words, the schema definition (SD) is a separate entity from the Data Set (DS) and in fact numerous Data Sets may be described by a single SD. Data Indexes (DI) store

the various views of the data set that are desired and are automatically maintained by the system. With just this much information about the environment, one may begin to describe the application to be developed. Below is a short run through a typical development sequence.

```
ACCENT

  Invoke the system

*create dbl biznis

  Create a Data Base Library (once only)

*define sd stock

  Create schema to describe data

--10 item.no,c,5,title='Item #'

  this 5 position field is a-n

--20 item.desc,c,15,title='Desc'

  note the title, it is used in reports

--30 unit.price,n,8,2,title='Price'

  this item is numeric with 2 decimals

--40 on.hand,i,3,title='Stock'

  This is an integer field

--50 on.order,i,3,title='On Order'

--save

  Save this schema definition

*create ds stock sd stock

  Create a data set in the format of sd

    At this point an entry is made in the
    DBL tying this DS to SD Stock.

*dir

  A directory of the elements in the DBL

TYPE                  NAME

DBL                   BIZNIS

DS                    STOCK SD STOCK

SD                    STOCK

*use ds stock
  Invoke the ds for use in the following commands

*enter new with prompts check

  Start entering data into the data set
  Stock, with field names being prompted

ITEM.NO: A-101

  User's entry is after the colon

ITEM.DESC:  Movie Camera

UNIT.PRICE: 445

ON. HAND:  20

ON.ORDER:  5

Okay ? y

  And an optional verification.

ITEM.NO:  A-102

ITEM.DESC:  Movie Screen

UNIT.PRICE:  17.50

ON. HAND:  10

ON. ORDER:  5

Okay ? y

ITEM. NO. ***

  Terminate the input session

*type @records

  How many records have I entered

*extract

  Let me look at all the data edited

A-101 Movie Camera  445.00 20 5

  Leaves spaces between fields
```

A-102 Movie Screen  17.50 10 5

*extract show item.desc,' ',on.hand

Movie Camera  20  Just show me these two fields

Movie Screen  10

There are various clauses that allow conditional extracts and saving that output on a print file or on another data set.

*extract if item.no has 'A'

Examples of such clauses

*extract if on.hand + on.order > 50

There are other verbs that are used interactively as the ones above and are extremely easy to understand even for non-technical users:

| | |
|---|---|
| SORT | Sort on some field name |
| DELETE | Delete a record by value or record number |
| ALTER | Change the content of a field(s) |
| CONVERT | Convert from one sd to another |
| CHANGE | Change field content |
| REMOVE | Delete an item from the DBL |
| RENAME | Rename an element of the DBL |
| SELECT | A Join command to select from 2 data sets |
| UPDATE | Allows a Master to updated from a transaction data set. |

Examples

*ALTER if cust.name = 'atlas' set '203-555-1212' to phone & show cust.name,phone
(change all records that meet the condition, and change the phone number, and show the new one and the customer name)

*change field on.order with prompts  show item.no, ' ',on.hand
(each record will be displayed and the user will be prompted for the value)

*use ds oldset

convert to newset
(having created a new ds newset, which perhaps has fields that ds oldset does not, you wish to transfer all the old corresponding data to the new ds. The move takes place if the field names are identical)

*sort on cust.name
(sort the data set on this field)

*use ds orders

*select with stock match on item.no if: t unit. price > 4.00 & show item.no, unit.price
(ds orders is the Master, Stock the transaction set; when item.no matches and the condition is met - :t meaning the transaction set must meet the condition - the item is displayed)

Once the data has been described, there are situations when the capabilities presented above are not sufficient to cover a particular need. Thus there are Process Modules - the programming capability of ACCENT R. It is a very structured model that is built into the logic of the various sections of the program. The basic model requires

| | |
|---|---|
| INITIALIZE | Tell ACCENT R what to do before processing records |
| PROCESS | Which operations to perform on all records |
| FINALIZE | What to do after processing records |

An actual Process module may have up to twenty different sections that organize the structure of the task.

In the example below, we have 5 sections to produce a report that has a heading, details of each salesperson's activity, district sub-totals and a grand total, creating a print file.

| | |
|---|---|
| CONTROL | Declares the data set(s) to be used |
| DECLARE | Working storage for temporary values |
| INITIAL | Sets up initial values |
| HEADINGS | Sets up headings to appear on each page |
| DETAIL | Specifies the processing details for each record |

```
TOTALS         Specifies the break point for
               sub-totalling

FINAL          Produces final totals
```

The actual process module looks like this:

```
00010 CONTROL SECTION

00020 RELATE DS SALPER AS MASTER
       !declare salper as the master file

00030 RELATE TERMINAL AS REPORT 1
       !the output will show at the terminal

00040 DECLARE SECTION

00050 CNT,I,2
       !variable cnt is 2 position integer

00060 YTD.SALES,N.10,2
       !this is a numeric field with 2 dec

00070 INITIAL SECTION

00080 3 TO @TOP.MARGIN,@BOTTOM.MARGIN
       !set 3 to these system fields

00090 60 TO @LINES
       !this system field stores lines/page

00100 HEADINGS SECTION

00110 SKIP 3

00120 CENTER "DISTRICT SALES REPORT" AT 38
       !center this title

00130 CENTER @FDATE AT 38
       !center the date also

00140 SKIP 2

00150 PRINT TAB 19,"EMPLOYEE",TAB 60, "YEAR-TO-
DATE SALES"
       !col title

00160 PRINT 75"="
       !print 75 = signs

00170 DETAIL SECTION

00180 0 TO YTD.SALES
       !set variable to 0

00190 START FOR CNT = 1 TO 12

00200 YTD.SALES + MO.SALES(CNT) TO YTD.SALES

00210 REPEAT
       ! a loop that adds up the mo.sales field
        that has 12

00220
       ! monthly sales figures, storing the result
        in YTD.SALES

00230 PRINT TAB 13,LNAME,1B,FNAME,TAB 63,YTD.SALES
       ! print the fields

00240 TOTALS SECTION

00250 ON DISTRICT

00260 PRINT TAB 64,9"-"

00270 PRINT "DISTRICT:, 1B,DISTRICT,TAB 51,
"SUBTOTAL",TAB 63,& SUM YTD.SALES
       ! print the fields and the sum of vtd.sales

00280 PRINT 75"-"

00290 SKIP 5

00300 FINAL SECTION

00310 PRINT TAB 48,"GRAND TOTAL",TAB 63,SUM TYD.
SALES
```

Notice that there are no input-output statements to read from the data set, as this is built into the logic of the DETAIL section. The Titling is automatically taken care of by the HEADINGS section. On a control break, the TOTALS section takes care of the mundane chores as does the FINAL section at the end. This logic is very crisp and clear, very easy to use and to teach. The framework of a report is often used to write QREP which prepares all the sections with the headers and fields all the user does is answer the questions.

There is a facility to control a sequence of jobs that have to be processed perhaps with some decisions along the way, even passing data from this module to a process module. This facility called a Control Module, is very flexible and useful in keeping control of a job.

Sample Control Module:

```
00010 TYPE @CR

00020 TYPE "TYPE 'MENU' FOR A LIST OF AVAILABLE
```

```
OPTIONS"'@CR

00030 START
        !start a loop

00040 TYPE "OPERATION?",NOCR

00050 ACCEPT @STRING
        !allow input of string info

00060 IF:5 @STRING = "MENU"

00070 TYPE "TYPE ONE OF THE FOLLOWING: ",@CR

00080 TYPE "ORDER TO ENTER ORDER,UPDATE STOCK"

00090 TYPE "MO COMMTO COMPUTE A GIVEN MONTH COM-
MISSION"

00100 TYPE "YTD COMM YTD COMMISSION OF A SALES-
PERSON

00110 TYPE "HIRE TO ENTER A NEW SALESMAN"

00120 TYPE "TERM TO TERMINATE A PERSON"

00130 TYPE "STOP TO EXIT THIS PROGRAM"

00140 ORIF:5 @STRING = "ORDER"

00150 USE DS ORDERS

00160 ENTER VIA ORDERS.NEW

00170 ORIF:5 @STRING = "MO COMM"

00180 USE DS OTHER

00190 TYPE "ENTER SALESPERSON NUMBER ",NOCR

00200 ACCEPT @INTEGER

00210 EXTRACT IF SALNUM = @INTEGER VIA MO.
COMMISSION

00220 ORIF:5 @STRING = "YTD COMM"

00230 USE DS OTHER

00240 TYPE "ENTER SALESPERSON NUMBER ",NOCR

00250 ACCEPT @INTEGER

00260 EXTRACT IF SALNUM = @INTEGER VIA YTD.COM-
MISSION

00270 ORIF:5 @STRING = "HIRE"

00280 USE DS OTHER

00290 ENTER WITH PROMPTS

00300 ORIF:5 @STRING = "TERM"

00310 USE DS OTHER

00310 USE DS OTHER

00320 "ENTER NUMBER OF TERMINATED EMPLOYEE ",NOCR

00330 ACCEPT @INTEGER

00340 DELETE IF SALNUM = @INTEGER APPEND DELETED
TO TERM.EMP

00350 ORIF:5 @STRING = "STOP"

00360 LEAVE

00370 ELSE:5

00380 TYPE "NOT A VALID OPERATION: TRY AGAIN "

00390 REPEAT
```

The above is rather typical menu where there is a combination of functions that are either process modules or ACCENT R's data manipulation language or both. One could also link to DCL and perform any operation and stay within ACCENT R. A brief explanation of the above CM follows:

Any field preceeded by @ indicates that it is a system field. The START and REPEAT form the loop to be executed that is terminated by the condition in the ORIF that corresponds to LEAVE being a true condition. There is a hierarchy to the logic, hence IF:5 means that is level 5 and all level 5 statements are equal and superior to level 6 statements, and so on. All the statements below the IF or ORIF would be executed if the condition is true.

In statement 160 the 'vie' represents a call to a PM written to accomplish the task at hand.

CONCLUSION

This paper is meant to inform the reader of

the capabilities of a data base system that we have found to be very easy to use, easy to learn and teach and very cost effective as far as programmer time is concerned. It has alot of features that can only be discovered through actual use and could not adequately be described herein.

All the applications listed are in production or have been and continue to run successfully on a daily basis, most of them with multiple simultaneous access. We have not come across an application that we have felt should not be done in ACCENT R, although, if there were a primarily computational job, I am sure we would not rush headlong into doing it in ACCENT R although it has a respectable array of statistical and mathematical routines and functions.

We have found the company to be extremely responsive to problem reports and even with enhancements that our site has needed urgently.

# EDUSIG

# MAKACT: An Account Maintenance Program for Large VAX/VMS Environments

**Pat Feldner** and **George Stefanek**
Illinois Institute of Technology
Chicago, Illinois

### Abstract

MAKACT is an account creation program designed for large VAX/VMS environments which allows the creation of categories of accounts through a shell interface. It includes the use of user modifiable templates which specify information regarding each category of account. This makes for fast and easy creation of accounts in an organized manner.

## Introduction

MAKACT is an acronym for *make account* which is a program for the creation and maintenance of accounts especially suited for large VAX/VMS environments. The motivation behind writing this user account creation program was to be able to create and delete thousands of accounts yearly, manage large numbers of dissimilar accounts, build in a standard organization for account maintenance, and organize the accounts by category. The application of this program is currently in a university environment containing three VAXs connected by DECnet.

Typically, 6000 student accounts are created and removed yearly. Also, there are many research accounts which are maintained on a yearly basis. Eventhough this program is suited for a university environment it can be used in any setting.

## Features

Some of the features of MAKACT are:

- *Interaction through a shell*: which allows flexibility and error checking. When MAKACT is invoked it puts you immediately into a shell which prompts for a command. From here you can enter "help" or issue one of the allowed commands. After selecting a command, the shell prompts you for all required information to create, delete or maintain accounts.
- *Modularization*: accounts are organized under categories where each category designates the *type* of department to which the account belongs. For instance, there may be *types* of categories such as dept, funded, external, and class which would clearly designate a class of account. These classifications will be used in creating a directory path for the account (e.g. drc0:[dept.math.smith]). As each category of account is created, it uses specific information stored in a template for that category. Separate programs and files are maintained for each command in the shell as well as to create the various categories of accounts. Therefore, modifications are easy since specific modules are modified rather than the entire main program.
- *Templates*: user modifiable templates are provided for each *type* of account. Various templates specify account information which may be tailored individually for each *type* of account. This tailored account information will be used by MAKACT to create accounts.
- *Batch jobs*: account creation and removal will occur when MAKACT submits a batch job with the appropriately created command files. DCL command files are created by MAKACT to carry out UAF, diskquota, and directory creation and deletion.

## Templates

MAKACT makes use of several templates which hold specific account information by department. The templates consist of a *budget template, default template, customize template*, and *directory template.*

The budget template (figure 1) specifies information for categories of accounts which includes fields for

- *type* of accounts within a category (e.g. math, chem, cs),
- disk to be used for a particular account (e.g. drc0:, dra0:),
- permanent and overquota limits,
- charge rate for billing (e.g. 1, 2, 3 where 1 may correspond to a CPU rate of \$2.00/cpu minute, 2 corresponds to \$2.50/cpu minute etc.)
- budget for the entire year.

Budget Template for Dept category

| Type | Sub-typ | Dsk | Grp | Perm | Over | Rate | Bud |
|---|---|---|---|---|---|---|---|
| +dept | cs | drc2 | 100 | 5000 | 1000 | 1 | 10000 |
| +dept | chem | drc2 | 220 | 5000 | 1000 | 1 | 7000 |
| +dept | math | drc2 | 305 | 5000 | 1000 | 1 | 4000 |
| +dept | econ | drc2 | 250 | 5000 | 1000 | 2 | 9000 |

This template can be modified through MAKACT and its field can be left blank if not applicable.

The *default template* specifies default UAF parameters [VMS84] to be setup for every account created. These usually include the following qualifiers: /device, /flags, /sflags, /defpriv, /priv and any other qualifiers which set UAF parameters which may be desired for each account. This template called *default.sav* is modifiable manually through EDT. To customize any of the default UAF settings for a particular *type* of account, or to add parameters which would not normally be added to an account, the *customize template* must be modified.

The *customize template* specifies customized UAF parameters for a specific category of account. The file *customize.sav* includes a label identifying the *type* of account (e.g. #dept, #funded, etc.) followed by modifications to be made to UAF parameters. These modifications would typically include granting ACLs and modifying existing UAF parameters. A typical entry would look like

```
#EXTERNAL
mod/prclm=1
mod/maxjobs=2
mod/access
grant/id mail_user
grant/id tape_user
```

This entry may be followed by another similar entry for a different *type* of account. This file is also modifiable manually using EDT.

Finally, the *directory template* specifies the last directory name that was assigned to a new user. This is an optional template which can be excluded from the creation of accounts if another methodology of naming user's directories is used. One way of assigning directory names is to give the directory a letter followed by a number (usually 3 digits) such as "E109.dir". The number "109" can be associated with a user's box number as well as directory specification. MAKACT reads the *directory template* to find the last directory assigned and increments the box number by two and then changes letters alphabetically upwards once all box numbers within a range (e.g. 0 - 999) are assigned. If you have fewer or more box numbers at your site then MAKACT can be modified to reflect the appropriate range.

### Directory Structure

There is an implied directory structure which is used by MAKACT. At top level, disk:[000000], directories correspond to the *type* of account, such as dept, class, external, funded, etc. Under each directory *type* are the specific category *sub-types* such as chem, ee, biol, nmr, etc. Under each *sub-type* of directory are the individual user directories such as x317, x319, etc. When creating an account withing MAKACT, the shell prompts with *type of account*, *sub-type of account* and checks the input against valid existing *types* and *sub-types* in the *budget template*. If they do not exist the user is warned that the types or sub-types do not exist and should be added. If all input is valid, then MAKACT proceeds to create the account and checks whether a directory *type* exists at top level, and if it doesn't then it creates it along with the *sub-type* and individual directories under the top level of the tree.

MAKACT creates three command files during account creation. *Makact.com* adds entries to the UAF, *makdir.com* creates the directories for new accounts following the previously described tree structure, and *makquo.com* adds quota entries to diskquota. Also, a *roster.dat* file is created which lists all new accounts created. These three command files are submitted as a batch job as soon as they are created. The existance of a file *makall.temp* is checked before submitting the batch job to verify that an account creation job is not already running.

During account removal three similar files are created. *Remact.com* removes entries from the UAF, *remdir.com* removes directories (only the individual user's directory, not the entire tree path), and *remquo.com* removes quota entries from disquota.

### Using MAKACT

MAKACT is invoked by typing

```
$ MAKACT
```

which prompts with

```
Command>
Type HELP to see what
     commands are available.

Command> help
Topic?<ret>

Add Delete Disuer Exit General_info
Help Labels Make Modify Old Remove Show
```

Most of these topics have examples available. A brief description of the functions of these commands is as follows,

- *add*: adds entries into the *budget template*,
- *delete*: deletes entries from the *budget template*,
- *disuser*: disusers a specified account(s),
- *exit*: exit the MAKACT program,
- *general_info*: description of MAKACT,
- *help*: description of how to use *help*,
- *labels*: make labels for many accounts,
- *make*: create a new account,
- *modify*: modify an account *type* in the *budget template*,
- *old login*: remove account(s) that haven't been used in a specified period of time,
- *remove*: remove an account(s),
- *show*: show all or specific *types* of accounts in the *budget template*.

### Creating an Account

To create an account invoke MAKACT and issue the *make* command,

```
$ MAKACT

Command> make

Type of account> dept
Sub-Type of account> chem
Money per account> 200.00

name> smith
name> jones
name>^Z

Type of acount>^Z

Command>^Z

$
```

In the example above accounts are created for smith and jones within the dept *type* of category belonging to the chem *sub-type*. Each account is given $200.00. This field can be left blank if user accounting is not enabled. To exit a specific mode of MAKACT, type a ^ Z which brings you one level higher. For instance, the ^ Z after the "name>" prompt brings you up one level higher at which point you can enter additional users belonging to a new *type* of category. Eventhough you are one level higher, MAKACT is still within the *make* mode of account creation. To exit the *make* mode of account creation, another ^ Z must be typed to move up on more level. Eventually, after enough ^ Z's are hit the program exits.

Another feature available in account creation is the ability to use a /list qualifier after the *make* command which allows one to use a file containing a list of new users.

```
$ MAKACT

Command> make/list
File = cs350.lis
Command> ^Z
$
```

To remove an account the *remove* command is issued within MAKACT.

```
$ MAKACT

Command> remove

Username> cs_taylor
Username> cs440smith
Username> ^Z

Command>^Z

$
```

The /list qualifier can also be used in the removal of accounts. When there are large numbers of accounts to be removed at any one time as in a university environment, a list of usernames can be put into a single file which can thereafter be used as input for removal. A listing from the SYSUAF.DAT file can be searched for a specific class of accounts and the output directed to an output file. That file can be used as input to the MAKACT program.

### Conclusions

This program has been in use in a university environment for one year and has proved to be extremely useful and especially time saving in the creation and maintenance of accounts. It also has forced an organization on the systems which has simplified system maintenance and problem solving.

Future modifications to MAKACT include the use of screen management to visually display accounts as they

are being created, the cleanup of IO to the terminal from the shell, and the improvement of our *treedelete* program to search the index for all files belonging to a user.

## References


[1] VAX/VMS, *Guide to VAX/VMS System Management and Daily Operations*, Maynard, Mass.: Digital Equipment Corporation, Sept. 1984.

# Printing Across the Network

**Ray Peterson, Mark Draughn, George Stefanek**
Illinois Institute of Technology
Chicago, Illinois

**Abstract**

A program is presented which allows VAXs running VMS to share printers across DECnet [1]. The DEC output print-symbiont routine is replaced with a user defined routine which redirects printing to a remote node connected by DECnet. A server on the remote node drives the printer. This allows the user to issue a regular print command and have the output printed on a printer attached to a remote node.

## Introduction

The motivation behind writing this program is

- to add the ability to share printers across DECnet,
- to be able to direct printing to an available printer when the current default printer goes down, and
- give users the option of choosing a printer of their choice which may be located on another node in a DECneted environment (e.g. laser, matrix, graphics printers, etc.).

The advantages of incorporating this ability into a networked environment are obvious, especially since each system doesn't have to have a printer attached to it. Based on one's printing needs a printer may be chosen which can service a few VAXs which are networked together via DECnet. If there are many printers scattered across a network of VAXs, then if one fails, printing can continue uninterrupted as soon as it is redirected to another working printer. The user can also choose to override the default printer by issuing the *"print/que=remote_node"* command as long as another print symbiont process is available which forms a link to the desired remote node. All qualifiers are honored when printing across the network.

Finally, users can choose specific printers scattered across various nodes on the network based on their printing needs. For instance, someone may want to print a document on a high quality printer located on node A, another person may wish to dump a data file on a regular impact printer located on node B, and still another person may want to print a graphics file on a graphics matrix printer located on node C.

## Implementation Features

The implementation features include

- replacing the *VMS Output Print Symbiont Routine (PSM)* with a user defined routine [2],
- this routine redirects printing to a network link connected by DECnet to a remote node, and
- a server resides on the remote node which drives the printer.

The *print symbiont* consists of several routines the first of which are a number of *VMS PSM input routines* which direct files to be printed into an input buffer. Next, a *formatting routine* formats the file(s) in the input buffer and places them in an output buffer. Finally, an *output routine* sends the file(s) in the output buffer to a printing device. The user-defined output routine *netwrite* replaces the *VMS print symbiont output routine* and leaves the *input* and *format routines* untouched. The replacement of the *PSM output routine* with *netwrite* is accomplished by calling PSM$REPLACE which replaces the specified *PSM* routine with a new routine. Once this is done the PSM is modified and the sending node software is in place. All that has to be done on the sending node is to start the *PSM* specifying the command *"start/queue/on=node||lpa0: lpa0:"*. The *"/on"* qualifier specifies the node and the device which printing is to take place on. The double bars "||" are used instead of two colons "::" since VMS thinks it looks like a cluster. The *"start"* command starts the print symbiont on the current node, opens up a channel to the remote node specified by the *"/on"* qualifier, and starts up a server process on the remote node [2].

The server routine on the receiving end allocates and deallocates the printer. The printer is deallocated after a file is printed so that other nodes can allocate the printer and print to it. Once the printer is allocated the server process receives a file to be printed from the sending node and prints it, then deallocates the printer.

## PSM and Server Protocol

This section will discuss the protocol between the *print symbiont* on the current node and the *server process* on the remote node.

- When the queue is started the PSM$K_OPEN function code is passed and *netwrite* opens a channel through DECnet to a designated remote node and starts the remote *server process* (REC).
- A start of task flag (SOTask) is sent to *netwrite* by the *print symbiont.* The user-defined *PSM output routine* sends a "B" to the *server process* on the receiving-end to inform it of a SOTask.
- Once the receiving-end gets a SOTask, the *server process* allocates the printer and attempts to assign a channel to it. If the printer is printing a file from another node which is allocated by another *server process*, then the *VMS lock manager* is used to arbitrate access to the printer that is being used.
- Once the printer assignment is done the server process returns a *VMS status code* to the *PSM output routine.* The *output routine* returns the status to the *PSM* and waits for information from the *PSM.*
- When the PSM$K_WRITE function code is passed to the output routine *netwrite*, it sends a "W" followed by the data to be printed on the receiving-end.
- The receiving-end's server process uses QIOs to print to the output device (printer) and when it is done it returns a status to the sending end.
- After receiving a status from the server the *print symbiont* sends another SOTask to *netwrite* to indicate a new file to be printed. There is no end of job (EOJ) sent by the *PSM.* Once *netwrit e* receives a new SOTask it sends an end of task (EOTask) to the receiving end.
- After the receiving-end *server process* gets the EOTask status, it deallocates the printer and returns a VMS status back to the sending-end. At this point another node can use the printer. If there is no other node or another *server process* which has allocated the printer, then the *server process* allocates the printer again and prints the next file as described in the above steps.
- If there is no new job, the receiving-end times out after 5 seconds. This parameter can be shortened or extended by the administrator of the system.

These steps describe all the steps in communicating between the sending node's *PSM output routine* and the receiving-end's *server routine.*

## Limitations and Conclusions

The routines for printing across DECnet have several limitations. First, error recovery is poor. When the reveiving-end's node goes down, no status is checked which results in the job getting lost and the *queue manager* going down. The *queue manager* and all queues have to be manually restarted. Error recovery should be upgraded to avoid this problem by requeuing the job.

Another limitation is that the server accepts jobs from nodes first-come- first-serve. There is no priority set to any node. Also, the server doesn't differentiate between entire print jobs and single files. After a single file is printed, the printer is deallocated and another node prints a file and then another node prints a file and so on in a circular manner. Therefore, files from a single print job will be interspersed between files printed from other nodes. The "*/flag=all*" qualifier must be set on the print queue so that each file is identified by a flag page since the files from a single print job will not be appended together.

Finally, the server is not multi-threaded. That is, there is one server process on a node for each node printing to that node. It may be desired to have one process to handle all nodes, but this is a difficult task to implement.

In conclusion, printing across the network is a useful program that lets you share printers across the network in a way beyond that provided by VMS.

## References


[1] DECUS Tape Library, "Source and Documentation for *printing across the network*", San Francisco DECUS Tape, 1986.

[2] VAX/VMS Vol. 5B, System Routines, PSM1-45, 1984.

# STUDENT/FACULTY COMMUNICATIONS BY COMPUTER

Claude M. Watson
Lansing Community College
Lansing, Michigan

ABSTRACT

This paper will focus on the benefits to students and faculty from internal communications via a VAX-11-780. The basic communication features will be identified with simple enhancements described. New features and capabilities that this communication provides the educational environment will be discussed.

## Introduction

The importance of communications by computer has long been recognized. Technology is providing better networking solutions to enable better communications. Educational institutions have reported on the design and implementation of communication projects to deliver more computing to more students. Common features of these projects include some form of workstation and campus-wide networking of the workstations through a large computer system. This paper will report on some specific benefits in course management and communication that already exist for a computer system which supports terminals, and can be extended to workstation networks as they become available.

## Computing Environment at Lansing Community College

Lansing Community College is located in downtown Lansing a few blocks from the State Capitol of Michigan and a few miles from Michigan State University. The College has an enrollment of approximately 20,000 students; equated to about 12,000 full time students. The College is divided into five semi-autonomous Divisions.

Computing support consists of an IBM 3083 for administrative functions and students of Computer-Aided-Design and data processing, an IBM 4381 which is also for CAD students, a DEC VAX 11/780 with 96 devices for Computer Based Education, two MicroVAX II's for CBE in the Arts and Sciences Division, a PDP 11/44 for administrative word processing, a PDP 11/34 for the Library, a PDP 11/23 for environmental control of the institution, and over 300 microcomputers mostly of the IBM family with a few from the Apple family

## CBE Computing Environment on the VAX 11/780

LCC's Computer Based Education environment on the VAX 11/780 has an account structure that parallels the administrative structure of the College. (Figures 1, 2 & 3) Directory and file protection defaults are set primarily to provide sharing and access within a group of accounts. (Figure 4) The user must override the defaults to obtain additional privacy or to share beyond the group. Accounts for classes that need computer support on a regular basis are also organized into groups with the instructor assigned a "leader account" within the group.

In addition to the standard DCL commands, a number of local commands have been created by the System Manager. (Figure 5) These commands make it easy for the instructors to access information in libraries and staff and student accounts.

The VAX environment was created by the System Manager for the support of computer based education. Two examples that demonstrate the promise and potential of this new kind of environment are the management of classes and the coordination of multiple-section courses.

## Management of Classes

There are many ways in which the system aids the instructor's management of a class. Most of them relate to communications. Every group of accounts has a library for common access. The instructor creates the contents of the library to provide instructional support for the course. Such support often includes samples of instructional materials, additional assignments, examples, references, corrections of common mistakes, etc. Through his own account the instructor can leave daily or weekly messages which appear on the screen at the time the student logs in to his or her account. The instructor has access to all of the student accounts in the class or group. This enables the instructor to be appraised of each student's progress in classroom assignments quickly and accurately. Requiring students to name each assignment with a common name allows the instructor to view or print any or all of the students' versions of a single assignment. When a student asks for assistance with an assignment, the instructor can copy the student's version into his own account where it can be modified or debugged for instructional purposes without changing the student's original version.

## Coordinating Multi-Section Courses

In addition to having a leader account in a class, each instructor has a personal account in a departmental group. Instructors can access each other's departmental accounts and the group's common library in the same manner in which student accounts can be accessed. Instructors do not have write or delete privileges in each other's accounts.

Multi-section accounts may have a course manager who may identify the files (handouts) which have

been created for the course in the past, distribute them to the other instructors by paper copy or computer mail, or place them in a group library. In the latter form, all instructors of a course may contribute to the development of common assignments, handouts or tests, working together as a team without the need to schedule group meetings. One obvious benefit is that errors by one member of a team are easily seen and corrected by another member. Where team effort is not appropriate, individual instructors may modify materials for their own sections. Copies of handouts can be placed in the student library for easy access by students who may have missed a class.

Examples

Examples describing how many of the activities discussed above are accomplished in the VAX CBE environment, including coding used, are included in the appendix at the end of the paper.

Conclusion and Comments

One of the important points about both the managing and coordinating activities is that they place very little burden on the computer system. The system is primarily used as a file server and at off-peak times. All of the features described can be implemented on networked micros linked to a VAX or MicroVAX II.

In the class management function, the instructor can have a more accurate and timely record of each student's progress and can quickly identify students who are falling behind or are in need of extra help. The end result could be both an increase in student performance and a reduction in the number of drop outs.

Major benefits relating to course coordination include improved quality of written course materials as a result of the cooperative efforts of instructional staff, and orientation and support of new or less experienced instructors in a non-threatening way.

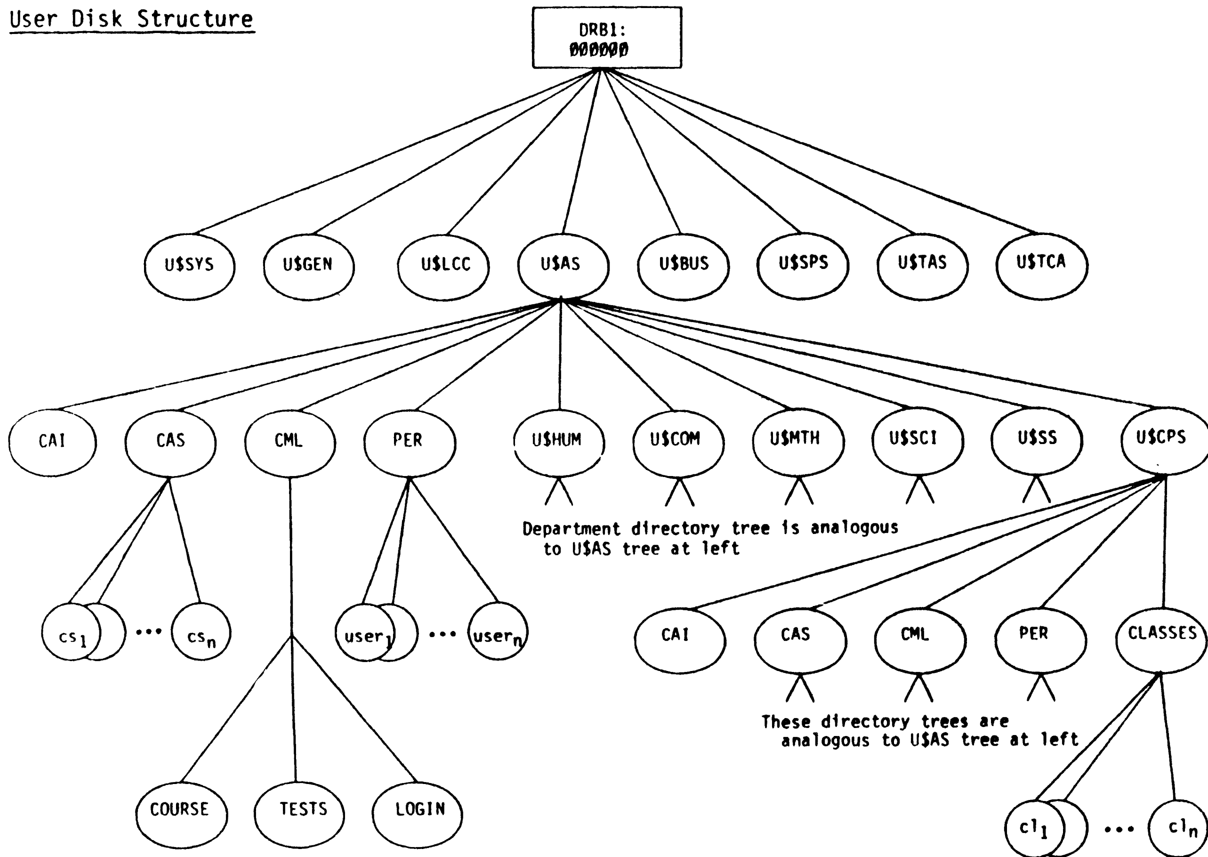



FIGURE 1

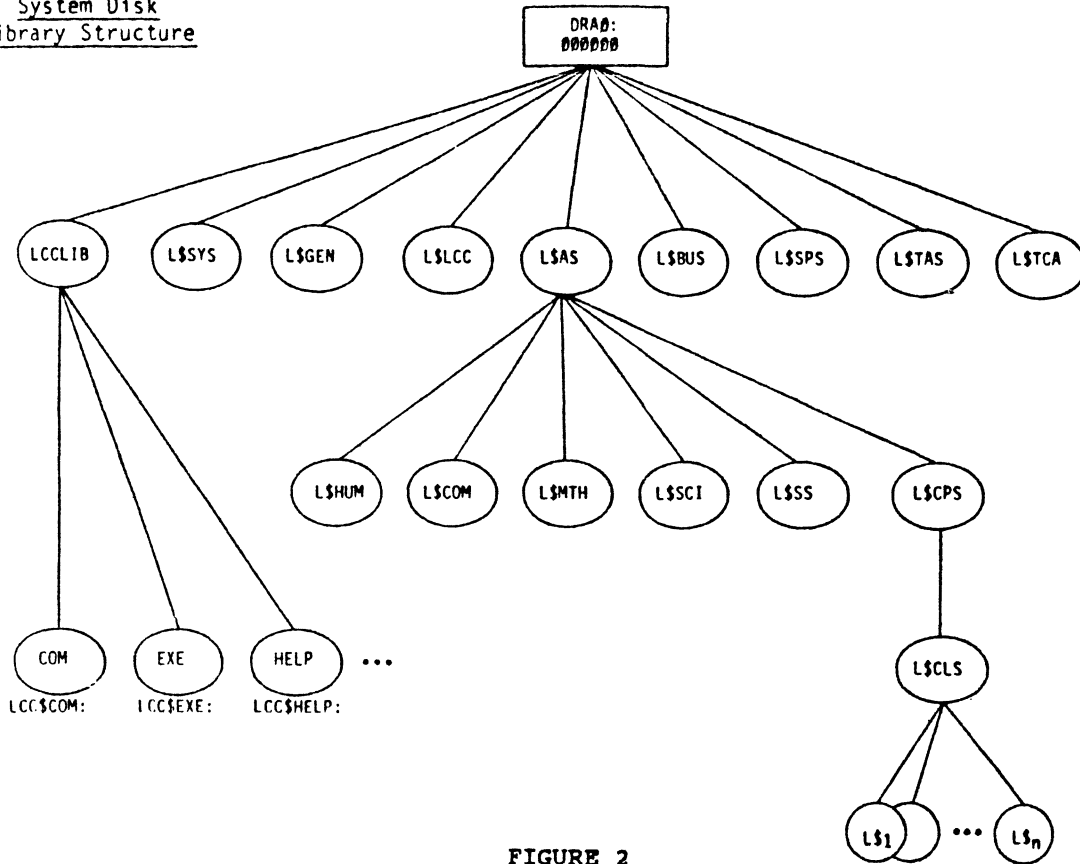



FIGURE 2

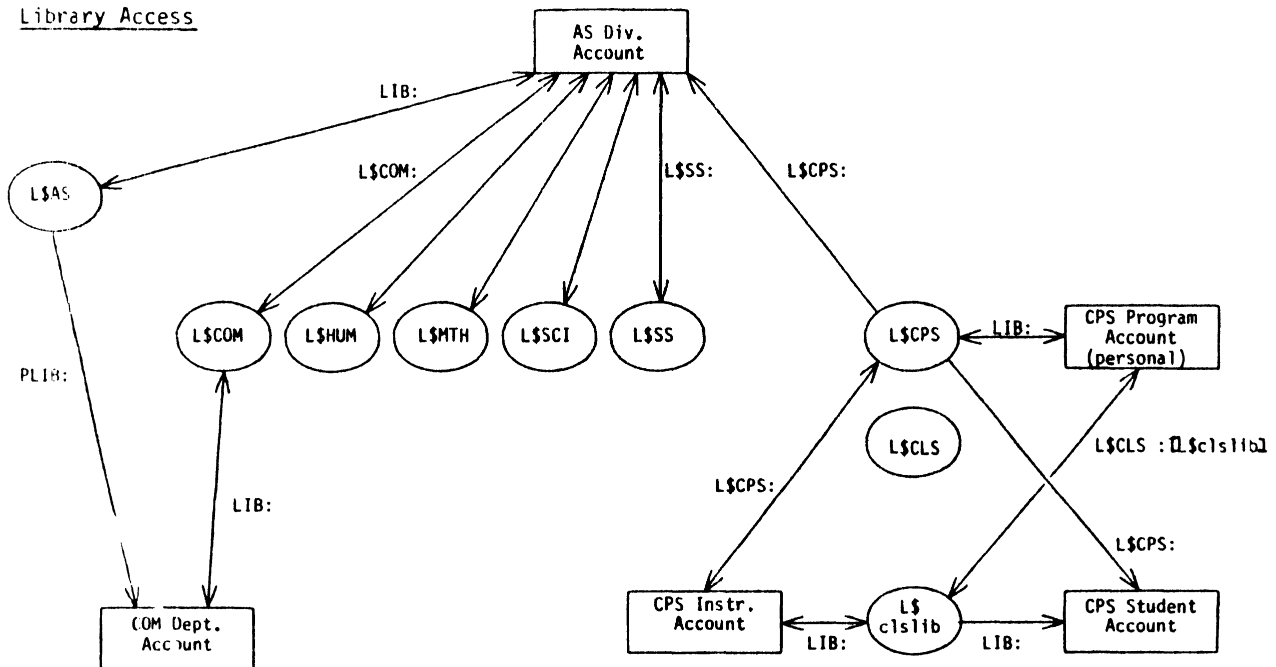



FIGURE 3

Default protection

| | System | Owner | Group | World |
|---|---|---|---|---|
| File protection: | RWED | RWED | RE | No Access |
| Directory protection: | RWE | RWE | RE | RE |

R = READ W = WRITE E = EDIT D = DELETE

FIGURE 4

Examples of commands useful in sharing and communicating

| DCL commands | Local commands |
|---|---|
| COPY oldfile.name newfile.name | DOWN dirname |
| DIR [BASQ.BASQ*] | UP |
| DIR LIB | NEXT dirname |
| PHONE | ENV watson |
| MAIL | ENV AS |
| SEARCH | CAT |
| SORT | WCAT |
| TYPE [BASQ.BASQ*]*.* | MCAT |
| PRINT [BASQ.BASQ*]RESUME.DOC | LIST filename |
| | SEND username |
| | TREE |
| | DIR/CAT LIB |
| | "ENV/DEF/LOG" |

FIGURE 5

EXAMPLE: Communications with Students

Message to students when they log in to the VAX system.

The instructor creates a LOGIN.COM file in the group library. This file is activated every time a student logs in to the system. One of the commands in the LOGIN.COM file is $ TYPE MESSAGE.DOC. The instructor also creates a file called MESSAGE.DOC with the current information to be provided to the students.

Students status check (viewing names of files students have created)

The instructor logs into his leader account BASQ00 and then types $ DIR [BASQ.BASQ*]. The student accounts are numbered BASQ.BASQ01 to BASQ.BASQ40. The asterisk is called a wild card and substitutes for ALL of the account numbers of the class. This command displays the directories of the entire class on the screen.

Student status check (viewing contents of all student files)

The instructor in his leader account, types $ TYPE [BASQ.BASQ*]*.* This will display the contents of all the files from all the accounts. The display will scroll up the screen unless stopped by toggling the no scroll key or hold screen key.

Student status check (viewing contents of one student files)

The instructor in his leader account, types $ TYPE [BASQ.BASQ*]RESUME.DOC. This will display only the files with this name from all the accounts.

Student status check (printing contents of all student files)

The instructor in his leader account, types $ PRINT [BASQ.BASQ*]RESUME.DOC. This will send to the printer the files with this name from all the accounts.

All of the files can be printed with
$ PRINT [BASQ.BASQ*]*.*

Checking the output of a student program.

The instructor from his leader account types
$ run [basq.basq08]progl.exe
This will run the compiled version of the students program.

If the student has source code and not a compiled version the instructor can copy the program to his account with

$ COPY [BASQ.BASQ08]PROG1.BAS *
This will produce a copy of the program in the instructors account. The program can be run with the BASIC interpreter or compiled. If there are errors they may be corrected to show the correct method or to debug the program.

Private message to a student.
----------------------------

The instructor can move to the students directory and use the editor to create a message. The instructor and student can agree on the file names for each to use to exchange messages. The instructor must delete any files he creates since the student cannot.

MAIL is disabled to make it more difficult for students to copy each others work.

EXAMPLES: Course Preparation and Management
-----------------------------------------

Move default to class lib:
--------------------------

The instructor can create and modify files in the class library easiest by moving to the library. This is done by

$ SET DEFAULT LIB        or $ CD LIB   or $ ENV LIB

Files can then be created or modified with the default editor by using the command

$ EDT filename.ext

Move default to Computer Science Library:
-----------------------------------------

Similarly, the instructor types

$ SET DEFAULT CPSLIB    or $ CD CPSLIB   or $ ENV CPSLIB

The dir command will show the files and subdirectories in the department library. Wild cards can be used to narrow down the amount of information displayed. For example

$ DIR *.DIR      will list only the subdirectories.

Finding the Course directory and the Topic subdirectories
---------------------------------------------------------

The subdirectory containing many of the shared files for the BASIC course is called CPS110CRD. After setting default to this directory typing
$ DIR *.DIR will show the directory names. Among them will be CPS100CRD and CPS110CRD.
Typing the command $ DOWN CPS110CRD will move the default directory to this subdirectory. Typing $ DIR wil now give a list of the subdirectories that contain course material.

## Moving a file from the library subdirectory

A file can be copied from one location to another by using COPY and specifing the complete directory name and file name for both the source and destination. This process can made easier by using symbols, logicals, and/or command files.

For example the SAVE command can be defined to run a command file that saves the name of the default directory. The name is saved in the logical "T:" by default or any other letter or letters specified.

The sequence of commands could be as follows

```
$ CD CPSLIB              ! This changes the default dir to cpslib
$ DIR *.DIR              ! This shows the subdirectories
$ DOWN CPS110CRD         ! This changes the default dir to CPS110CRD
$ DIR                    ! This shows the subdirectories
$ DOWN HANDOUTS          ! This changes the default dir to HANDOUTS
$ SAVE                   ! Sets T = CPSLIB:[CPS110CRD.HANDOUTS]
$ HOME                   ! This changes the default dir to login dir
$ COPY T:HANDOUT1.DOC *  ! This copies the file handout1.doc
```

## Examples of Sharing and Communications

Computer students are assigned user names coded so that each section is a separate group. The instructor has a "leader account" as a part of the group. A typical group would be assigned the name CPLITQ00 through CPLITQ40, or BASQ00 through BASQ40 with the leader account being the "00" account. The group shares a class library. In the library a LOGIN.COM file will apply to the whole group.

If the command $ TYPE MESSAGE.DOC is placed in the LOGIN.COM file and a file MESSAGE.DOC created, the contents of this file will be displayed on the screen of each user when he/she logs in. Instructors use this method of communicating with the members of the class on a regular basis.

If the instructor wants to quickly determine the productivity of each student, the command

```
    $ DIR cplitq.cplitq*    <RETURN>
```

will display the names of all the files in all the student accounts.

If the students have been asked to write an assignment or program, the instructor can quickly survey the entire class by looking at the contents of the paper or program with the command

```
    $ TYPE CPLITQ.CPLITQ*:filename.ext   <RETURN>
```

That file, when it exists, will be displayed on the screen starting with the "00" account and ranging through the entire group.

If an instructor wishes to examine a specific student's paper or program, the easiest way is to type:

```
    $ NEXT CPLIT07  <RETURN>
    $ DIR/CAT       <RETURN>
    $ TYPE filename  <RETURN>
      or
    $ RUN filename   <RETURN>, if the file is executable.
```

If the instructor wishes to debug a student's program he/she can copy the program to his/her own account with

```
    $ HOME  <RETURN>
    $ COPY CPLITQ.CPLITQ07:filename.ext *   <RETURN>
```

The instructor can modify the program as much as desired; however the original unmodified program is still in the student's account.

# PLANNING, IMPLEMENTING AND MANAGING A COMPREHENSIVE CAMPUS-WIDE NETWORK

Don Shehi
Maricopa Community Colleges
Phoenix, AZ 85034

**ABSTRACT**

The purpose of this paper is to share with you the way in which a large community college computer system converted from a centralized mainframe computer shop to a decentralized computer network. In 1982, we started with five VAX 11/780's and now have six times the computing power and a communication network that covers the metropolitan Phoenix area.

## MARICOPA'S NETWORK

Maricopa's network started in 1982 with the purchase of five VAX 11/780 computers. Each computer had the capacity to handle 96 communication lines. At this time, the computers were configured using the Digital DZ communication hardware. The computers were installed at five different college sites.

Now let's talk about just what Maricopa is and where it is!

The Maricopa Community College system is made up of seven individual colleges. Five of the seven colleges are full service colleges. By this, I mean they are comprehensive, including university transfer and occupational programs. The sixth college is more occupationally oriented. The seventh college is a nontraditional college with centralized administrative offices but with all of the classes held in remote classrooms. Some of these classrooms are rented for the class period and others are leasted on a permanent basis. Two additional full service colleges are currently under construction. These are scheduled to open in the fall of 1987.

The college system operates in Maricopa County which takes in the Phoenix metropolitan area. The Maricopa Community College District is the third largest community college system in the United States. Our enrollment this year is greater than 70,000 for credit courses. If you include the non-credit courses that we teach, we have an enrollment of greater than 100,000.

The first, and largest, computer site is located at the District office. This is where the centralized accounting, personnel, purchasing, etc., is for the district-wide system. This location is shared with a technically oriented college, Maricopa Technical Community College. The district support center site is centrally located among all our community colleges.

The other four computer sites are located at our largest traditional colleges: Glendale Community College, on the northwest side of the Phoenix area; Scottsdale Community College on the northeast side of the Phoenix metropolitan area; Phoenix College, in about the center of Phoenix, and Mesa Community College on the southwest side of the Phoenix metropolitan area.

The other two colleges, which do not have locally installed computer systems are being serviced by the district support center. One of these colleges is our non-traditional college, known as Rio Salado Community College, whose administrative offices are located in downtown Phoenix. South Mountain Community College is the other college which does not have a computer center on campus. This campus is located in the South central part of Phoenix.

The colleges with computers, each of which started with one VAX 11/780 computer, use their campus computers for student instructional support. They also maintain all student records for online registration at their site.

Now that you have an idea of our size and where the campuses are located, you can see that when we installed the five 11/780 computers we were going to a decentralized concept. At the initial installation of the five sites, we started our star network. We connected all four campus computer sites to the district office via standard telephone lines. We operated these at 9600 baud and used Digital's DECnet for the protocol to connect all of our computers together. We used the DECnet primarily for the application programming staff, the system software staff and electronic mail. All administrative programming support is still centralized from the district office computer site. All use Digital's All-In-1 software product extensively for district-wide electronic mail.

In planning for the computer installation at each college site, a survey was made to determine what the needs were. At that time, our concept was to have all terminals that students used for classwork centralized in large lab areas at each college. Our first local network effort was to direct-wire all terminals to the computer at each site. The remote buildings, such as student services, administration, and admissions and records (online registration area), were connected to the computer building by multiple pair copper wire cables. These cables are like the ones you would see when your local telephone company installs a telephone system. We used line drivers on all lines going to remote buildings. This device will extend a terminal for several miles and solve the problem of electrical differences that one may have between buildings.

The colleges without computers on their campuses are operated by using telephone lines and statistical multiplexers. We found we could support 16 terminal lines at 2400 baud using a 9600 baud telephone line. The Rio Salado administrative office started with one set of statistical multiplexers and the South Mountain campus started with two sets; one for the administration and online registration and the other for instructional support.

The next three years only saw the need for more computers, more microcomputers and more terminals. By the end of the third year, we had a network of ten VAX 11/780s. We had 1,100 microcomputers, 1,000 terminals, and 250 word processors in our district-wide computer system. Terminals were still connected directly to computers via DZ communication devices and the need to have a better communication network was required.

In the summer of 1985, we installed Ethernet at our central support complex. By this time, the district office computer center had grown to three VAX 11/780 computers and they were operating in the Digital cluster.

With three computers at the district location supporting the functions of the district support center and three colleges, we were continually moving terminals from one system to another. Before Ethernet, this required physical cable wire movement. Ethernet gave us the flexibility we needed to move terminals with a simple software change or simply letting the users decide which computer they wanted to use. Ethernet also gave us the ease to add new terminal ports to the district complex at a reasonable dollar rate per terminal. Even with using the new technology of Ethernet, we have still maintained the use of the statistical multiplexer for the colleges served out of the district support center complex. We simply extended the connection from the Ethernet terminal server by use of our existing equipment.

With the successful installation of Ethernet at the district site, which proved to us the usefulness of Ethernet. We extended our plan for Ethernet to our other computer sites.

In January of 1986, we upgraded all computer sites. At the Glendale and Mesa Community College locations, we added a VAX 8600 to each site and installed Ethernet. Each of these sites now has one 8600 for administration and two 11/780s clustered for instruction. Again, Ethernet gives us the flexibility to move terminals to any computer and to add terminals at a reasonable cost.

In this same timeframe, we added three MicroVAX II computers and Ethernet at Scottsdale Community College. They are using the MicroVAX's to support the instructional program. The VAX 11/780 is used to support the administrative computer needs. The Ethernet at Scottsdale allows them to use any terminal on any computer.

Ethernet was also installed at Phoenix College, which now has two VAX 11/780's and a MicroVAX II to support educational and instructional computing needs.

Our two new colleges, scheduled to open in the fall of 1987, are now operating as educational centers. One is being supported by the Scottsdale Community College computers, using 9600 baud telephone lines and statistical multiplexers, and the other is being supported by our Mesa Community College with the same types of communication equipment.

All computer sites originally used Ethernet only in the computer room. Late in 1985, Glendale Community College installed a fiber optic cable from their computer room to their student services building. In this same timeframe, Phoenix College extended the Ethernet cable across campus to three additional buildings. Four more buildings are currently being wired with Ethernet.

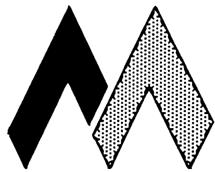
With the increasing needs to teach and use application software that requires the use of a hard disk on microcomputers, we are planning the implementation of DECnet-DOS at two of our campus sites. DECnet-DOS will allow us to use the disk on a VAX in place of adding a hard disk to each microcomputer. We will be using Digital's thin wire Ethernet to connect groups of microcomputers to our backbone Ethernet.

We are in the process of implementing a district-wide library automation system. This system will require an additional 200 terminals to be installed, with a proportionate number per library. These terminals are to be connected to a central VAX 8700 located at the district support center. In library systems, this has traditionally been done by the use of telephone lines and multiplexers. We will be using our Ethernet communication system to connect the 200 terminals to the central VAX.

We are in the process of logically connecting the Ethernet at our five sites. We will be using a digital 56 Kb line and translan bridges manufactured by VitaLink. Through the use of this equipment, a terminal at the Mesa Community College campus will use the Ethernet communication system to connect to the library VAX at the district computer site. The colleges that do not have a computer on-site will have a local Ethernet that is connected to the district complex by the use of the translan bridges. We plan to have Ethernet to all colleges and logically connected via the bridges by the end of 1986.

As you can see, when this project is completed we can install a special purpose computer anywhere in our college system and allow any college in our district to use it.

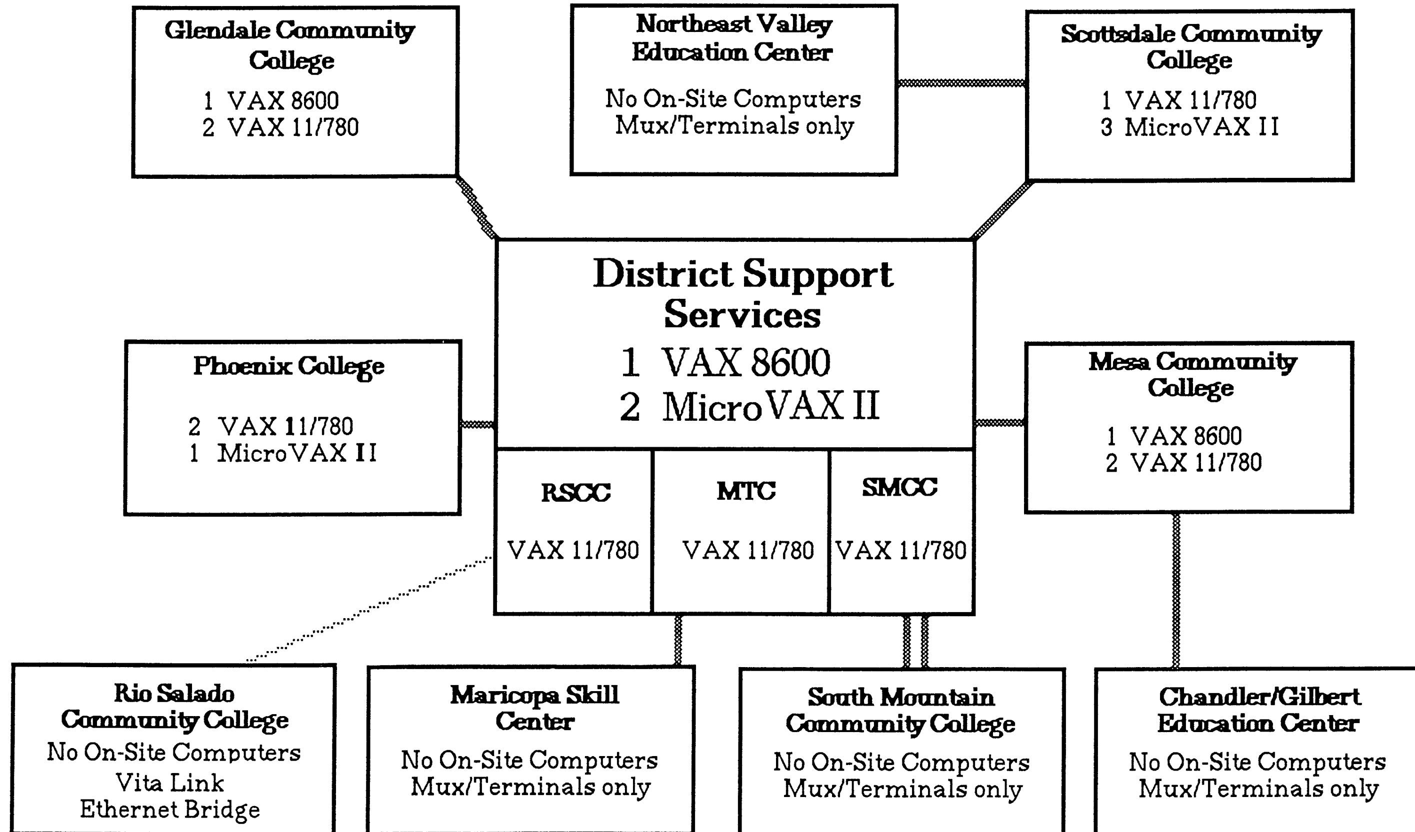
Within 18 months, we will have all colleges connected via our own microwave system. We will allocate a T1 circuit to be used by the bridges. At that time, we will have our 10 Mb Ethernet on each college connected by a 1.54 Mb microwave communication link. In future years, as the requirement for faster links between our colleges become necessary for more distributed processing, we will enhance this microwave system to meet our needs.



Maricopa
Community
Colleges

# Current Computer Network

October 1, 1986




Dedicated Full Duplex Phone Line
Dedicated 56KB Digital Data Circut

# CMU TUTOR

Bruce Arne Sherwood
Carnegie Mellon University
Pittsburgh PA 15213 (412-268-8530)

## ABSTRACT

CMU Tutor is a programming environment which makes it possible for non-expert programmers to exploit the potential of advanced-function workstations. Productivity tools include an integrated graphics editor and an interactive on-line-reference manual.

An integrated programming environment called CMU Tutor has been developed at Carnegie Mellon University. CMU Tutor fills the need for an easy-to-use programming environment that exploits the power of advanced-function workstations, especially in the context of the window-oriented Andrew system developed at Carnegie Mellon in a joint IBM-CMU project. CMU Tutor enables those with limited programming experience to develop interactive graphics applications, especially educational applications, without having to depend completely on the assistance of professional programmers.

CMU Tutor is based on the MicroTutor language developed at the Computer-based Education Research Laboratory at the University of Illinois (the PLATO project). It incorporates MicroTutor's important constructs for interactive educational programming, including easy graphics production, support for diverse kinds of text, rich sequencing facilities, various input analysis routines, and good calculational capabilities.

CMU Tutor uses incremental compilation; that is, only those program segments which have been changed are automatically recompiled. As a result, after making a change in the source code the developer can see the effect right away. Since the source code is compiled rather than interpreted, execution speed is quite fast.

All special text forms supported by the Andrew editor (italics, bold, large, small, centered, etc.) appear as such in the source code, eliminating the need for complex text output commands. A novel graphics editor, tightly coupled with the source code, obviates the need to use a separate command language for creating and editing displays. Automatic scaling to arbitrary window dimensions is supported with options to scale x, scale y, preserve aspect ratio, and scale text. Compile-time and execution-time error diagnostics are very specific, both in the content of diagnostic messages and in pinpointing the location of the error.

An important component of the programming environment is a powerful on-line reference manual for the CMU Tutor language. Language features can be accessed either through hierarchical indices or through names of commands in the language. In either case selections are made simply by pointing with a mouse. Language features are illustrated in context by sample routines. A unique feature is the ability to execute these samples immediately by using the mouse to copy them into the programming window where they can be executed right away. These sample routines can act as nuclei for further elaboration by the programmer.

The Andrew system software operates on Berkeley Unix-based workstations, including the IBM RT PC, DEC's Vaxstation II, and Sun workstations. CMU Tutor source code can be compiled and executed on ordinary IBM PC's and Apple Macintoshes. Although these microcomputers don't offer the programming productivity tools of the workstations, such as the integrated graphics editor and on-line reference manual, the ability to port applications among all these systems is extremely useful.

## REFERENCES

Crecine, J. P. The next generation of personal computers. Science **231**, 935-943 (Feb. 28, 1986).

Morris, J. H., Satyanarayanan M., Conner, M. H., Howard, J. H., Rosenthal, D. S. H., and Smith, F. D. Andrew: a distributed personal computing environment. Communications of the ACM **29**, 184-201 (March 1986).

Sherwood, B. A. An integrated authoring environment. Proceedings of the IBM Academic Information Systems University AEP Conference, Alexandria, Virginia, 29-35 (June 1985). Here it is explained that CMU Tutor gets its name from being implemented in C, with MU being the Greek letter for Micro.

Sherwood, B. A., and Sherwood, J. N. CMU Tutor: An integrated programming environment for advanced-function workstations. Proceedings of the IBM Academic Information Systems University AEP Conference, San Diego (April 1986).

Sherwood, B. A., and Sherwood, J. N. *The CMU Tutor Language, Preliminary Edition.* Stipes Publishing Company, 10 Chester Street, Champaign, Illinois 61820 (1986).

Sherwood, J. N. *CMU Tutor Reference Manual.* Carnegie Mellon University internal report (1986). This is a printed version of the on-line reference manual.

Trowbridge, D. Using Andrew for development of educational applications. Proceedings of the IBM Academic Information Systems University AEP Conference, Alexandria, Virginia, 85-89 (June 1985).

Trowbridge, D. A sampler of educational software at CMU. Proceedings of the National Educational Computing Conference, San Diego, 135-142 (June 1986).

Sherwood, B. A. Workstations at Carnegie Mellon. Proceedings of the Fall Joint ACM-IEEE Computer Conference (November 1986).

# MICROCOMPUTERS: SUPPORT AND OTHER ISSUES

David V. Cossey
Union College
Schenectady, New York 12308

## ABSTRACT

Microcomputers have been touted in many institutions as the cure to all computing problems. They are inexpensive, small and user-friendly. They also come with their own set of support problems, and many of these problems require solutions outside of those normally and traditionally provided by a mainframe-oriented computer center staff. The microcommputer also provides a tremendous opportunity for the college, university, staff member, faculty member, student and yes, for the computer center.

Some of the support problems that will be addressed in this paper include: computer center staff training and retraining, user training (especially college/university staff and faculty), maintenance support, microcomputer laboratory support, delivery and setup, staff incentives, staff overload, software evaluation and selection, hardware and software support.

In addition to problems that need to be solved, there are some unique opportunities that are created for the college/university and computer center. It is exciting to see faculty and staff members get excited about computing. The microcomputer, and all the media attention it is receiving, is making nontraditional computing users take notice. They get the feeling that if they do not get in on the microcomputer revolution, they are missing something. They look to someone for guidance. The computer center has a tremendous opportunity to lead. If the center does not lead, it will be forced to follow.

The work described in this paper pertains to work done while the author was Director of Computing at The Wharton School of the University of Pennsylvania. Since July 1, 1986 he has been Director of Computer Services at Union College.

## 1 Introduction

This paper will describe a project that took place during the Spring and Summer of 1984 at The Wharton School of the University of Pennsylvania. The project involved the largescale introduction of microcomputers at Wharton to support academic, research and administrative functions.

The Wharton School is the business school at the University of Pennsylvania, and the population of potential computer users is divided approximately as follows:

| | |
|---|---|
| Standing faculty: | 176 |
| MBA Students: | 1,500 |
| PhD students: | 400 |
| Undergraduates: | 2,200 |
| Evening students: | 1,500 |
| Administrative/Staff: | 275 |

Wharton installed a DECsystem-10 timesharing system in 1974, and the Wharton Computer Center was established around 1975-76 to support computer use. The University of Pennsylvania is a very decentralized university, and essentially each school is responsible for meeting its own computing needs. There is no central university computer

center. In June, 1984, a Vice Provost for Computing was hired by the university, and through this office a central computing support function is being established.

It became clear that a major commitment needed to be made in the area of microcomputers, and during the 1983-84 academic year serious negotiations and discussions were conducted which culminated in a purchase of over 350 microcomputers. This paper will discuss the implementation of microcomputers and associated services at Wharton.

The Wharton Computer Center, which had been responsible for the DECsystem-10 and its support, was combined with Wharton Audio-Visual Services and augmented with new support personnel to form Wharton Computing and Instructional Technology (known as WCIT). WCIT is now responsible for supporting the large system, microcomputer and audio-visual needs for Wharton.

The selection of computer hardware was done through a process of negotiation with various vendors, and was coordinated by the university. The computer system chosen was the Digital Rainbow-100, and the configuration chosen was as follows:

Hardware:

- Digital Rainbow-100
- 256 KB of memory
- 5 MB hard disk
- Graphics option
- Floor stand
- LA50 dot matrix printer

Software:

- MS-DOS operating system
- Lotus 1-2-3
- The Finalword word processing package

In addition, a lab of twenty Hewlett-Packard HP-150 computer systems was established.

## 2 Preparation

The Rainbow systems were ordered in March/April, 1984 and the first group of twenty-one systems arrived in April. These systems were distributed to WCIT personnel and other key users. The second group of 129 machines arrived in May, and the remaining systems arrived during the remainder of the summer. In all, 372 Rainbow systems were delivered, and the distribution was divided roughly as follows:

| | |
|---|---|
| Public labs | 100 |
| Faculty | 150 |
| WCIT | 25 |
| Other | 97 |

During the time when the project was beginning, responsibilities for support were divided between the Computer Center and Audio-Visual Services. The Computer Center was responsible for hardware preparation, setup, delivery and support, and Audio-Visual Services was responsible for software evaluation, selection and support. A hotline phone number was also established that would take any calls regarding Rainbows. A document was drawn up that detailed all issues that we anticipated as well as possible solutions. This document, while not complete, formed the focus and basis for our subsequent planning and implementation. The members of the Computer Center fulltime staff assumed responsibilities for different aspects of the project.

In preparation for the summer, approximately twenty students were hired to work for the microcomputer project. There were not enough fulltime staff members to handle the scale of the project (eight in the Computer Center and six in Audio-Visual Services). Without the valuable contributions of the students, the project would have been impossible. Many of these students continued on a parttime basis during the 1984-85 academic year.

## 3 Delivery and Setup

The first delivery of twenty one systems went smoothly, except for one problem. Delivery on each order was specified as an "Inside Delivery". However, as each truck arrived the driver pleaded that this was impossible. Faced with a non-delivery, we were forced to unload the first trucks ourselves. Unloading a full truck took between three and four hours, with 10-12 people. The later deliveries were "inside" deliveries.

There is no loading dock at Wharton, and there is a lack of storage space available. For the summer, we were able to get classrooms to use for storage, setup and testing. These rooms were equipped with security systems.

After a delivery the classrooms were piled to the ceiling with boxes of

Rainbows (each system came in approximately 6-7 boxes). The Rainbows were unboxed, assembled and tested before being delivered. They were not "burned in." Each Rainbow came with a one year warrantee, and most problems showed up in the first 30-60 days of use. An ongoing problem was that of keeping track of serial numbers (system, keyboard, monitor and printer).

## 4 Software Decisions

It became clear that the MS-DOS operating system would be the one that we would support (the Rainbow also came with CP/M). Also, Lotus 1-2-3 would be the spreadsheet package that would be supported. This was the almost unanimous choice of the faculty and students, and we did not seriously evaluate any other spreadsheet package. In the area of terminal emulation/file transfer packages it was decided to support the KERMIT protocol since we already had KERMIT on the DECsystem-10 and the VAX 11/750. LCterm was a public domain package that provided terminal emulation and both the KERMIT and XMODEM protocols. LCterm is now sold commercially, and it is also available for the IBM PC. Other areas where the choice was not as apparent were database and word processing packages.

### 4.1 Evaluation and Selection

Since many faculty members owned IBM PC's, and this trend would probably continue, it was decided to only evaluate software packages that ran on both the DEC Rainbow and the IBM PC. Although this limited our choices at the time, it was felt that this would make future support easier.

The most pressing need was to choose a word processing package. A list of selection criteria was drawn up, and approximately eight packages were evaluated. The final choice was between The Finalword by Mark of the Unicorn and WordPerfect from SSI. We chose Finalword. There are now additional packages that would be considered that were not available in 1984. One of the features available in Finalword was a means to protect the user from inadvertent loss of work in the case of a power failure or inadvertent powering down of the system. This feature has since been added to WordPerfect.

In the database area the package chosen was KnowledgeManager from Micro Data Base Systems, Inc. Again, today, there are other options that would be considered.

Some of the criteria used in evaluating software were:

- Availability for both the DEC Rainbow and IBM PC
- Availability under MS-DOS for the Rainbow
- Ease of use
- Flexibility
- Price
- User-friendliness

### 4.2 Support

A telephone hotline and a walkin support area was established. Documentation was written to provide quick and easy introductions to the various packages. In the case of Lotus 1-2-3, users were pointed to the Lotus-supplied tutorial as an introduction. For the Fall, short courses were established for the various packages. These short courses now include:

- Introduction to Microcomputers

- Introduction to Spreadsheets
- Introduction to Lotus 1-2-3
- Lotus 1-2-3 Database facilities
- Lotus 1-2-3 Graphics
- Lotus 1-2-3 Macros

- Introduction to Finalword
- Advanced Finalword

## 5 Training

When the microcomputer summer project began, there was a lack of fulltime staff members to support the effort. During the summer, the fulltime staff performed their "old" duties as well as new duties. While this effort could work for a time, it could not be sustained through the academic year that began in September, 1984. Additional fulltime staff were hired, but most of them did not begin work until September, 1984 through Spring, 1985.

One of the functions that needed to be provided was for initial training for faculty and staff. The decision was made to contract with an outside vendor for these services. Three vendors were contacted, and Digital Equipment Corporation's local Software Services group was chosen to develop a one-day training session and associated materials. The one-day session was most convenient for the majority of those who would receive the training. It would

have been difficult to sustain a program spread out over several days for 2-3 hours per day.

The course was divided as follows:

Morning

Introduction to Microcomputers
Introduction to MS-DOS

Afternoon

Introduction to Finalword
Introduction to Lotus 1-2-3

Everyone who was going to receive a Rainbow system was required to attend a session. The sessions ran for approximately 30-35 days, and attendance was limited to nine per session. There were two instructors available at all times - one instructing and one available for individual help. Each "student" had his/her own Rainbow system, and the instructor had a Rainbow connected to a video projection system connected to two large monitors. The session followed the prepared materials, so that there was a constant reinforcement of the concepts presented. This also eliminated the need for the student to take copious notes.

Delivery of the Rainbow systems to an individual's office was coordinated with the instruction. The Rainbow was delivered the day before the person was scheduled to take the course. We did not want the systems to show up too many days before an individual received instruction, since we did not want the individual to be overwhelmed by the amount of documentation delivered. On the other hand, we wanted the student to know that the system was in his/her office, and that immediately after the class the system was available for use.

The classes were well-received. The audience was mixed (those with no computer experience and those who had used computers for years were in the classes). Although it would have been nice to separate the new users from the more experienced ones, it was not practical to schedule.

## 6 Maintenance Support

The Rainbow systems came with a full year's warranty, which alleviated the need for us to find a vendor to service the systems during the first year. This year also gave us experience in diagnosing the problems, and many of these turned out to be software related. What emerged was a system whereby we would take a call on the hotline, and we would send someone to the office to diagnose or fix the problem. If the problem turned out to be a hardware problem, we would swap the faulty part (we kept back some machines to use as "swappers") or the entire machine. The technician from Digital Equipment Corporation would then swap the faulty part for a good one when he made his service call to our "depot" location.

By attending to the machine immediately, we were able to reduce the downtime experienced by a user. Thus, we were able to reduce some of the frustration of firsttime computer users.

## 7 Staff

When the project began, we were not staffed at appropriate levels to provide ongoing support for the microcomputer use at Wharton. Thus, immediate steps were taken to hire new staff, and to set up a new organization, built around the existing Wharton Computer Center and Audio-Visual Services. However, it takes time to assess one's needs and also to hire people. Most of these new people were hired after the summer was over.

### 7.1 Needs

We divided up the responsibilities for microcomputer support within our Operations, User Services (newly developed) Technical Services (Systems) groups and Audio-Visual groups. The bulk of the support eventually centered in the Operations and User Services units. This has since evolved into a model in which most of the support is centered in the User Services group. Network and communications support is provided out of the Technical Services group.

We felt the need to hire a fulltime consultant within the User Service group with responsibility for microcomputer support. We also hired a fulltime person with responsibility for logistical and operational support for microcomputers. These fulltime people were supplemented by many student parttime employees.

### 7.2 Training

Most of the initial training for microcomputers was done by individuals on an ad hoc basis. Later, as formal courses were developed (such as the one-day training session), staff members were encouraged to participate. Enthusiasm was high, and most staff members did not want to be left out. It was an exciting, but hectic, time.

### 7.3 Incentives

Incentives for the efforts during the summer were mainly intangibles, but bonuses were obtained for those who participated heavily in the project.

## 8 Opportunities

The introduction of microcomputers at Wharton provided many opportunities for WCIT. It was a time in which many people were being introduced to computing for the first time. There were many people who had never used computers before, and many others who had never used a microcomputer. There were others who were very frustrated with computers because of bad past experiences. Thus, a goal at WCIT was to make the experience as enjoyable and as painless as possible.

We had an opportunity to show that we could be responsive to the needs of individuals and departments. I received more thank yous and commendations from this project than I had in my previous five years at Wharton. Part of this was a result of increasing the base of computer users at Wharton perhaps by a factor of three or four.

Many computer centers have faced the introduction of microcomputers as a threat rather than as an opportunity. They will be the losers if they persist. Microcomputers are here to stay, and they will continue to pervade the academic and business environment. The computer center can follow or lead; it cannot afford to ignore.

# STATE OF WASHINGTON SCHOOL NETWORK

Al Huff, Executive Director
Washington School Information Processing Cooperative
Lynnwood, Washington 98036

**ABSTRACT**

This paper briefly describes the WSIPC organization and four aspects of its operations; namely its equipment and network strategy, in-house maintenance, centralized systems software support and the software development environment.

The WSIPC (Washington School Information Processing Cooperative) is a rather unique agency in the public sector in that it exists solely to provide data processing services to the State of Washington K-12 Public School system. It has its own Board of Directors which represent the state geographically in their individual appointments. WSIPC contracts with 1) the state Office of the Superintendent of Public Instruction, 2) the nine Educational Service Districts in the state, and 3) 270 of the 298 local school districts to supply services. WSIPC receives no funds other than what is earned via the above contracts which are mostly signed on an annual basis. So far as I know, WSIPC is unique in providing software services through an integrated network to all three tiers of the school hierarchy.

WSIPC currently operates with an annual budget of about $5,000,000 and employs 60 people. The software applications are payroll/personnel, general ledger, budgeting, purchasing, payables, receiveables, inventories, warehousing, student grading, scheduling, transcript, attendance and general demographics, special education and user defined data bases, state reporting in all areas and electronic mail services.

WSIPC has created a DEC VAX network, utilizing DECNET to facilitate processing and support up and down this structure. It should be understood that the software developed by WSIPC in the administrative area is highly integrated with state reporting requirements and by its nature, would not be suitable for use in another environment without significant modification. So what is of interest to members of DECUS? I believe that there are at least four things which WSIPC has developed which are useful for other service agencies. They are:

1. A practical, useable hardware model.

2. An example of in-house maintenance and systems integration.

3. A successful model for centralized systems support over a distributed network.

4. A successful evolutionary applications software development model.

The WSIPC hardware model is relatively simple due to the nature of DEC's networking and configuration tools. The theory at present is that computing ought to be as close to the end user as is economically possible and therefore, the 19 VAX systems are spread around the state. They are generally in the population centers with additional local access provided by statistical multiplexors in the secondary population centers. Further, where school districts desire more computational power, they have a VAX on their premises and/or PDP 11s in their secondary schools. The PDP 11s are not networked and are scheduled for replacement by VAX systems.

All VAX machines are permanently linked into the communications network through Digital Router Servers. They are either connected to the Router Servers via leased lines and 9600 Baud modems or on Ethernet. With this straight-forward "star" type network, every VAX is virtually one step away from any other VAX and the state governmental pyramid is relatively simple to implement. We found that using pass-through VAX nodes on DECNET was undesirable for performance and that the network is much more stable with the nodes arranged so that no single node is dependent upon another to function in the network. Since we do not have critical realtime considerations, it is acceptable to have a node out of the network temporarily. The only central failure point is one of the Router Servers and they are not a problem. They have few components which might fail and they are engineered to allow on-the-fly plug-ins. The long term network configuration will have several Router Servers connected via high-speed links. This topology will be predominantly influenced by costs of the communications lines and is almost infinitely flexible.

A typical configuration for a VAX node is an 11/785 with 24MB of memory, one system pack and a three or four disc packset for user data (or a gigabyte of storage), two

high speed line printers, two 6250BPI tapes and around 64 or 80 communication lines which are a mixture of hardwired, local dial-up and multiplexed lines. Such a configuration can handle the interactive and batch work of about 50 concurrent users before response time degrades. The network is mostly 785s with three 750s, three MicroVAX IIs and two 8300s. All systems have a DECNET connection.

WSIPC is somewhat unusual in that it has its own maintenance group of five employees. This activity began when the previous Xerox Sigma 9 systems had little commercial support available. Then, the expansion of our communications equipment and lines made it desirable to have the ability to troubleshoot and fix modem and multiplexor problems in-house. The service evolved into printer maintenance and when difficulties occurred with two different PDP 11 integraters, we stepped into first maintaining those systems and now to integrating MicroVAX II systems for use in the network. This group will also maintain the MicroVAX IIs. While this group does not maintain the other VAX CPUs, they do work directly with Digital. We have a single maintenance contract for all the CPUs and Digital disks. Our experience with this indicates that our credibility with the user is increased because we can address most any problem without having to ask multiple vendors to isolate problems which may be either software, hardware or communications. So far, this has proved to be less expensive and more responsive for our network. A great deal of site planning is done by this group as well.

WSIPC supports about 35 PDP 11 systems and 19 VAX systems with one centrally located systems software group of five people. These people must be experienced in handling operating systems in order to be successful. Almost none of the other sites have a systems person on staff. In our opinion, they are not required. WSIPC staff use the network to monitor and tune VMS. They use the network with automated routines to gather device error statistics and to measure workload. Problem resolution is often done over the network as well as occasional distribution of software. We have found that great economies are possible with this arrangement. The staff gets much more experience than single site people can and they have ready comparisons when aberrations appear. They are mutually supportive in that they can afford to specialize in particular areas and share the expertise.

In addition to the systems management use of the network, WSIPC uses the network as the vehicle for transferring files which are usually reporting instruments destined for the State Office from the local school districts. This process can work in reverse as well. The network also is used as an electronic message switch under All-In-1 and our own bulletin board.

The applications software development model has evolved over the years. Several years ago, we found ourselves totally tied up with maintenance and enhancement work in existing code. This often led to difficulties in implementing related pieces of code which were in maintenance and at different points. No new development was possible. We were forced to regroup and evaluate the process. We decided that reliability was the number one goal. That meant the software worked like it was documented, that systems were up, that phone lines worked and were not all busy and the whole gamut of what the user perceives when he or she wishes to do work. That led us to a more planned approach to applications development. We promised that existing code would only be worked on once per year and it would be stable the rest of the year. All modifications and enhancements would be done at the same time. We called it routine maintenance and set out an annual schedule. All requests for enhancements would wait for the scheduled cycle. If the number of enhancements exceed the amount of budgeted routine maintenance time, then the project team must make the decision on deletions prior to the start of the project. All reports of deficiencies or bugs would be classified as critical or not-critical. If they were critical, an immediate fix would be made. If not, they would wait along with the enhancements for the routine maintenance schedule. This very quickly stabilized the applications code and made it more reliable. Further, it became possible to schedule new development and complete it. It should be stated, that it is important for all projects to be estimated in advance and then agreed to by the project team. No open-ended projects are allowed.

The WSIPC applications are coded in Cobol for the financial side and Fortran 77 for the student record side. Several productivity tools are employed such as PCA, Language Sensitive Editors, TPU, Symbolic Debugger and so forth. These are much appreciated tools. The applications are well established and contain about 1,000,000 lines of code. So far, our analysis is that 4GLs are still not adequate for our purposes. This could change and is a continuing subject of research.

WSIPC has a far flung user community and must formalize the process for instituting changes or enhancements or new programs. All such requests are submitted in writing and get the sign-off of the district's chief administrator which in and of itself deletes some of the chaff. Then enhancements are handled as the routine maintenance schedule can accommodate if the enhancement does not negatively impact other users. Requests for new applications are given to a representative committee of users who reject and or prioritize them.

Then our Executive Committee determines how much of the newly prioritized work it can fund. Once projects are funded, they are scheduled and turned over to a team made up of Applications Analysts and Product Support people. The Product Support people have control of the objectives of the project and use several devices for eliciting user input on design objectives. Once the two groups have agreed on the design, the Product Support people begin writing user documentation and the Applications people begin coding. All code is turned over to Product Support for final approval and validation prior to release. No code is released without the accompanying documentation so these groups must work in tandem and act as a check on one another. The Product Support people then sign off on program release which occurs every Friday evening in a pre-determined manner which requires routine operations. Product Support also schedules training for the field user trainers and carries that out. We now can tell users just where projects are and give expected release dates with about 80% certainty. We have gone from a situation of being almost overwhelmed to a highly predictable process which gets much more accomplished.

# "E067 ENHANCING CAMPUS AND COMMUNITY COMMUNICATIONS THROUGH VOICE, DATA, AND VIDEO TELECOMMUNICATIONS"

J. D. Thomas
Freed-Hardeman College
Henderson, Tennessee

## ABSTRACT

Freed-Hardeman College extended campus and community communications in 1986 in one of the first educational implementations of the DECconnect cabling scheme, providing both reduced costs and enhanced services to faculty, staff, and students at 887 multioutlet faceplates. A voice-data PBX with LAN features for occasional users of the central resources, an Ethernet-DECnet LAN with terminal servers, and continued direct wiring of some workstations now give on and off-campus computer access. Parallel fiber optic, twisted pair, and coaxial cables and an empty conduit buried between all buildings support enhanced telephone and CATV services and provide for expansion of the Ethernet and any future technologies.

## THE SETTING

Freed-Hardeman College is a four-year liberal arts college related to churches of Christ. It is located in a small town in rural West Tennessee, twenty miles south of the regional center of Jackson and about eighty-five miles northeast of Memphis. Originating with the Henderson Male and Female Institute--proudly but atypically coed when chartered as a high school and college in 1869, it has always offered liberal, character, and career education. General education no longer means the classics, but there has always been a strong core curriculum. Teacher, business, and ministerial education have long been featured. Of the 40 major programs of study, mass communication, social work, and computer studies now also attract significant numbers of students. The Computer Science major is based on ACM recommendations, and the Computer Information Systems major on DPMA guidelines. Both are organizationally in the Department of Mathematics and Computer Science, and in the fall of 1986 majors were equally divided between the two programs.

The college draws its 1083 students from two-thirds of the states and about ten other countries. The students and a faculty of 89 full- and part-time teachers and administrators, along with student spouses, and other staff employees, have a substantial impact on a town of under 5,000 population in a county of 11,000. Many local citizens are alumni or have some connection with the college or with its personnel. There are about 25 main buildings on a 100-acre campus.

## COMPUTING ON CAMPUS

Combining Title III, Higher Education Act, and non-Federal funds, Freed-Hardeman has since 1981 acquired and put to good use a significant amount of computer equipment. Counting students and employees, there is approximately 10:1 person to workstation ratio. Except for some portables bought primarily for evaluation and for Apple IIe's used in an introductory applications course for nonmajors, in teacher education strategies courses, and in a few other departmental settings, most of the equipment bears the Digital logo. All but about 50 of the nearly 200 microcomputers, terminals, and printers in ten of the buildings have been hardwired to one or the other of three DEC minicomputers located in the Computer Center--a PDP 11/70 used for administrative applications, a PDP 11/44 used for the library, and a VAX 11/750 dedicated to faculty and student use. Major labs open to students from all departments are equipped with VT100s, Rainbows, or VT241s. The color terminals in the Graphics Place lab are used by students completing assignments programmed by faculty from all fifteen academic departments during a series of three 5-week summer workshops on Digital's Courseware Authoring System. Three-fifths of the faculty have some graduate or on-campus computer training; the published objective for the eighties is for three-fourths of the faculty to be active users of computing.

## TELECOMMUNICATIONS PLANNING

The telecommunications project under way grew out of long-range planning; financing for a subset of it was included in a capital campign. Computer Services has been stretched to meet faculty and student requests for access to the academic VAX mini, and programs being developed for use by advisers and others for library searches by all faculty and students require access also to the PDP 11s. Replacement of a deteriorating PBX and of outside cable buried or strung by at least three different companies over the past thirty years was an urgent need verified by a firm of consulting

engineers and by the experience of trying to maintain a switch and wiring no longer supported by the manufacturer or vendor.

Vendor Selection

A telecommunications planning committee consisting of four vice presidents and the director of computer services began meeting in the fall of 1985 with DEC and telephone marketing representatives. The vision of comprehensive recabling of the campus and of enhanced services developed when pre-formal announcement of the DECconnect cabling scheme was presented by DEC sales and networking personnel. The opportunity to do something more significant than replacing the old PBX was attractive.

To preclude the possibility of buying another PBX which might be orphaned or which might have to be serviced by technicians from distant cities, the committee solicited proposals only from AT&T Information Systems (ATTIS) and from South Central Bell Advanced Systems Incorporated (SCBASI)--the nonregulated subsidiary of the regional Bell holding company, BellSouth. Advanced Systems made an office-by-office study of existing patterns of telephone and computer use and proposed--from among the various manufacturers lines they distribute--a Northern Telecom (NT) Meridian SL1-N voice-data PBX. ATTIS spent less time on campus, but prepared a very competitive bid for a System 85. In the final analysis, South Central Advanced Systems was chosen because (1) they were a DEC OEM with access to DEC information and expertise, (2) Northern Telecom and Digital had jointly developed the CPI computer-to-PBX interface, and (3) ATTIS seemed to be unduly interested in introducing their mini- and microcomputers to the campus and emphasized Unix and interfaces with IBM PCs and true compatibles--of which we had only one, a Compaq portable bought for a software transporting project. The first consideration became a detriment as DEC district personnel feared loss of control of the college account and of credit for any sales. The second became insignificant when a decision was made not to use the CPI product.

Needs and Opportunities

There were three major communications services problems: (1) the PBX was obsolete; (2) hardwired access required choosing one of the three timesharing computers; and (3) cable TV service was offered by the local community antenna television (CATV) company to only three of nine residence halls. Tone dialing was not available on campus or through the local central office. Although there are ways to combine pulse and tone signaling to use remote databases and discount long-distance services, the equipment and technique seemed awkward and many unidentified calls were placed more expensively with 1+ access to AT&T.

In addition to solving the problems identified above, the telecommunications study committee saw other opportunities. These included (1) universal telephone service for students; (2) elimination of high installation and service change fees and deposits charged students by the telephone company; (3) obtaining less expensive long-distance service and sharing the discount with students; (4) reducing the cost of cable TV service; (5) using a campus cable network to distribute educational and local origination programs; (6) reducing the incremental cost of adding and connecting a terminal or a personal computer; and (7) giving students access to the academic VAX computer when laboratories and buildings are closed.

Advertisements of the availability of National College Television (NCTV) and of educational and documentary collections such as the Video Encyclopedia of the 20th Century added to the interest in the project. Broadcasting students and faculty are interested in significant or realistic learning activities, and the prospect of producing documentaries as well as live originations was exciting. Library AV personnel liked the idea of transmitting video to classroom monitors without moving either VCRs or students.

System Design

The cabling, networking, and user equipment plans were designed by the computer and telephone vendors and the local CATV company, in consultation with college personnel. Each of them underwent significant design changes as vendors came up with what they believe to be better ideas.

Cabling - Unshielded twisted-pair copper wire, four-fiber Siecor optical cable, and standard CATV 75-ohm coaxial cable were direct buried in the summer of 1986 between all buildings. Fiber at first was to range from 24- to 4-fiber with dropoffs at buildings between the center and the extremes, but was reconfigured to use 4-fiber cable in a multiple-hub or multiple-star pattern. Independent engineers reviewed the design and expressed the opinion that it would support any foreseeable needs. The committee realized, however, that nothing is as unforeseeable as the future. Alongside is a 1-inch flexible plastic conduit for any replacements or future developments in cable technology. You could say we have four nets--a fiber net, a coax net, a twisted pair net, and the conduit as our safety net.

Faceplates - Three or four twisted pairs and coaxial cable are terminated at each of nearly 900 triplex faceplates. The DECconnect scheme brings all cabling into a Satelite Equipment Room, called by telephone people an IDF (intermediate distribution facility), including the fiber. Data is connected to the using locations on twisted pair or thin-wire Ethernet. Delivery of the four-port faceplates in sufficient quantity could not be assured by Digital, so SCB Advanced Systems had triplex faceplates custom-made for installation in residence hall rooms, classrooms, laboratories, and most offices. From top to bottom are telephone, data, and video jacks. We did get enough quad-outlet DECconnect faceplates for the CIS faculty offices in the Education Center--a building being renovated during the summer.

Ubiquitous campus-standard multioutlet faceplates eliminate the need to decide in advance where voice, data, and video will be needed. They do not necessarily provide the best solution for a particular location. In many residence hall rooms, for example, the TV outlet needs to be on the wall opposite the telephone and computer connections, so we have some duplex "RJ" faceplates with video jacks at a separate location. It seems unlikely that thin-wire Ethernet for high speed end-node connections will be needed in all locations. Three

different types of phones used are supported by different PBX software and line cards, so phones may not be moved at will or without special programming.

Central Office Support - To support the college's new telephone system, South Central Bell installed a Northern Telecom DMS-10 central office switch--bringing 7-digit tone dialing, equal access, and extra-cost options to the community. The biggest threat to community relations was the elimination of 4-digit dialing in the local exchange. Seventy trunks bring T-1 service from the central office to the campus; the copper originally used has been replaced with LightGate fiber.

TELEPHONES, DATA CONNECTIONS, AND SERVICES

Direct inward dialing, call forwarding, easy conferencing, and a message desk for faculty and other office telephone users have greatly increased the probability of talking with or getting a message to a teacher. Electronic mail will be extended as more desks have computer screens.

Telephones are sufficiently state-of-the-art that the vendor and installers were unfamiliar with them, and many of them have been relabeled and reprogrammed after placement in offices. Eighty-seven are M2009s which have RS232-C data jacks ready for a terminal or microcomputer connection. They use only one pair of wires between the office and the switch room to provide independent voice and digital data transmission. (See figure: LANSTAR Connections.) Voice and data are then terminated separately on special line cards. The M3000 multifeatured Touchphones promised the vice presidents are recedingware--apparently beyond the state of the art. Pictures of them have been available for more than a year, but distribution has been delayed by problems with LCD screen visibility. Other casual data users will have computer connections through the second RJ-11 (that is, standard telephone) jack. An AIM (asynchronous interface module) cable may be used without a modem for dial-up access to other computers--mini or micro--through the Meridian LANstar features of the PBX switch, or the user may be conneced to a terminal server in his building's equipment room. Heavy users with single access needs--such as accounting--will continue to be served with direct-wiring into minicomputer ports.

The NT Meridian SL1-N was placed in service for testing in mid-August of 1986--two or three weeks after the central office cutover. The previous system was left in operation through the end of August, providing backup service during orientation, training, debugging, and the relabeling and reprogramming which proved to be necessary.

Long Distance Service - Students--except for the dozen from Alaska, which is outside WATS band 5, and for international students--buy toll service at a discount from the college. We have initially contracted for service and billing with a WATS reseller based in Jackson, Mississippi, but with offices and service in nearby Jackson, Tennessee. LDDS offers toll service to the U. S. Virgin Islands, but the college's half dozen Virgin Islanders are from the British Virgin Islands. Speed call dials the local access code, but each caller must input a personal or office authorization code.

Computer Service - Data connections and transmission through the switch have been demonstrated but not yet widely implemented. As soon as the Ethernet-DECnet backbone connecting the three DEC minicomputers with each other and with the PBX has been installed, the need to activate Ethernet hubs and to switch users to terminal servers will be evaluated. Use of the direct CPI connection between the VAX and the SL1-N was rejected because of the number of ports it would have utilized and because it appears that reverse terminal servers will enable the PDP 11s to be accessed as well as the VAX. RSTS/E is not yet supported on DECnet, so this solution seems to be the best available.

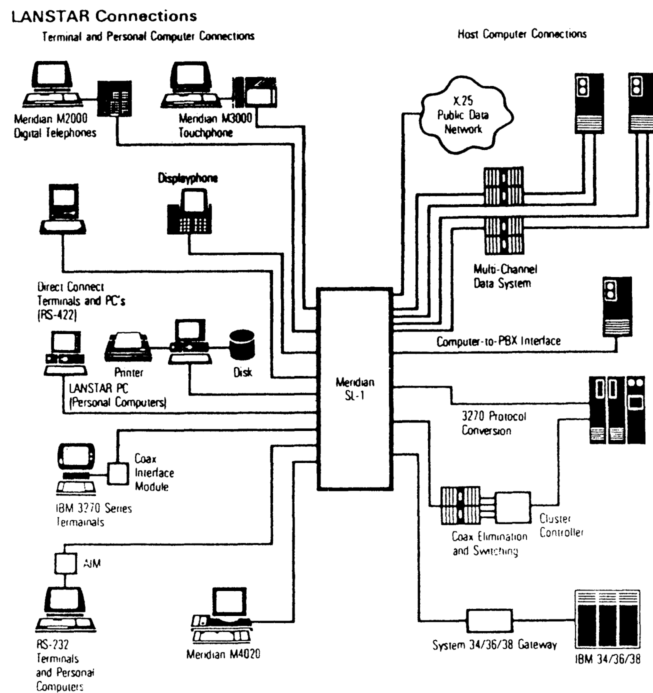
Message Service - One of the best-received services associated with the new system is a message desk. Calls coming into the campus to office phones not answered by the fourth ring revert to the campus operator, who offers to connect the caller with another party or the message center. Every office phone has a message waiting indicator. After-hours and weekend calls are recorded and transcribed at the beginning of the next business day. Messages are entered into a computer file and retrieved and read to the employee by the message desk attendant. When ALL-IN-1 and DECtalk Mail Access--now deferred--are installed, those with a terminal or micro screen should be able to retrieve their messages through the ALL-IN-1 menu. Others may use their Message button to autodial the access codes necessary to have DECtalk read messages to them over their telephones. We have examined voice-store-and-forward systems, but they do not seem necessary or cost-effective for us now. The message desk is located in Computer and Telecom Services and makes it easy to communicate requests. It is more difficult to fulfill them quickly.

Extended Services - Technology will support additional services to faculty, students, and others in the community, off- as well as on-campus. These include voice, data, and video services. Negotiations are under way with Essex Cable TV as to the periods on the time-weather channel which will be made available for college-originated programs or NCTV.

Personal Computing - Announcements continue to be made about hardware and software interfaces between the Northern Telecom Meridian PBXs and Apple Macintosh, IBM PC, and other micro- and minicomputers. These are being studied carefully in order to develop a plan for promoting personal computer purchase and use by students. Many of their questions have gone unanswered as the college takes preparatory steps to give them authorized access to some files and programs on each of the three central DEC minicomputers. Most students are content to do their computing in one of the laboratories or off-line for a while.

PROGRESS AND PLANS

When resources permit, a shared computer room with monochrome and color terminals, micros, and a printer connected through a terminal server will be




located in each residence hall. These rooms have been identified and prewired for multiple data connections. Some things such as this will have to wait until the need and the feasibility coincide. Staging full implementation has also been necessary because of delays in installation, in testing of fiber and data capabilities, and in deliveries, some of which are related to fiber and construction.

Voice communications between faculty and staff and between faculty and students--who may leave but not receive messages--have already been facilitated. Data communications have been enhanced with a dial-up choice of computers. The video system will have multiple points of origination and some 900 potential points of reception on campus in addition to the 1350 community subscribers. In a town without a television station, there is considerable interest in an evening telecast of local news and features. Videotext is feasible and could be in place by next year.

Enhanced campus and community communications through voice, data, and video telecommunications is a prospect--not just a dream, but like many of our goals in higher education it is not yet a reality. Freed-Hardeman College doesn't have it all now, but we are pleased with what we have and what we expect.

# DATA MANAGEMENT SYSTEM FOR ACADEMIC INSTRUCTIONAL PLANNING

Lisa M. Rotunni
Edward C. Hohmann
James A. Rounds
School of Engineering
California State Polytechnic University
Pomona, California 91768

The School of Engineering at California State Polytechnic University, Pomona offers approximately eight hundred course sections within sixteen academic disciplines during each of four quarters per year. A data management application, SIPS (School Instructional Planning System) has been written in the database management program RDM to aid in coordinating the information necessary to schedule courses, provide classrooms and laboratory facilities, arrange faculty, and communicate effectively with the University and Departmental administrations. This paper will discuss the planning, programming and operation of SIPS as well as the difficulties and the benefits of applying database technology to this administrative situation.

## OVERVIEW

The School of Engineering at California State Polytechnic University, Pomona is one of the largest in California. It offers approximately eight hundred course sections during each of four quarters per year, serving roughly 4,300 full and part-time students. The University uses a computerized system to enroll students in courses. However, before that point is reached, a large administrative effort is required to schedule courses, provide faculty to teach them, and arrange the classroom and laboratory facilities required.

Since the Summer of 1984 we have made a determined effort to take advantage of computer technology in performing administrative tasks within the School of Engineering. The School acquired a computer system consisting of a PDP 11/23 processor with 1 megabyte of memory and a 76 megabyte winchester disk, 7 DEC VT102 terminals, a dot matrix printer and two laser printers. The total value of this system is under $30,000. The school computer system uses the RSX multi-user operating system, and includes packaged word-processor, spread-sheet and database management programs.

As part of our computerization effort, we have developed a data management system to help us perform course scheduling tasks: SIPS, the School Instructional Planning System. SIPS is an applications package written within the commercial database management system RDM, Responsive Data Manager, produced by Interactive Technologies, Incorporated.

RDM provides the system framework in a series of PASCAL programs which perform file manipulation and data input and reporting functions. In developing a custom application, like SIPS, the programmer uses RDM's programming applications package to define data files, create data input screens, write reports and set up menus.

SIPS contains the course schedule information for each quarter of the academic year, the catalog of courses which may be offered, the description of rooms and facilities available, and other data necessary for printing a variety of course related reports. SIPS can also look up information regarding the faculty, stored in the School Academic Personnel System, SAPS.

## COURSE SCHEDULING

In developing a database system, it is important to understand and keep in mind the requirements of the task you are trying to facilitate. In order to schedule courses, it is necessary to decide which courses should be offered, have faculty to teach them, and arrange facilities in which to hold them.

### Facilities Scheduling

SIPS helps in all areas of course scheduling, but the best example of what it can do is found in facilities scheduling. Making sure that 800 course sections fit, without conflicts, into 65 lecture and laboratory rooms can be a mind boggling proposition. Facilities organization is especially difficult in the School of Engineering, where laboratory work is an important part of the program. Many courses must be held in a particular laboratory because of equipment requirements. Lecture rooms are more interchangeable than labs, but they are in limited supply and vary in size. Without a systematic approach, and even with one sometimes, facility scheduling is enough to give the scheduler fits.

In an effort to break down the problem, rooms are initially assigned to each academic department. The department scheduler attempts to put courses into the department's rooms. Most departments, however, are not able to fit all of their courses into their own rooms. After the department's original attempt, all rooms in the School belong to the school scheduler. The school scheduler attempts to fit all courses without rooms into the school's rooms. Finally, the university scheduler will help the school scheduler find rooms belonging to other schools for any courses which cannot be fit into the school's rooms.

Prior to SIPS, the school scheduler wrote all the courses being offered in each room into a time chart for that room. After determining which course sections still needed rooms, by looking through an inch high stack of schedule request forms, the scheduler would attempt to find empty slots on the room charts for these courses. To do this job efficiently, the school scheduler needs to know how large a particular room is, and whether the set-up of the room is suitable for the course in question. Sometimes it is necessary to juggle several courses in order to fit in as many as possible, which means dealing with several rooms at once.

Why Computerize?

This work had been performed without the aid of computers for years. However, with the availability of computer technology, certain drawbacks in the system became more apparent. Much of the work being done by hand could be done more quickly, easily and accurately on a computer. For example, filling out room charts by hand seemed like wasted time when a computer could fill them out for us. The computer could also generate the list of courses still needing classrooms, saving a lot of time spent searching through forms. Also, course scheduling, room scheduling, and faculty hiring were all performed as separate functions. It was impossible to coordinate these activities.

From our experience with SAPS, our employee information system, we knew that the development of a database system for course scheduling would provide some immediate benefits, and inherent liabilities. Using a database system would help us gather and coordinate information related to the course schedule. Developing our own system would allow us to produce reports in formats convenient for the work performed in our office, adaptable to changing needs. However, the information needed to accomplish these tasks does not leap into the computer on its own, unfortunately. Someone has to enter and maintain the required data. New methods of processing the course schedule paperwork would also have to be developed in order to put the computer into the work flow.

Our success with SAPS encouraged us to take the next logical step and tackle the course schedule.

SCHOOL INSTRUCTIONAL PLANNING SYSTEM

Planning the System

The primary objectives in planning SIPS were that the system be useful, that it be useable by the staff who perform course scheduling tasks, and that it be ready almost immediately.

In planning any database system the most critical item is the data structure. It is important to know what items of information will be needed for a particular project and how these items relate to each other. If all of the necessary information is gathered together and organized efficiently to begin with, then it is easy to produce the procedures for inputing and reporting this information.

Unfortunately, it is almost impossible to think, from scratch, of each and every data item needed for a large project. Some organized approach is required or all sorts of things will be overlooked. In planning the data structure for SIPS, we started by looking at all of the course related paperwork we could find in our office: reports from the University, handwritten room charts, university course scheduling forms. These reports contained the information we wanted our database system to be able to handle, so we used them to determine the data items we needed in our system.

In designing our data structure, we were also able to take advantage of clues provided by the way the University does course scheduling. For example, they have a catalog file, separate from their course file, which contains information common to all sections of a course. There was good reason to believe that we would also probably want a catalog file in addition to our course file.

Database Fundamentals

In RDM, data items are stored in fields. A group of fields directly related to one another, which together describe a whole item, is called a record. Records are grouped in files. For example, for a particular course, the course code, starting time, ending time, meeting days, and instructor would all be separate fields. Together they make up one record. Each course being offered would have its own record, and the whole group of records make up the schedule file.

Different fields can be defined to contain different types of data. The course code is a number without a decimal and would be put in a numeric, integer, field. Time is another data type, and the instructors name would be placed in an alphanumeric "string" field. Other possible data types in RDM include dates, dollars, boolean (yes/no) and real numbers.

Data is entered into a file using an input form displayed on the terminal screen. Input forms are defined in a form control table. This table contains the data field name and the location on the screen to display or receive data. Additional headers and other special items may also be defined. The user never sees the form control table itself, only the forms on the screen into which data is typed. Menu screens are defined and function in a similar fashion. The command tables in RDM are accessed by the programmer in the same way that the applications data files are accessed by the user. Examples of input forms and menus are provided in Appendix I.

Putting data into the computer doesn't do any good if you can't get it out again. Data is usually output in some sort of report. A group of data items is printed, either on paper or to the terminal. In RDM, the specifications for reports are defined through an internal programming language. The programmer determines which data files will be used in the report, the sort order for the data, what information will be printed, and in what format. Almost any final report format is possible using RDM.

Work After Database

Once we planned and programmed a database system for course scheduling, we needed to figure out how to use it. The first concern was getting and maintaining the data. Incorrect or outdated information can be worse than no information at all. Data input was added to the school scheduler's tasks. After receiving schedule request forms from the academic departments, the school scheduler now types the information into the database, and prints

out new copies for the university scheduler, instead of xeroxing the department's forms and forwarding them. This task definitely adds extra time at this stage of the process, but the benefits make it worthwhile.

Focusing again on facilities scheduling, the school scheduler is able to print room charts for all the rooms in the School, rather than filling them out by hand. Time is saved here. A room specifications file was developed which includes the information the scheduler needs to know about each room: capacity, type of room, equipment available, and other useful items. The most critical parts of this information are printed right on the room chart for the room, the rest can be looked up in a facilities book which is also produced from the database.

Instead of sifting through several inches of paper to determine those courses without rooms, the school scheduler can now print a list directly from the course schedule, selecting only those courses with rooms 99-999, the code for "no room". Using this list, the scheduler can look for empty slots on the room charts and juggle courses more easily than in the past. Generally, courses are written into the charts by hand for awhile, but whenever they become too messy to be comfortable, the scheduler can print a new set, containing all the latest information. If a course is cancelled, the scheduler cannot forget to remove it from the room chart, the course will disappear from the room automatically.

Similar results are obtained in other areas of course scheduling. Many changes occur to the actual course schedule after it is published by the University. Courses are added, and cancelled; course times, instructors or rooms are changed. Prior to SIPS, lists were kept by the academic departments of changes to the schedule. Now, lists can be prepared for the entire School showing the most current information, and also showing those changes that have taken place since the published course schedule.

By creating a file to contain the office hours of the faculty, we were also able to produce time schedules for each faculty member in the School. These schedules show their course information, office hours, and significant meetings which they attend. They can be posted outside their offices for students and other faculty trying to locate them. Such schedules were previously prepared by the academic departments, but since we had all of the course schedule information on-line in SIPS, it was reasonable to print schedules for the entire School at one time. We have also developed a faculty chart, similar to the room charts. These charts may be superimposed, so that if you wanted to find a common time for a meeting between four faculty members, you could look at all four of their charts simultaneously on a light table and find any empty slots which they have in common.

Another report never possible before SIPS is the final exam schedule for courses. The University determines that, for example, all classes meeting Monday, Wednesday, and Friday at 7:30am will have their final exam on Wednesday of finals week at 7:00am. SIPS contains the university final exam schedule and can look up the final exam time for each course in the School. This schedule is then printed and posted for students during final exam week, and answers quite a few questions from students, who are always running around trying to find out when their final exams are. This list also includes any changes to the final exam time made by individual instructors for particular courses, which cannot be included in the generalized schedule produced by the University.

Several examples of SIPS reports are found in Appendix II.

Benefits in Work with SIPS

The most significant benefit incurred in working with SIPS is consistency. Information regarding a particular course will be the same whether you are looking at a room chart, the course schedule, a faculty members schedule, or the final exam schedule. If the information is entered into the database correctly, it will remain correct, and it will be correct everywhere it appears. If the information is, by chance, entered incorrectly, it will be wrong everywhere, and perhaps easier to catch and correct.

Another benefit which is nice, although not as significant, is neatness. Typed room charts are more pleasant to read than hand written ones. And the university scheduler seems to prefer checking fifty pages of typed course request forms to checking fifty pages hand written by a variety of people in all colors of ink or pencil.

Finally, using SIPS, we are able to do some things which were never possible before. It is now possible to check and make sure that every faculty member who was hired for a particular quarter was also given courses to teach, and that every faculty member who is teaching courses was actually hired. There is a rule that full time faculty must have five office hours on four days during the week. Using SIPS, it is simple to add up the office hours for each faculty member, check the number of days and print a list of any faculty who are deficient in either. What used to take a clerical staff person hours can be done by the computer in minutes. These and other auditing functions are a significant benefit of using a database system.

APPENDIX I - INPUT FORMS AND MENUS

```
RDM 3.1G                                                          ** RSX **
┌──────────────── SCHOOL INSTRUCTIONAL PLANNING SYSTEM ─────────────────┐
│              *               SIPS                     *                │
│              *        MISCELLANEOUS REPORTS           *                │
│      >                                                                 │
│          SCHEDULE    Dump Report of the Schedule Data                  │
│                                                                        │
│          ROOMS       Room Specifications - Sorted by Room Number       │
│          LOCATION    *                   - Sorted by Room Location     │
│          OWNER       *                   - Sorted by Room Owner        │
│          SUMMARY     Room Assignment Summary Report                    │
│          USE         Room Utilization Report                           │
│                                                                        │
│          PHONES      Phone Auditing Report                             │
│          KEYS        Room Key Report                                   │
│                                                                        │
│          RETURN      Return to the SIPS Main Menu                      │
│                                                                        │
└──── Serial #IX3S000025 Copyright 1984 Interactive Technology, Inc. ───┘
                                                            Cal Poly Univ
```

Menu Screen Sample

COURSE SCHEDULE INFORMATION

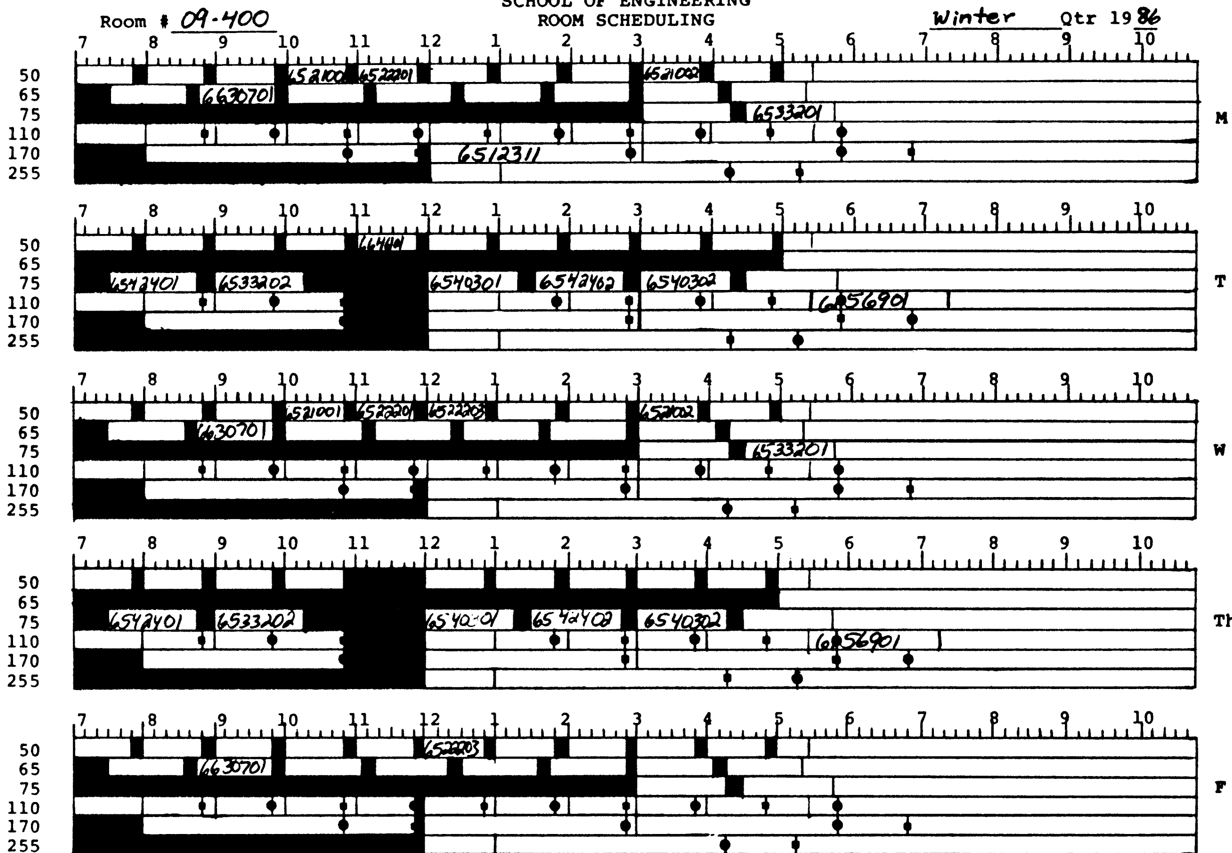
```
Revised  24-APR-86     Quarter  8671

CourseCode  61 544 01  Units   4.0_     Seats  20_   Enrollment  16_
  Continue  0            CS    C05

  AddLines  0          Combined  N    GroupCode 1  Status  _

  M  T  W  T  F  S  S                 Begin         End
  Y  N  Y  N  N  N  N                 05:30 pm     07:20 pm

  Location  09_ 521_

  FI  RJH_                            TeamTeaching%  100

  Notes  ______________________________           Footnote  0_
```

Data Input Screen Sample

# APPENDIX II - SIPS REPORTS

```
                    SCHOOL OF ENGINEERING - CALIFORNIA STATE POLYTECHNIC UNIVERSITY, POMONA
                                            ROOM USAGE CHART
                                         For AYQuurter: 8672
16-OCT-86                                                                                            17:36:08

OCATION: 09-400   Lecture Room
unction: LEC  Capacity: 42    Extension:              Notes: Sinks

Assignment: Engineering   Civil Engineering                       Days:            From:        To:

MONDAY                                                                                                  MONDAY
7---:---8---:---9---:---10--:---11--:---12--:---1---:---2---:---3---:---4---:---5---:---6---:---7---:---8---:---9---:---10
              XXXXXXXXX XXXXXXX XXXXXXX XXXXXXXXXXXXXXXXXXXXXX XXXXXXX     XXXXXXXXXX
              6630701   6521001 6522201 6512311                6521002     6533201
              Ferguso   Carlyle Clark   Coduto                 Carlyle     Janger

TUESDAY                                                                                                TUESDAY
7---:---8---:---9---:---10--:---11--:---12--:---1---:---2---:---3---:---4---:---5---:---6---:---7---:---8---:---9---:---10
    XXXXXXXXXX  XXXXXXXXXX      XXXXXXX XXXXXXXXXX  XXXXXXXXXX XXXXXXXXXX          XXXXXXXXXXXXXXX
    6542401     6533202         6646101 6540301     6542402    6540302             6156901
    Schneid     Wells           Staff   Borowic     Coduto     Borowic             Staff

WEDNESDAY                                                                                            WEDNESDAY
7---:---8---:---9---:---10--:---11--:---12--:---1---:---2---:---3---:---4---:---5---:---6---:---7---:---8---:---9---:---10
              XXXXXXXXX XXXXXXX XXXXXXX XXXXXXX                XXXXXXX     XXXXXXXXXX
              6630701   6521001 6522201 6522203                6521002     6533201
              Ferguso   Carlyle Clark   Izadi                  Carlyle     Janger

HURSDAY                                                                                               THURSDAY
7---:---8---:---9---:---10--:---11--:---12--:---1---:---2---:---3---:---4---:---5---:---6---:---7---:---8---:---9---:---10
    XXXXXXXXXX  XXXXXXXXXX              XXXXXXXXXX  XXXXXXXXXX XXXXXXXXXX          XXXXXXXXXXXXXXX
    6542401     6533202                 6540301     6542402    6540302             6156901
    Schneid     Wells                   Borowic     Coduto     Borowic             Staff

FRIDAY                                                                                                  FRIDAY
7---:---8---:---9---:---10--:---11--:---12--:---1---:---2---:---3---:---4---:---5---:---6---:---7---:---8---:---9---:---10
              XXXXXXXXX                 XXXXXXX
              6630701                   6522203
              Ferguso                   Izadi
```

SIPS Room Chart




Pre-SIPS Room Chart

```
Revised: 12-OCT-86                                                     8671 ACS T
                       CALIFORNIA STATE POLYTECHNIC UNIVERSITY, POMONA
                                  SCHOOL OF ENGINEERING
                                     Fall Final Copy
     Name: Stoll, A. George                                              Office: 13-228
           Professor                                                     Ext:    2626
     Dept: CHE

                                  - - CLASS SCHEDULE - -

     6147006  COOP ED CHE                  By Appt.              98-000
     6147106  COOP ED CHE                  By Appt.              98-000
     6147206  COOP ED CHE                  By Appt.              98-000
     6147306  COOP ED CHE                  By Appt.              98-000
     6342511  UNIT OP I LB        T        03:00 - 05:50 pm      11-135
     6342512  UNIT OP I LB        W        12:00 - 02:50 pm      11-131
     6346103  SR PROJECT                   12:00 - 02:50 pm      98-000
     6346208  SR PROJECT          M        12:00 - 02:50 pm      98-000

                             - - OFFICE HOURS/MEETINGS - -

                                  Th       01:30 - 03:30 pm      13-228
                                  MTW      11:00 - 11:50 am      13-228

     Schedule Coordinator                  -                     -

     SEAC Meeting                 T        09:00 - 11:00 am      09-510
     CAM Committee                         -                     -
     Energy Engineering Minor              -                     -
```

```
MONDAY                                                                                                    MONDAY
7---:---8---:---9---:---10--:---11--:---12--:---1---:---2---:---3---:---4---:---5---:---6---:---7---:---8---:---9---:---10
                                XXXXXXX
                                Office
                                13-228
TUESDAY                                                                                                   TUESDAY
7---:---8---:---9---:---10--:---11--:---12--:---1---:---2---:---3---:---4---:---5---:---6---:---7---:---8---:---9---:---10
                  XXXXXXXXXXXXXXXXXXXXXXX                       XXXXXXXXXXXXXXXXXXXXXXX
                  Meeting       Office                          6342511
                  09-510        13-228                          11-135
WEDNESDAY                                                                                                 WEDNESDAY
7---:---8---:---9---:---10--:---11--:---12--:---1---:---2---:---3---:---4---:---5---:---6---:---7---:---8---:---9---:---10
                                XXXXXXX XXXXXXXXXXXXXXXXXXXXXXX
                                Office  6342512
                                13-228  11-131
THURSDAY                                                                                                  THURSDAY
7---:---8---:---9---:---10--:---11--:---12--:---1---:---2---:---3---:---4---:---5---:---6---:---7---:---8---:---9---:---10
                                                    XXXXXXXXXXXXXXX
                                                    Office
                                                    13-228
FRIDAY                                                                                                    FRIDAY
7---:---8---:---9---:---10--:---11--:---12--:---1---:---2---:---3---:---4---:---5---:---6---:---7---:---8---:---9---:---10
```

Faculty Schedule Report

CALIFORNIA STATE POLYTECHNIC UNIVERSITY, POMONA - CLASS SCHEDULE FORM

DEPARTMENT Aerospace Engineering
QUARTER WINTER 1987

PAGE 7
16-OCT-86 ITEMS

| ACTION | COURSE CODE | LINE CONT | M T W T F S S | NIGHT CLASS | HOURS BEGIN | END | CLASS UNITS | BLDG | ROOM | AVAILABLE SEATS | TEAM TEACH% | FOOT NOTE | COMB. |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| DEL 3 | 6235511 | | | | | | | | | | | | |

INSTRUCTOR'S NAME SOCIAL SECURITY

| ACTION | COURSE CODE | LINE CONT | M T W T F S S | NIGHT CLASS | HOURS BEGIN | END | CLASS UNITS | BLDG | ROOM | AVAILABLE SEATS | TEAM TEACH% | FOOT NOTE | COMB. |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ADD 1 | 6240101 | | 2 4 | 1 | 05:00 | 06:50 | 04.0 | 13 | 215 | 028 | | | |

INSTRUCTOR'S NAME Newberry CF 515-26-0904 SOCIAL SECURITY

| ACTION | COURSE CODE | LINE CONT | M T W T F S S | NIGHT CLASS | HOURS BEGIN | END | CLASS UNITS | BLDG | ROOM | AVAILABLE SEATS | TEAM TEACH% | FOOT NOTE | COMB. |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CHG 2 | 6240901 | | | | | | | 09 | 251 | | | | |

INSTRUCTOR'S NAME SOCIAL SECURITY

| ACTION | COURSE CODE | LINE CONT | M T W T F S S | NIGHT CLASS | HOURS BEGIN | END | CLASS UNITS | BLDG | ROOM | AVAILABLE SEATS | TEAM TEACH% | FOOT NOTE | COMB. |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CHG 2 | 6241001 | | | | | | | 09 | 245 | | | | |

INSTRUCTOR'S NAME Staff 000-00-0001 SOCIAL SECURITY

Course Request Form Report

# GRAPHICS APPLICATIONS SIG

# What is PostScript?

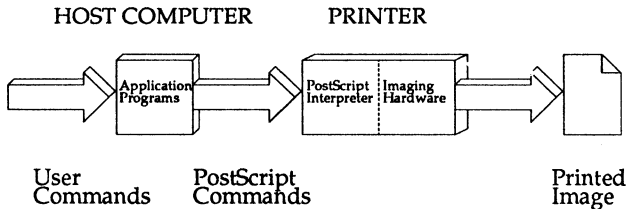
Ann Robinson
Adobe Systems Incorporated
Palo Alto, California

## Abstract

PostScript® is becoming widely acknowledged as an industry standard for printing. Digital Equipment Corporation, among others, has adopted PostScript in its low and medium range printers. This talk discusses PostScript and why it is a key ingredient in printing and publishing.

*What is PostScript?* PostScript is a powerful programming language for describing the appearance of pages, including pages that combine text, graphics, and scanned images. Essentially any page can be specified with a PostScript program.

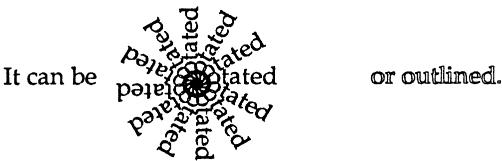
*Where is PostScript?* The interpreter for PostScript resides in the controller of printers and typesetters. An application program on a host computer generates PostScript commands describing each page to be printed. Those commands are sent to the printer where they are interpreted. The interpreter causes the page to be printed.




Commands in PostScript are independent of the resolution of the printer to which they are sent. The same file can be printed on a PostScript-equipped laser printer with a resolution of 300 dots per inch or on a typesetter with a resolution of 2540 dots per inch. The application program does not need to know the resolution of the PostScript device.

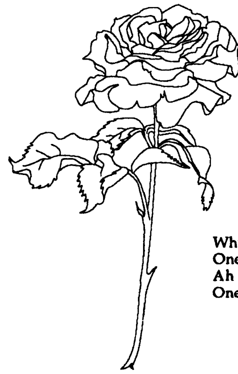
Text in PostScript is treated like graphic images: characters can be scaled, rotated, stretched, outlined, and filled with black or shades of gray. PostScript typefaces have been licensed from Mergenthaler and International Typeface Corporation, two major type design companies.

Text can be very small or very large.




And printed in shades of gray.

PostScript contains a range of graphic primitives including line, arc, and area filling commands. These commands allow very complex graphics to be described and printed.

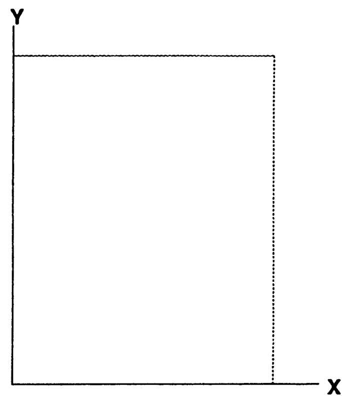


Why is it no one ever sent me yet
One perfect limousine, do you suppose?
Ah no, it's always just my luck to get
One perfect rose.
*Dorothy Parker*

In addition to text and graphics, PostScript's halftone mechanisms permit full control over halftone screens. Spot angle, size, and shape can all be specified.

*What does a PostScript program look like?* PostScript is a full high-level programming language with a Forth-like syntax. The basic imaging model of PostScript is that of writing on a page with opaque ink: every successive mark covers the one below.

In specifying a page, the origin is always in the lower left corner of the page. Units are in printer's points (1/72").



To write a line of text, first the proper typeface must be located, scaled, and set as the current font. In this case, we are selecting Times-Roman at 40 points.

```
/Times-Roman findfont
40 scalefont
setfont
```

Then the current point is positioned where the text is to begin. Here it is one inch from left edge and one inch from the bottom edge of the page.

```
72 72 moveto
```

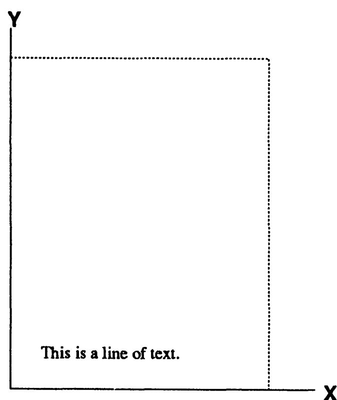
Next, the text is put onto the page

```
(This is a line of text.)show
```

And the page printed.

```
showpage
```

This sequence produces the page:




The text starts one inch from the left and one inch from the bottom of the page.

Similarly, lines are drawn by moving to a point

```
72 72 moveto
```

drawing the path for the lines,

```
72 144 lineto
144 144 lineto
144 72 lineto
```

closing the path,

```
closepath
```

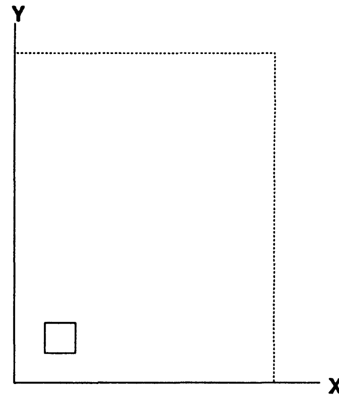
stroking the path,

```
stroke
```

and printing the page.

```
showpage
```

This program produces a box one inch square, with its left edge one inch from the left edge of the page, and its bottom edge one inch from the bottom of the page.




PostScript commands also include arc, fill, scale (to change the units of the coordinate system), translate (to move the default origin), rotate (to rotate the origin), and programming constructs suchs as for and repeat.

Although to date, no color printers have been announced, PostScript also has a full color model. Colors can be specified in terms of red, green, blue, and hue, saturation, brightness.

Future developments for PostScript are expected to include higher performance printers, color printers, and PostScript on other raster devices, such as displays.

PostScript is a registered trademark of Adobe Systems Incorporated.

# HALFTONE - A PROGRAM FOR CONVERTING GREY-SCALE IMAGES TO HALFTONES

Robert B. Goldstein, Eli Peli*
Eye Research Institute of Retina Foundation
20 Staniford St
Boston, MA 02114

Karl Wooledge
Image Analysis Laboratory
Tufts-New England Medical Center
Boston, MA

## ABSTRACT

A program has been written that converts grey-scale images to halftones. This enables the images to be displayed on devices that normally do not have grey-scale capability.

Target devices for output include the VT100, VT240, LA50, LA120 and LP26. Knowledge of the aspect ratio of those devices is built into the program so that the image is not distorted when displayed. The sixel capability of the VT240 and LA50 is used.

The algorithm is based on an error-propagation method published by Saghri, Hou and Tescher. We improved the algorithm by including a factor that takes into account the difference in size between printed and non-printed points. Also, the execution time is improved by eliminating interpolation in most cases.

## INTRODUCTION

A program has been written that converts grey-scale images to halftones. Figure 1 shows the results as printed on an LA50 printer.

Figure 1. Halftoned GIRL on LA50

This program was written in the search for cheap, 'notebook-quality' output from our image analysis systems. In addition, we are studying the enhancement of images to improve the printing process, and halftoning is part of that process. The algorithm of Saghri, Hou and Tescher[1] was taken as a starting point.

The program has the capability of preparing the image for output onto many devices. Figure 2 shows output from a VT240 (a sixel device having 800x250 cells) and Figure 3 shows output from an LA120 (130x66). As can be seen, the fewer cells available for grey-scale simulation, the cruder the appearance of the output picture.

Figure 2. Halftoned GIRL on a VT240.

*Eli Peli is also with Tufts-New England Medical Center

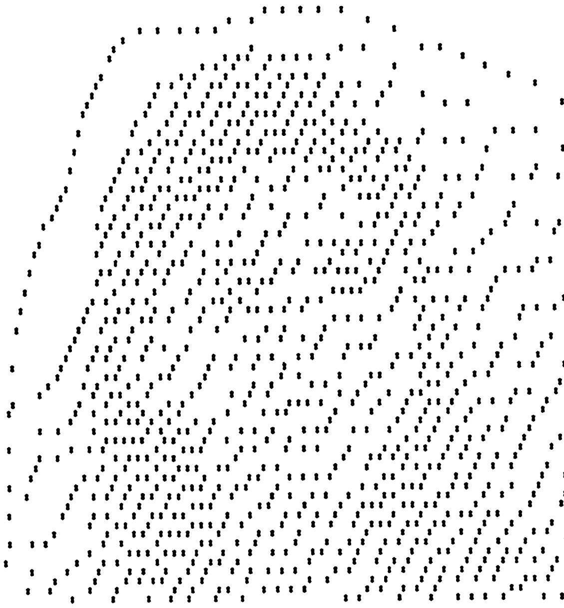


Figure 3.Halftoned GIRL on an LA120.

### TECHNICAL DETAILS

A flowchart is shown in Figure 4. Logically the process is divided into 2 tasks: a)choosing and pre-processing of the image for the 'target device', and b)performing the halftoning. The original image is expanded or compressed to fit into the number of 'cells' available on the target device. Knowledge of the aspect ratios of the pixels on the target devices is built into the program (for example, the LA50 is 2:1) and this is taken into account in the preprocessing step. The details of the preprocessing are adequately described in Saghri, Hou and Tescher[1].

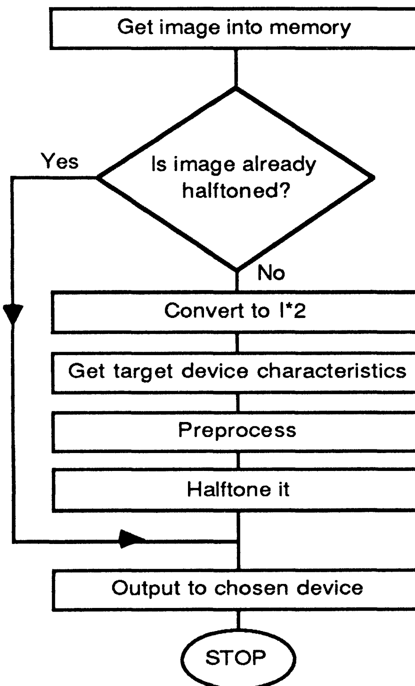



Figure 4. Flowchart of Halftone program.

When an expansion of the image is required, the Saghri, Hou and Tescher[1] algorithm interpolates to fill in the 'missing' pixels. We have speeded up the process by NOT interpolating, but replicating the bounding pixels. This works because the second stage, the halftoning process, is a form of interpolation and adequately 'fills in' the missing pixels. The expanded image has the dimensions of the target device and is now ready for halftoning.

The halftoning process consists of assigning each cell to be a zero or one. Assigning a zero generates an 'error' equal to the actual grey-scale value. Equivalently, assigning a one generates an 'error' equal to the difference between the maximum and actual grey-scale values. When a cell is assigned a value, the errors introduced due to assignments of previous pixels are taken into account. Following the methodology and notation of Saghri, Hou and Tescher[1], we define $E_g(m,n)$ as the total error generated, I(m,n) as the grey-scale value, and H(m,n) as the halftone assignment at position (m,n). Then to calculate the error propagated to point (m,n), we have (equation 10 of Saghri, Hou and Tescher[1] )

$$E_p(m,n) = \sum_{i=1}^{I}\sum_{j=1}^{J} C_{ij} E_g[(m-i+1),(n-j+1)]$$

where

$$\sum_i \sum_j C_{ij} = 1 \quad \text{and} \quad C_{11} = 0$$

The C matrix is the error distribution function and serves to give less weight to positions further from (m,n). As illustrated in Figure 5, this calculates the errors contributed by the upper left neighboring pixels.

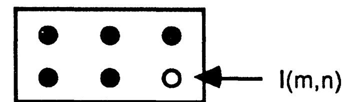



Figure 5. Pixels that contribute to error at (m,n)

H(m,n) is assigned in the following way:

$$H(m,n) = \begin{cases} 1 \text{ if } I(m,n) + E_p(m,n) > \text{thresh} \\ 0 \text{ if } I(m,n) + E_p(m,n) < \text{thresh} \end{cases}$$

$E_g(m,n)$ is then calculated:

$$E_g(m,n) = \begin{cases} I(m,n) + E_p(m,n) + K_{fact} & \text{if } H(m,n) = 0 \\ I(m,n) + E_p(m,n) - \text{maxval} & \text{if } H(m,n) = 1 \end{cases}$$

The difference from Saghri, Hou and Tescher 's[1] formulation is in the addition of $K_{fact}$ . On the printed page, when a black dot is printed, it spreads ink over surrounding

cells so a black pixel is larger than a white pixel. This introduces additional errors. The $K_{fact}$ is introduced into the above equation to account for these additional errors. It is determined empirically for each device. Presently $K_{fact}$ ranges from 0 to 1024.

### TUNING

In our experiments with the program we varied the C matrix and observed the effect on CPU time. Reducing the C matrix from 2x3 to 2x2 reduced the CPU time from182 to 136 seconds. It degraded the resulting image somewhat but the quality was still adequate for some purposes. Replacing a matrix multiplication subroutine with in-line code further reduced the CPU time to 57 seconds. These timings were for a 500x1000 output image where the target device was the LA50. The CPU was a dedicated microVAX II.

### CLINICAL USE

At the Tufts-New England Medical Center in Boston, this program has been used in a clinical setting. Muscle biopsy slides are received for image analysis. Measurements of two types of cells (light and dark) found in these biopsies are performed. Examples of the types of parameters reported are number of cells, average diameters, average areas and descriptors.of clustering and neighborhood relationships. The physician receives the report on an LA50 printer in his office. The halftoned image (Figure 6) is provided as part of the report to provide a meaningful reference for the accompanying statistics. Cell boundaries and the two types of cells are easily distinguishable in this image.

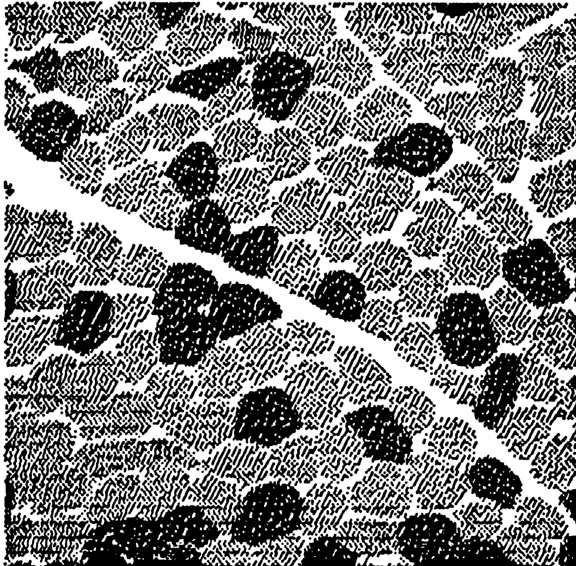


Figure 6. Halftoned Muscle Biopsy

### CONCLUSION

Additional work that will be done on this program includes:

1. Adapt the algorithm so that if more than one bitplane is available, the program would use it. This would allow us to use the two bitplanes on the VT240.
2. Use overprinting to simulate grey-scale on an alpha printer.
3. Do additional tuning. For example, some operations can be done in byte arithmetic rather than integer arithmetic.
4. Add an image enhancement step to pre-process the image before halftoning. Preliminary experiments using adaptive enhancement[2] look very promising.

This program has been put on a DECUS tape. It provides an easy-to-use, "entry" level way for users to get hard-copy outputs from images.

### REFERENCES

(1) Saghri, J.A., Hou, H.S., Tescher, A.G. "Personal Computer Based Image Processing With Halftoning", Optical Engineering 25:499-504, 1986.

(2) Peli, T. and Lim, J.S., "Adaptive Filtering for Image Enhancement", Optical Engineering 21:108-112, 1982.

### ACKNOWLEDGEMENTS

This work was partially supported by NEI grant EY05957 and a grant from the Alcoa Foundation.

# A GRAPHICS EDITOR FOR 3-D CT-SCAN DATA FOR MUSCULO-SKELETAL MODELING

L.M. Myers (1), W.L. Buford, Jr. (1), and D.E. Thompson (2)

1 - Rehabilitation Research Dept., Gillis W. Long Natl Hansens Disease Center, Carville, LA 70721
2 - Department of Mechanical Engineering, Louisiana State University, Baton Rouge, LA 70808

## ABSTRACT

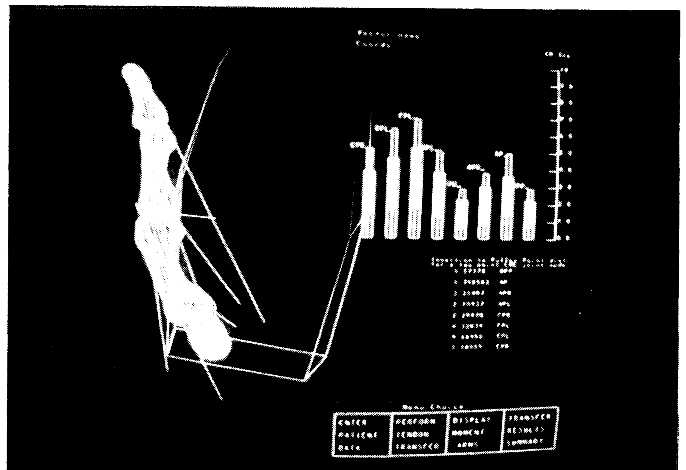
A real time interactive graphics system has been developed for the creation of anatomically correct kinematic models of the hand from CT scan data. The system reconstructs three dimensional skeletal boundaries from planar CT scans of a hand specimen, segments the 3D boundaries into individual skeletal units, and interactively builds a kinematic hierarchy defining Euclidean transformations which are applied to each segment. A concurrent processing arrangement is used to ease the computing burden. This consists of a VAX 11/750 connected to an Evans and Sutherland PS 330 high resolution, high speed 3-D graphics display system via a DMA interface.

With the system, the user manipulates a three dimensional structure created by the boundary reconstruction algorithm and identifies and creates new structures named for each skeletal segment. The individual segments are manipulated into proper anatomical positions and iteratively rotated and translated, thereby defining transformation nodes in a kinematic hierarchy. The resultant data structure and kinematic hierarchy is the basis for a working three dimensional simulation of the hand. This type of computer modeling, utilizing realistic imaging, musculoskeletal models, and interactive programming shows great potential for bringing biomechanics models into useful clinical, research, and educational applications.

## INTRODUCTION

Hansen's disease patients, as a result of peripheral nerve damage, develop deformities of the hands which are routinely treated by reconstructive surgery. For such surgery to be effective, a clear understanding of the mechanics of the hand is essential. Unfortunately, most surgeons do not possess the engineering background necessary to grasp what is already known about this subject (5,6,7,8,9) in its current form.

Work in our lab (9,10,11,12,13,14) is directed towards learning more about the biomechanics of the hand, and making that information available to clinicians in a form which they can understand. Represented in mathematical terms or even cleverly defined graphical plots, the analytical results of a simulated tendon transfer go unused in the environment in which they are



-Figure 1. Graphics display depicting movement of the human thumb during manipulation and analysis with the hand simulation of Buford(13).

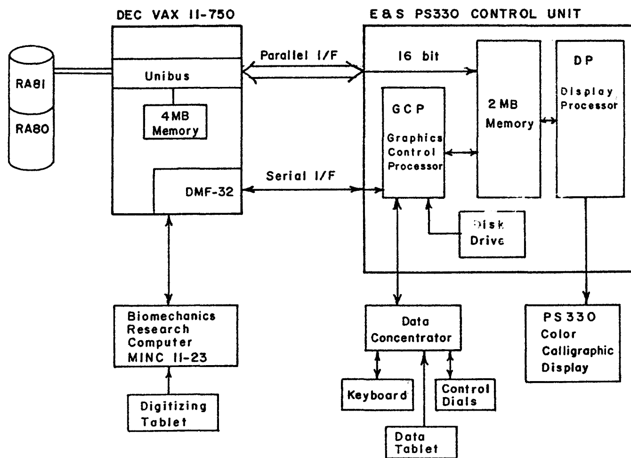
most needed - the orthopedic clinic. However, when incorporated into a 3D computer graphics model with which the clinician interacts in real time, the same information is readily assimilated (Figures 1,2).

Essential to the realism and clinical acceptance of such a model is an accurate 3D representation of the bones of the hand. This report describes the development of a system which uses CT scans of a cadaver specimen as an initial source from which skeletal surfaces are detected and individaul bones derived.

-Figure 2. The PS330 system running the hand simulation

## DEVELOPMENT ENVIRONMENT

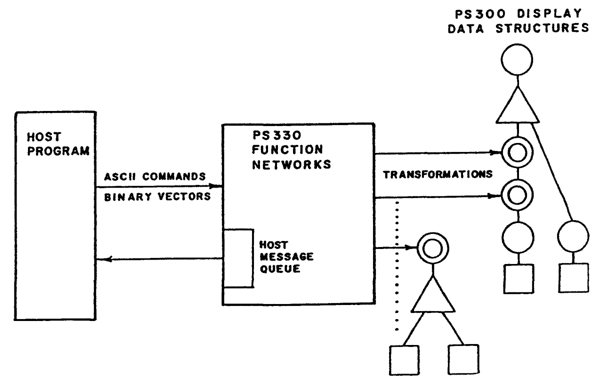
The hardware development system depicted in Figure 3. runs the Interactive Graphics Editor as well as the previously mentioned hand simulation. A VAX 11-750 (host) and an Evans and Sutherland PS330 (display) share the computational load. The




-Figure 3. Hand simulation development system.

PS330 is a high performance color calligraphic system which handles display transformations (scaling, rotation, translation, and viewing transforms), data structure interactions, and interactive device I/O. The VAX is assigned the responsibility for mass storage of raw and edited data, 2-D edge detection, and generation of the initial 3D boundary vectors. In the hand simulation, muscle-tendon kinematics are solved on the VAX, updating display variables on the PS330 as required for the simulation. Communication between the host and the display is via an RS232 serial line at 9600 baud and a parallel interface at 1 Mbytes/sec. By sharing the computational load, the interactive nature of the simulation is enhanced.

The software environment depicted in Figure 4 represents three modes of program activity. FORTRAN programs running on the




-Figure 4. The software programming environment.

host send PS330 application programs in character or binary form to the PS330 Graphics Control Processor where they are parsed and set up in the display memory. The FORTRAN program then waits for input signals from the display programs which determine subsequent computation action. The application display programs are derived from the PS330 display data structure language and function network language.

The display data structure language provides the means by which hierarchical PS330 display structures are built. Elements of this language define the transformation nodes required to position hand structures relative to one another and to dynamically alter them with respect to their axes of motion.

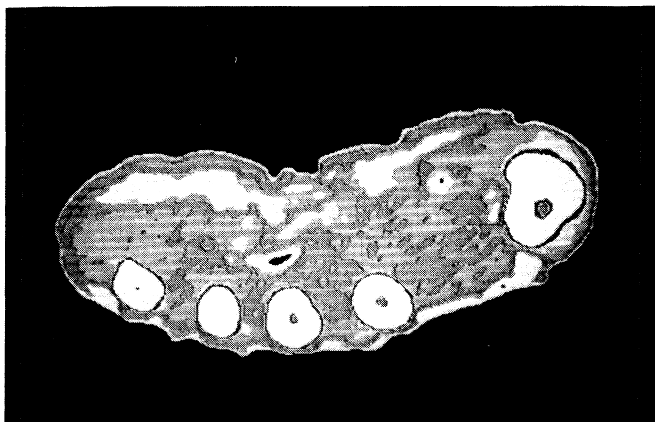
The function network language is composed of a large set of block structured functions which provide input, output, arithmetic, logic, manipulation, comparison, transformation, and conversion capabilities. Function networks define the connections between the interactive devices and the hand display data structures, perform

transformations and other computations to update interaction and anatomical elements in the display structure, and define the link between the display structure (PS330) and the host program (VAX).

## DATA ACQUISITION AND STORAGE

Computed Tomography is a radiological technique which generates a series of 2-dimensional cross-sectional images at specified increments along a third axis(Z) which is perpendicular to the plane(X,Y) of the cross-sections.

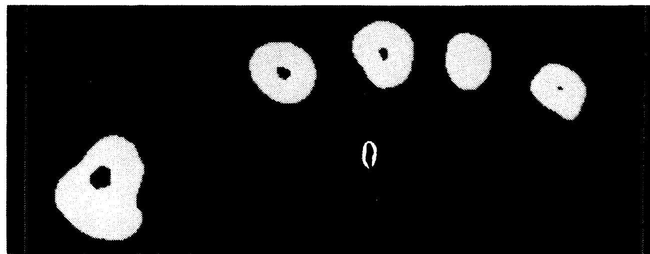
In this study, a cadaver hand specimen was mounted in wax on a specially designed fixture which would allow for the rotation of the specimen by 90 degrees in the Y axis. The fixture was then attatched to the bed of a Seimens Somatom DR3 CT scanner so that the long axis of the hand corresponded to the direction of bed travel (i.e. parallel to the Z axis). 172 slices were taken at 1mm increments of bed position, the fixture was rotated 90 degrees, and an additional 100 slices were taken in the YZ plane, also at 1mm increments. This was done to obtain additional resolution at joint surfaces in the resultant 3D data structure.



- Figure 5. Raster vector rendering of pixel data of a CT section of a right hand at mid metacarpal level.

Each individual slice is composed of a matrix of 512 x 512 pixels, each of which is represented by a 16-bit word. In addition, each slice contains 8 512 byte header blocks containing information such as the bed position, the X,Y reconstruction center, and the zoom factor for a given slice. This information is used by the 3D reconstruction algorithm to determine the Z coordinate, the X,Y offset of a given pixel from the physical CT scan center, and the physical size of each pixel (.29 mm square in our case) respectively.

Since the total amount of raw data was in excess of 140 megabytes, it became necessary to develop some sort of data compression technique. A run length encoding algorithm was written and applied to the data, resulting in a reduction of overall storage requirements by approximately a factor of 5. A side benefit of this storage reduction was an increase in processing speeds of up to an order of magnitude in certain instances due to the decreased data load times and system memory requirements.

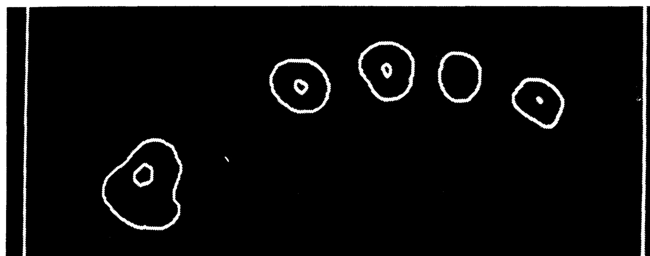


-Figure 6. Raster vector rendering of bones only at mid metacarpal level.

## SURFACE RECONSTRUCTION

Software was developed for viewing 2-dimensional slices which allows the user to interactively define up to 16 intensity ranges between 0 and 4095 (the limits of CT intensities). Each pixel intensity value is then sorted into one of these 16 ranges. Each resultant group of horizontal-adjacent pixels of the same intensity range is then replaced by a single horizontal vector segment to form an image such as the one seen in Figure 5. Using a threshold of 1200 CT intensity units, the image of hand bones in Figure 6 was generated. A curve-tracing algorithm is then applied to the set of horizontal segments corresponding to bone and a single 2D vector list is generated (Figure 7).

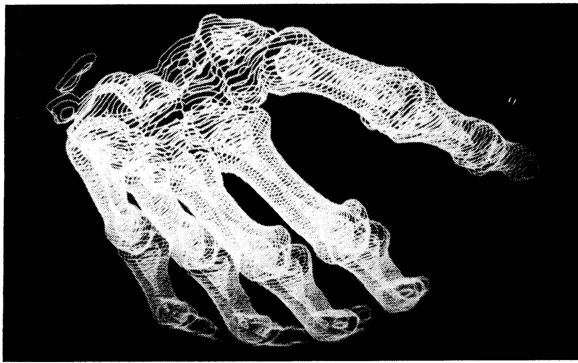
This processing, when applied over a range of CT slices, forms the basis for generation of the initial 3d sturcture. By obtaining the z coordinate for each 2-D slice from the Bed position at which that slice was made, the 3D vector list of the hand bones shown in Figure 8 is generated.



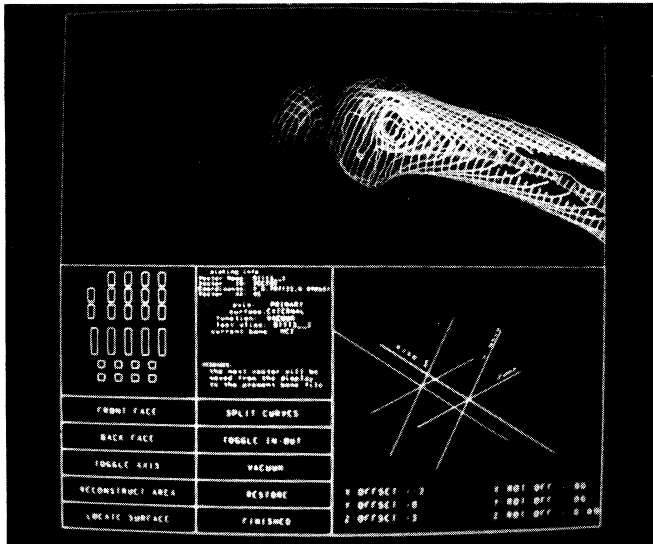
-Figure 7. Vector list representing the bone edges at mid metacarpal level.

## EDITING AND SEGMENTATION

The vectors representing the axial and longitudinal sets of orthogonal sections are combined into a single 3D structure which is manipulated to create the kinematic model composed of bone segment primitives. This



-Figure 8. Vector list representing the bony surface of the entire hand.



-Figure 9. Graphics display of the interactive segmentation tool. 3D display of the bony surface (top). Menu selection and feedback (lower left). Reference axes (lower right).

segmentation procedure uses host and graphic programming to interactively transform the combined scan data to separate boundary vector lists for each bone.

The 3D graphics editor has three operational modes of interaction:

1. When in the hand icon block, the graphics tablet allows the user to select icons which represent bone data files into which subsequently picked individual curve vector lists will be stored or removed.

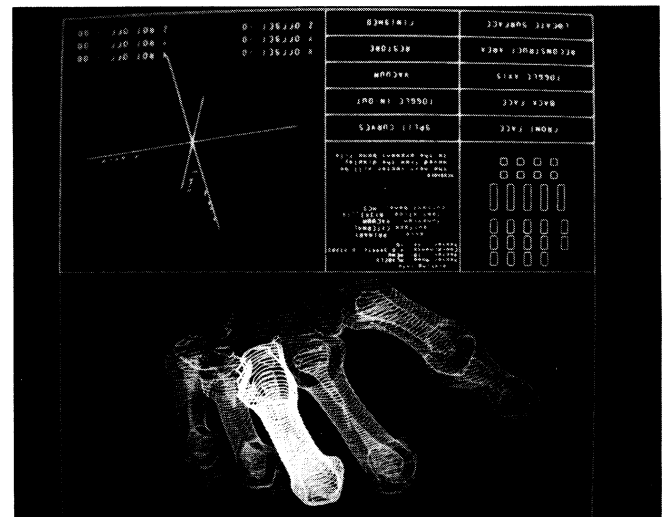
2. When in the menu block (lower left of display), the cursor is used to select editing operations which include:

   - storage of the next selected vector list to the current bone file
   - removal of the last selected vector list from the current bone file
   - interactive splitting of a single vector list into two sub-lists

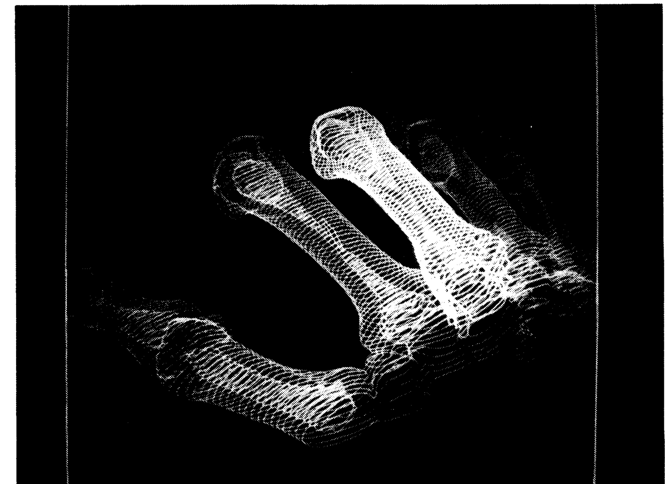
3. Alternatively, when the cursor is within the 3D viewing window, it is used to specify the individual curve vector to which the current menu selection operation will apply.

Color on the PS330 is used to great advantage in the editor software. The currently active bone data file icon is highlighted in red in the menu window. A change of color of the displayed curves is used to denote that they have already been picked and appended to the currently active data file. Red is used within a dialogue box to send error messages to the user when an abnormal condition has occurred.

Use of the interactive segmentation tool for the removal of a metacarpal bone is depicted in the sequence of Figures 9-12. At all times, the user may interactively rotate, translate, and scale the entire hand display using control dials.



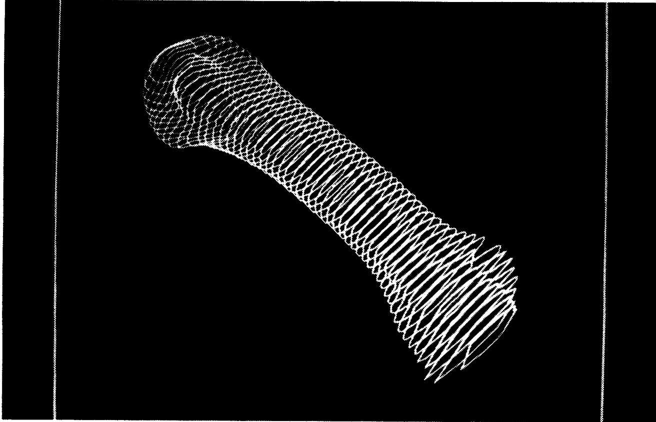
-Figure 10. Graphics display of segmentation during selection of surface vectors from the 3rd metacarpal surface.



-Figure 11. The display following identification of the third metacarpal.

## CONCLUSION

The same system used for interactive simulation of skeletal kinematics is used in this application to develop 3D bone segments from CT scan data. These segments then become the primitives for inclusion in the hand model and for later improvement in 3D bone and joint surface models. Data structure refinements, tissue deformation models, musculoskeletal dynamics, improvements in interaction, display realism, and the speed of the simulation will continue to be important factors in the eventual acceptance of a functional model for clinical and educational use. Meanwhile, the system reported here is proving to be an extremely powerful research and development biomechanics simulation tool.



-Figure 12. Isolation and manipulation of segmented 3rd metacarpal.

## ACKNOWLEDGEMENTS

The authors gratefully acknowledge the support of Digital Diagnostics, Inc. of Baton Rouge (Dr. Charles Grieson, Director) for their assistance. This research is supported by USPHS contract number 240-83-0060.

## REFERENCES

1. R. Williams and A. A. Seireg, "Interactive Computer Modeling of the Musculo-skeletal System," IEEE Transactions on Biomedical Engineering, Vol. 24, No. 3, pp. 213-218, May 1977.
2. P. Dev, "A Simulator for the Analysis of Wrist Position Control," Proceedings of the 1982 American Control Conference, Arlington, VA., June 14-16, pp. 1199-1204, 1982.
3. D. C. Hemmy, D. J. David, G. T. Herman, "Three - dimensional reconstruction of cranio - facial deformity using computed tomography," Neurosurgery, Vol. 13, no. 5, pp534-541, Nov 1983.
4. G. T. Herman, H. K. Liu, "Three - dimensional display of human organs from computed tomograms," Computer Graphics and Image Processing, Vol. 9, pp1-21, 1979.
5. R. Fick, Handbuch der Anatomie und Mechanik der Gelenke unter Berucksichtigung der bewegenden Muskein, 1904-1911, Vol. 3, Specielle Gelenk und Muskelmechanik, Gustav Fischer, Jena, 1911.
6. J. M. F. Landsmeer, "Studies in the anatomy of articulation," Acta Morphologica Neerlando - Scandinavica, Vol. III, no. 3-4, pp304-321, Jan, 1960.
7. A. E. Flatt, G. W. Fischer, "Biomechanical factors in the replacement of rheumatoid joints," Ann. Rheum. Dist., Vol. 28, 1969.
8. G. W. Fischer, "A Treatise on the Topographical Anatomy of the Long Finger and a Biomechanical Investigation of its Interjoint Movement," Ph.D. thesis, Engineering Mechanics, Univ. of Iowa, Univ. Microfilms, Inc. Ann Arbor, Michigan, 1969.
9. P. W. Brand, R. B. Beach, D. E. Thompson, "Relative Tension and Potential Excursion of Muscles in the Forearm and Hand," J. of Hand Surgery, Vol. 6, No. 3, pp. 209-219, May 1981.
10. K. N. An, E. Y. Chao, W. P. Cooney III, R. L. Linscheid, "Normative Model of Human Hand for Biomechanical Analysis," J. of Biomechanics, Vol. 12, No. 10, pp. 775-788, 1979.
11. J. Agee, P. W. Brand, D. E. Thompson, "The moment arms of the carpometacarpal joint of the thumb: their laboratory determination and clinical application," Proc. of the 37th Annual Mtng., Am. Soc. for Surgery of the Hand, No. 14, New Orleans, Jan 1982.
12. D. E. Thompson, "Biomechanics of the Hand," Perspectives in Computing, Vol. 3, No. 3, pp. 12-19, Oct 1981.
13. C. A. Ou, "The Biomechanics of the Carpometacarpal Joint of the Thumb," Ph.D. Dissertation, Department of Mechanical Engineering, Louisiana State University, Dec 1979.
14. W. L. Buford, Jr., "An Interactive three dimensional simulation of the kinematics of the human thumb," Ph.D. Dissertation, Dept of Engineering Science, Louisiana State University,1984, Ann Arbor, MI: University Microfilms International, 85-15 133, 1985.
15. W. P. Cooney, M. J. Lucca, E.Y.S. Chao, R. L. Linscheid, "The Kinesiology of the Thumb Trapeziometacarpal Joint," J. of Bone and Joint Surgery, Vol. 63A, No. 9, pp. 1371-1381, Dec 1981.
16. J. M. Ramselaar, Tendon Transfers to Restore Opposition of the Thumb, Leiden: H. E. Stenfert Kroese N. V., 1970.
17. P. K. Scherrer and B. M. Hillberry, "Piecewise Mathematical Representation of Articular Surfaces," J. of Biomechanics, Vol. 12, pp. 301-311, 1979.

# Simplified User Interface for Technical Systems

Charles S. Janik

Bell Helicopter TEXTRON

Fort Worth, Texas

The creation of a user interface to a wide variety of analysis and display programs can be accomplished in many different ways. Use of a process subprocess scheme with associated global commons allows for new modules to be added to such a system with the minimum of effort.

## The System as Image

In any large scale system that seeks to combine many different analysis and displays the implementation is required to maintain information about the current state of the system in order to allow for proper processing of the data. In most such systems the programmer has saved important values in commons and used these commons in the various subroutines that handle the individual parts of the application. This method of system building lacks sufficient flexibility in that adding a new analysis or display module to increase the functionality of the system requires that the driver routine be modified to include a call to the new module subroutine. That the new subroutine module be compiled and linked to the driver routine; and that the image as a whole be rebuilt. Besides these obstacles, there is a limit (even on virtual memory machines) as to how much code can reasonably be kept in a single image.

If we examine the functionality of the parts of the single image system we have built we see three distinct sections. First, the main routine, which retains the commons and does all the bookkeeping and setup required to operate the system. Next, a section in the main routine which allow for selection of the subroutine to be run. This part of any system is often a menu screen or command line prompt with its associated parsing rules to allow the user to select the flow of control to the actual module routines. Last, of course are the actual functional subroutines which do the work. In looking at this hierarchy we can see some important points which can guide us in revising our methods of system construction.

In figure 1 these three parts of our system code are labeled: Main Routine; Selection Section; and Function Section. Of these three parts the Main Routine does not change at all when a new function is added. The Selection Section may change by as little as a line or two in order to accommodate the new selection and its parsing rules. The Function Section will have a new routine added. From a programming and resource point of view the need to recompile the Selection Section is a waste of time; the need to relink the entire system image to add one function is also counter productive as the new module will often require serious debugging before final use. In fact the relinking is the largest single toll extracted in the module adding cycle as it is performed again and again during debugging. The changes to the Selection Section are often completed with little change thereafter. As the image gets larger the chance of an unsuccessful link increases as does the need for larger working set sizes. If there were some way to retain the common information and update the Selection Section as needed when functionality is increased via a separate image, the cost of adding a new function to the system would be simply a compile and link of the function and a revision of he parsing rules in the Selection Section code.

Fig 1.

```
+----------------------------------+
|           Main Routine           |
|                                  |
|  o RETAINS COMMONS               |
|  o DOES INITIAL SETUP            |
|  o DOES BOOKKEEPING              |
|                                  |
+----------------------------------+
|        Selection Section         |
|                                  |
|  o PROMPTS USER FOR INPUT        |
|  o PARSES THE INPUT              |
|  o SELECTS SUBROUTINE TO RUN     |
|                                  |
+----------------------------------+
|         Function Section         |
|                                  |
|  o PERFORM ANALYSIS OR DISPLAY   |
|                                  |
+----------------------------------+
```

## The System as Process-Subprocess

A model of how this desirable effect can be achieved is available to every user of a VMS system. In particular, VMS retains for each user process specific information, much like our Main Routine. It allows selection of images to run via the DCL tables, like our selection routine. Last, the addition of functionality is provided by images run to perform actions in response to DCL command input, just like our Function Section.

Any system of worth must be able to run with the minimum of privileges, just like the operating system. In building our system we must seek to emulate many features of the operating system without any privileges. To do this we will use a process subprocess pair with a global common to retain system information.

The process subprocess pair is necessary because under VMS it is not possible to have two images active in a single process at the same time. Further as we retain the information in a global common there must always be an image active referencing the global common in order to keep the common's reference count above zero. The process must therefore activate an image we will call SUPERVISOR which is comparable to the Main Routine in the single image system or the Process Control Block in VMS. Supervisor performs several functions. First, it learns the identity of the user and image with a $GETJPI call followed by queries to the terminal line, if our $GETJPI reveals an interactive job. Next it creates a Global common with the $CRMPSC command using the process id as a base to build the global section name. Third, the information garnered in step one is placed in the global common for use of the subprocess. The global common is page aligned and placed at a specific starting address to insure that the subprocess can associate with it. The page align and address setting is accomplished during the link via an options file like that in figure 2.

Fourth, a subprocess is initiated with it's process name based upon the process id of the creating process. The subprocess is started with a LIB$SPAWN call to allow the process to name the subprocess and set up IO streams back to the terminal. The call to LIB$SPAWN is shown in figure 3.

The SPAWN can either wait for completion or continue. If continuation is selected, the process image must synchronize its image rundown in order not to remove references to the global common. The command string references a command procedure used to perform the work of the selection section of the system diagrammed in figure 1.

Fig 2.

```
        .
        .
        .
$ LINK SUPERVISOR,(lib),G_COMMON/OPT
        .
        .
        .
$ TYPE G_COMMON.OPT

CLUSTER=COMMON_CLUSTER,%X200,,G_COMMON
PSECT_ATTR=G_COMMON,PAGE
[EOF]
        .
        .
        .
```

NOTE: G_COMMON.OBJ is a module that contains just the global common storage. The global common begins at byte address 512 or 200 hex.

FIG 3.

```
CALL LIB$SPAWN ( '@SUBPROCESS' ,,
                 'TT' ,
                 'TT' ,
                 ,
                 *note*,,
                 ,,,,'SUB> ',)
```

NOTE: Name of subprocess is based upon the process id of the parent process.

## The Subprocess Loop

The subprocess loop provides the funtionality of the selection section and the function section of figure 1. It is a simple DCL command procedure loop that provides for prompting; function selection; parsing; and function startup. This loop is presented in a simplified form in figure 4.

The image SELECTION.EXE is an executable that allows for the presentation of a menu or prompt to allow user input which is then parsed and the function selected is activated via a LIB$DO_COMMAND call. The SELECTION image is associated with the global common by retrieving the subprocess name via a $GETJPI call and then modifying the returned name to obtain the name of the global common. This global common name is used to map the global common created in the process to the subprocess image. In order to do this the subprocess image makes a call to $MGBLSC using the global section name. At the time this is done all information

in the global common is made available to the subprocess image. The image activated via the LIB$DO_COMMAND will often also link to the global common to retrieve session specific information.

The sequence of steps involved in the selection of an image for run are all handled via the LIB$DO_COMMAND which runs down the current image (SELECTION) and loads the function image (selected) to be run. At the conclusion of the selected function image the DCL loop falls through to the top of the loop and the SELECTION image is once again initialized to allow the user to choose a new option. To exit the system the image activated in the option selection section should be LOGINOUT.EXE.

Command Buffer Usage.

In any system such as that described above the need for a command buffer to pass execution information to the function image is required. In particular this corresponds to the call frame of the subroutines in the original method described in figure 1. These command buffers can be either fixed length or variable. If a fixed size command parameter buffer is chosen then the values can be saved in global common and passed on to any image that is linked with the option file of figure 2 and subsequently maps the global section.
If the buffer is made variable in length then the linkage to the command parameter buffer is more complicated. In fact the use of variable parameter buffers causes the LIB$SPAWN call to be done as no wait and then to coordinate the access to the global common. In this scheme we will return to the process and perform IO to a file using QIO's which read and write to the global common. In the VAX documentation set volume on RMS services a method of creating the FIB block for this sequential buffer file is presented. The buffer file's FID must be obtained from the NAM block to build a FIB for use in a series of QIO calls to directly update the file. These updates could be handled via RMS reads and writes, but that method is wasteful of memory for buffers which are not needed and processing time which could be better used in processing. The use of a file to hold the parameter buffer allows for the extension of the parameter buffer to any desired size and further allows for a record to be maintained of all calls to the function images to be kept for debugging. The file is opended with a standard RMS open statement and a block

FIG 4.

```
$ LOOP:
$ ON WARNING THEN GOTO LOOP
$ ON ERROR THEN GOTO LOOP
$ ON SEVERE_ERROR THEN GOTO LOOP
$ SET NOON
$ RUN SELECTION
$ GOTO LOOP
```

is written out to the file. The RAB address is obtained from which the RAB is formated. From the formated RAB we obtain the FAB address and format that structure. Finally from the formated FAB we obtain the NAM address which we use to format the NAM block to obtain the necessary info to build a FIB for use in direct access to the file. The use of the FIB for direct access to the file is covered in the IO users guide of the VMS documentation set.
To extend the file for a large buffer the QIO call is used to deaccess the file and then the channel is deassigned. The file is then opened for extend and additional blocks are written out to meet the size of the requested buffer. After the extend is completed the FIB is once again used to open the file for direct access to allow paging in of the requested part of the command buffer. Note that the subprocess does not handle the IO but must make request of the SUPERVISOR image to page in various blocks of the command buffer. To perform this feat the SUPERVISOR, SUBPROCESS, and (function) images must coordinate the order and block number of command buffer pages. This coordination can be accomplished via all the standard VMS methods i.e. event flag clusters, mailboxes, and files. Of these three methods the use of mailboxes to send commands and responses to the SUPERVISOR code from the subprocess images is the most understandable and expandable. The method of extending the command buffer on the fly in FORTRAN is provided in the example program in figure 5.

Conclusion

The use of a process-subprocess pair for a system provides for a flexible method of system expansion. The use of variable size command buffers allows the function images to not be restricted to the usual constraint of 254 characters per command line. As a final point the use of standard documented system services and library routines mean that the code is farely easy to support and sturdy over new releases of the operating system.

```
      PROGRAM FILEPAGER
C
C------------------------------DATA------------------------------
C
CPDL THE FORTRAN SUBROUTINE
C
      EXTERNAL FOR$RAB
C
CPDL INCLUDE FILES TO FORMAT RMS BLOCKS
C
      INCLUDE '($FABDEF)'
      INCLUDE '($RABDEF)'
      INCLUDE '($NAMDEF)'
      INCLUDE '($FIBDEF)'
C
CPDL INCLUDE FILES FOR SYSTEM SERVICE CALLS
C
      INCLUDE '($SYSSRVNAM)'
      INCLUDE '($SECDEF)'
      INCLUDE '($IODEF)'
C
CPDL QIO READ AND WRITE BUFFER
C
      STRUCTURE /S_DATA/
        LOGICAL*1      B_DATA(512)                        ! DATA BLOCK
      END STRUCTURE
C
CPDL RMS EXTEND BUFFER (NULLS)
C
      STRUCTURE /S_FILL/
        CHARACTER*512 T_BLANK                             ! DUMMY BLOCK
      END STRUCTURE
C
CPDL CHANNEL ASSIGNMENT WORD
C
      INTEGER*2  W_CHANG                                  ! CHANNEL WORD
C
CPDL BYTE FOR EQUIVALENCE TO J INDEX
C
      LOGICAL*1  B_J                                      ! INDEX BYTE
C
CPDL LOCAL STORAGE
C
      INTEGER*4  J                                        ! INDEX
      INTEGER*4  L_STATBLK(2)                             ! STATUS BLOCK
      INTEGER*4  L_ADDRESS(2)                             ! FIB DESCRIPTOR
      INTEGER*4  FOR$RAB                                  ! EXTERNAL
C
CPDL RECORDS FROM BUFFER STRUCTURES
C
      RECORD /S_DATA/ R_DATA
      RECORD /S_FILL/ R_FILL
C
CPDL RECORDS FROM RMS BLOCK STRUCTURES
C
      RECORD /FABDEF/ R_FAB
      RECORD /RABDEF/ R_RAB
      RECORD /NAMDEF/ R_NAM
      RECORD /FIBDEF/ R_FIB
C
CPDL EQUIVALENCE A BYTE TO THE INDEXES FIRST BYTE
C
      EQUIVALENCE ( J , B_J )
C
C------------------------------CODE------------------------------
C
CPDL OPEN THE FILE WE WILL ACCESS VIA QIO'S WITH A FORTRAN OPEN
C
      OPEN   (UNIT = 66 ,
     -        FILE = 'T00000000' ,
     -        FORM = 'UNFORMATTED' ,
     -        STATUS = 'NEW' ,
     -        RECORDSIZE = (512/4) ,
     -        BLOCKSIZE = 512 ,
     -        ORGANIZATION = 'SEQUENTIAL' ,
     -        IOSTAT = L_ISTAT ,
     -        ACCESS = 'SEQUENTIAL' ,
     -        RECORDTYPE = 'FIXED' ,
     -        SHARED ,
     -        INITIALSIZE = 3 ,
     -        EXTENDSIZE = 81 )
C
CPDL OBTAIN THE RAB ADDRESS FROM A FORTRAN SUBROUTINE CALL
C
      L_RAB = FOR$RAB ( 66 )
C
CPDL WRITE A BLOCK TO THE FILE
C
      WRITE ( 66 ) R_FILL.T_BLANK
C
CPDL COPY THE RAB USING SUBROUTINE
C
      CALL M_COPY_RABBLK ( %VAL ( L_RAB ) , R_RAB )
C
CPDL FROM THE RAB COPY THE FAB USING A SUBROUTINE
C
      CALL M_COPY_FABBLK ( %VAL ( R_RAB.RAB$L_FAB ) , R_FAB )
C
CPDL FROM THE FAB COPY THE NAM BLOCK USING A SUBROUTINE
C
      CALL M_COPY_NAMBLK ( %VAL ( R_FAB.FAB$L_NAM ) , R_NAM )
C
CPDL USE THE INFORMATION IN THE NAM BLOCK TO BUILD THE FIB c.f. IO USERS GUIDE
C
      R_FIB.FIB$L_ACCTL    = JIOR ( FIB$M_NOWRITE , FIB$M_WRITE )
      R_FIB.FIB$B_WSIZE    = 0
      R_FIB.FIB$W_FID_NUM = R_NAM.NAM$W_FID_NUM
      R_FIB.FIB$W_FID_SEQ = R_NAM.NAM$W_FID_SEQ
      R_FIB.FIB$W_FID_RVN = R_NAM.NAM$W_FID_RVN
C
CPDL CLOSE THE FILE
C
      CLOSE ( UNIT = 66 )
C
CPDL SET THE CURRENT BLOCK NUMBER TO THE FIRST BLOCK
C
      L_BLOCK = 1
C
CPDL GET THE CHANNEL NUMBER ASSIGNED DURING THE LAST OPEN AND DEASSIGN IT
C
      L_CHANNEL = R_FAB.FAB$L_STV
      L_STATUS = SYS$DASSGN ( %VAL ( L_CHANNEL ) )
C
CPDL DO FOR 512 BLOCKS
C
      DO J = 1,511
C
CPDL FILL IN ALL THE BYTES OF THE CURRENT BLOCK WITH THE BLOCK NUMBER
C
      DO I = 1,512
```

```
        R_DATA.B_DATA(I) = B_J
      END DO
C
CPDL ASSIGN A CHANNEL TO THE DISK FOR USE IN THE QIO CALLS
C
      L_STATUS = SYS$ASSIGN ( 'SYS$DISK:' , W_CHANG , , )
      L_CHANNEL = W_CHANG
C
CPDL CREATE THE FUNCTION CODE FOR THE ACCESS QIO
C
      L_FUNCTION = JIOR ( IO$_ACCESS , IO$M_ACCESS )
C
CPDL CREATE THE DESCRIPTOR FOR THE FIB
C
      L_ADDRESS(1) = FIB$K_LENGTH
      L_ADDRESS(2) = %LOC ( R_FIB.FIB$L_ACCTL )
C
CPDL OPEN THE FILE FOR QIO ACCESS
C
      L_STATUS = SYS$QIOW ( ,
     -                      %VAL ( L_CHANNEL ) ,
     -                      %VAL ( L_FUNCTION ) ,
     -                      L_STATBLK(1) , , ,
     -                      L_ADDRESS(1) , , , , , )
C
CPDL WRITE THE DATA BUFFER TO THE FILE AT THE CURRENT BLOCK
C
      L_STATUS = SYS$QIOW ( ,
     -                      %VAL ( L_CHANNEL ) ,
     -                      %VAL ( IO$_WRITEVBLK ) , , , ,
     -                      R_DATA.B_DATA(1) ,
     -                      %VAL ( 512 ) ,
     -                      %VAL ( L_BLOCK ) , , , )
C
CPDL DEACCESS THE FILE
C
      L_STATUS = SYS$QIOW ( ,
     -                      %VAL ( L_CHANNEL ) ,
     -                      %VAL ( IO$_DEACCESS ) , , , , , , , , , )
C
CPDL DEASSIGN THE CHANNEL TO THE FILE TO ALLOW RMS WRITES
C
      L_STATUS = SYS$DASSGN ( %VAL ( L_CHANNEL ) )
C
CPDL INCREMENT THE CURRENT BLOCK NUMBER
C
      L_BLOCK = L_BLOCK + 1
C
CPDL OPEN THE FILE FOR EXTEND
C
      OPEN   (UNIT = 66 ,
     -        FILE = 'T00000000' ,
     -        FORM = 'UNFORMATTED' ,
     -        STATUS = 'OLD' ,
     -        RECORDSIZE = (512/4) ,
     -        BLOCKSIZE = 512 ,
     -        ORGANIZATION = 'SEQUENTIAL' ,
     -        IOSTAT = L_ISTAT ,
     -        ACCESS = 'APPEND' ,
     -        RECORDTYPE = 'FIXED' ,
     -        SHARED )
C
CPDL WRITE OUT ONE MORE BLOCK TO EXTEND THE FILE
C
      WRITE ( 66 ) R_FILL.T_BLANK
C
CPDL OBTAIN A COPY OF THE FAB
C
      CALL M_COPY_RABBLK ( %VAL ( L_RAB ) , R_RAB )
      CALL M_COPY_FABBLK ( %VAL ( R_RAB.RAB$L_FAB ) , R_FAB )
C
CPDL CLOSE THE FILE TO ALLOW QIO ACCESS
C
      CLOSE ( UNIT = 66 )
C
CPDL GET THE CHANNEL NUMBER ASSIGNED DURING THE LAST OPEN AND DEASSIGN IT
C
      L_CHANNEL = R_FAB.FAB$L_STV
      L_STATUS = SYS$DASSGN ( %VAL ( L_CHANNEL ) )
C
CPDL ASSIGN A CHANNEL TO THE DISK FOR USE IN THE QIO CALLS
C
      L_STATUS = SYS$ASSIGN ( 'SYS$DISK:' , W_CHANG , , )
      L_CHANNEL = W_CHANG
C
CPDL SET UP THE FILE FOR QIO ACCESS
C
      L_FUNCTION = JIOR ( IO$_ACCESS , IO$M_ACCESS )
C
CPDL SET UP A DESCRIPTOR FOR THE FIB
C
      L_ADDRESS(1) = FIB$K_LENGTH
      L_ADDRESS(2) = %LOC ( R_FIB.FIB$L_ACCTL )
C
CPDL OPEN THE FILE FOR ACCESS
C
      L_STATUS = SYS$QIOW ( ,
     -                      %VAL ( L_CHANNEL ) ,
     -                      %VAL ( L_FUNCTION ) ,
     -                      L_STATBLK(1) , , ,
     -                      L_ADDRESS(1) , , , , , )
C
CPDL READ THE CURRENT BLOCK (IT WAS JUST WRITTEN VIA FORTRAN)
C
      L_STATUS = SYS$QIOW ( ,
     -                      %VAL ( L_CHANNEL ) ,
     -                      %VAL ( IO$_READVBLK ) , , , ,
     -                      R_DATA.B_DATA(1) ,
     -                      %VAL ( 512 ) ,
     -                      %VAL ( L_BLOCK ) , , , )
C
CPDL DEACCESS THE FILE
C
      L_STATUS = SYS$QIOW ( ,
     -                      %VAL ( L_CHANNEL ) ,
     -                      %VAL ( IO$_DEACCESS ) , , , , , , , , , )
C
CPDL CLOSE THE CHANNEL ASSIGNED
C
      L_STATUS = SYS$DASSGN ( %VAL ( L_CHANNEL ) )
C
CPDL GO TO THE TOP OF THE LOOP
C
      END DO
C
CPDL STOP/END
C
      STOP ' '
      END
C
```

```
      SUBROUTINE M_COPY_NAMBLK ( R_NAMI , ! INPUT NAM
     -                           R_NAMO ) ! OUTPUT NAM
C-----------------------------------------------------------------------
C
CT - M_COPY_NAMBLK ( R_NAMI ,
CT -                 R_NAMO )
C
CD - RETURN THE NAM FROM THE NAM ADDRESS
C
CS - R_NAMI IS THE INPUT NAM RECORD
CS - R_NAMO IS THE OUTPUT NAM RECORD
C
C-----------------------------------------------------------------------
C
      INCLUDE '($NAMDEF)'
C
      RECORD /NAMDEF/ R_NAMI
      RECORD /NAMDEF/ R_NAMO
C
      R_NAMO = R_NAMI
C
      RETURN
      END
```

```
      SUBROUTINE M_COPY_RABBLK ( R_RABI , ! INPUT RAB
     -                           R_RABO ) ! OUTPUT RAB
C-----------------------------------------------------------------------
C
CT - M_COPY_RABBLK ( R_RABI ,
CT -                 R_RABO )
C
CD - RETURN THE RAB FROM THE RAB ADDRESS
C
CS - R_RABI IS THE INPUT RAB RECORD
CS - R_RABO IS THE OUTPUT RAB RECORD
C
C-----------------------------------------------------------------------
C
      INCLUDE '($RABDEF)'
C
      RECORD /RABDEF/ R_RABI
      RECORD /RABDEF/ R_RABO
C
      R_RABO = R_RABI
C
      RETURN
      END
```

```
      SUBROUTINE M_COPY_FABBLK ( R_FABI , ! INPUT FAB
     -                           R_FABO ) ! OUTPUT FAB
C-----------------------------------------------------------------------
C
CT - M_COPY_FABBLK ( R_FABI ,
CT -                 R_FABO )
C
CD - RETURN THE FAB FROM THE FAB ADDRESS
C
CS - R_FABI IS THE INPUT FAB RECORD
CS - R_FABO IS THE OUTPUT FAB RECORD
C
C-----------------------------------------------------------------------
C
      INCLUDE '($FABDEF)'
C
      RECORD /FABDEF/ R_FABI
      RECORD /FABDEF/ R_FABO
C
      R_FABO = R_FABI
C
      RETURN
      END
```

# Portable Graphics Packages for the C Language

**J-F Vibert, J-N Albert, and M. Rousseaux**
CHU Saint-Antoine
Universite P. & M. Curie
Paris, France

## Abstract

Because of LUN conflicts between FORTRAN and Decus-C, it is not possible to use FORTRAN libraries such as PLOT10 and BENLIB from C programs. Three libraries entirely written in C were developed in order to access from C programs graphic displays and plotters. One is devoted to graphic displays and is intended to be used with TEKTRONIX 4010, 4100 and 4105 or compatible (VT241 in TEKTRO mode), one is for Regis mode, and the third for BENSON plotters. Their functionalities will be described. They allow management of 10 independent windows, work in subject and screen (or paper) space, absolute or relative, and color management. Device dependent code is grouped into a few modules, thus allowing easy portability to compatible devices. They are written in "standard" C in order to be portable. All sources and a full documentation are provided using the DECUS getrno C-tool (UNIX like manual) and will be submitted to the DECUS library.

After a ten year period of FORTRAN programming, only C is now used in our laboratory, for several reasons. The most important concerned the possibility of developing real time acquisition program without have to use an assembly language, such as MACRO 11, even for high speed sampling (see Vibert, 1986 in these proceedings). Unfortunately, LUN conflicts between FORTRAN and Decus-C do not allow calling of FORTRAN-written subroutines such as those found in PLOT10 (for video TEKTRONIX terminals and clones), or BENLIB (for the BENSON incremental plotters) from C programs using the available `call(...)` C function. Owning TEKTRONIX aliases and a BENSON 1332 plotter, we were led to write the necessary libraries. We run both RSX11M v4.2 and RT11 v5.2.

Four libraries entirely written in C were developed in order to access from C programs graphic displays and plotters. Three are devoted to graphic displays and intended to be used with TEKTRONIX 4010, 4100 and 4105 or compatible: QUMEC.OLB is for the 8 colors QUME QVT511 gtx, which emulates a 4100 and 4105 Tektronix terminal, AF410.OLB is for the french ARINFO AF410, a monochrome 4100 and 4010 Tektronix terminal alias, and VT241TEK.OLB is for the DEC VT240 et 241 in Tektro 4010 mode. PLOT.OLB is devoted to the BENSON plotter 1332, a microprogrammed incremental plotter with 3 colors.

All the code is entirely written in DECUS C, using only standard functions for purpose of portability. This code is fully modular and thus easy to modify if some new functions are to be added, or existing one to be modified in order to adapt to some new device. Device dependent code is grouped into few modules, thus allowing easy portability to compatible devices. All sources are documented in the code, and full documentation is provided using the DECUS getrno C-tool that produces an UNIX-like manual. This manual is in French for the video-terminal graphic package, and in English for the plotter package. Sources can be obtained upon request from the authors, and will be soon available through the DECUS Library.

## Video terminal graphic packages

The functions are grouped together in several files according to their role. The lower functions relate to the internal conversion from the decimal representation of values to the special tektronix code with low, high and extra weight ASCII characters. Only the name of the functions, without any detailed argument list will be given here. The following functions address this problem, and are only used by others functions of higher level: `conv_int()` converts an integer, `conv_txt()` a text string, `conv_gin()` a graphic input, `conv_a_s()` convert from absolute to subject space a coordinate value. Other functions allow to send an integer (`entier()`), a coordinate (`send_xy()`) and to move the beam either switched off (`a_move()`) or on (`a_draw()`).

The more device dependant functions are those that permit switching from one mode to another, i.e. from the Tektronix mode to the ANSI mode (`t_ansi()`), or vice versa (`t_tektro()`). When in Tektronix mode, it is pos-

sible to be either in alphanumeric mode (**t_alpha()**) or in vector mode (**t_vecteur()**). The screen area can be cleared by **t_erase()**, while the dialog area can be managed by several functions: **dial_area()** allows to turn it on or off, **dial_color()**, **dial_size()** and **dial_lines()** manage its color, buffer size and number of visible lines, while **clear_dial()** clears it.

For the color video terminals, it is possible to define colors for text (**coul_texte()**), for lines (**coul_lignes()**) and for the background area (**coul_fond()**).

Text can be drawn on the graphic display using t_texte() choosing the text direction (**txt_dir()**), rotation (**txt_rot()**) and size (**txt_size()**). Vector patterns can be chosen in order to obtain either a continuous (**trait()**), a dotted (**dotted()**), a dot-dashed (**mixte()**) or a dashed (**tiret()**) line. To mark a given position on the screen, **marker()** draws a marker of a given shape.

The screen can be subdivided in up to 10 windows that can overlap. Windows are created using **cre_window(w,xl,yl,xr,yr)** that defines the physical window by its absolute diagonal in the screen space. When created, a current working window must be selected using **window(w)** and scaled using **scale(xmin,xlen,ymin,ylen)** which makes the correspondence between the physical space on the screen and the user units referred to as subject space. Thus two spaces coexist, the screen space in pixel units referred as **tektro** (abbreviated as **t**) and the previously defined subject space in user units (abbreviated as **s**).

When a beam move or draw is to be done, the x-y coordinate given as an argument of the move or draw function can thus be given in either tektronix (pixel) or subject (user) units. These coordinates can also be understood to be either absolute (abbreviated as a) or relative (abbreviated as r) to the previous beam position. This leads to four move functions (**moveta(x,y)**, **movetr(x,y)**, **movesa(x,y)**, **movesr(x,y)**) and four draw functions (**drawta(x,y)**, **drawtr(x,y)**,**drawsa(x,y)**, **drawsr(x,y)**).

An important interactive medium is the possibility of moving a cursor on the screen in order to indicate a coordinate, a given part of a figure. The function **get_curs(x,y,char)** displays a cursor on the screen at the current position. Entering GIN mode (Graphic INput), allows one to move the cursor using the arrows, the numeric keypad or a mouse, and sends back its coordinates and the typed character when a key is depressed.

Other higher level functions allow, for example, drawing a surface filled with a predefined pattern (**begin_panel()** and **end_panel()**), drawing a rectangular frame (**cadre(length,width)**) or a curve (**courbe(x,y,nb_pts)**) whose coordinates are in two (x and y) floating arrays. Last but not least, the produced figure can be kept using **hd_cpy()** that sends a hard_copy, monochrome or color, full or half-size.

A header file (qumec.h, af410.h or vt241tek.h) must be included in the C source code in order to use all these functions. Moreover, it predefines several symbolic constants related to the colors, the character size and the text direction.

## Benson plotter graphic package

The graphic package developed for the Benson plotter produces an ASCII file that is optionally spooled on the Benson Plotter. It is composed of three modules: PLOT0 in which are all the very low level functions and the file setting, PLOT1 that contains the intermediate level functions allowing mainly the raw pen displacement, with the same names than in BENLIB.OLB (**traa(x,y,j)**, **tras(x,y,j)**), and text plotting (**pcara()**, **pcars()**, etc..., there exist 14 text plotting functions depending on the way the text must be drawn according to the pen position before and after the function call), and finally PLOT2 that contains the high level displacement functions. As for video functions, plotting knows two spaces: the Benson space (abbreviated as b) in centimeters, and the subject space (abbreviated as s) in user units, and two displacement modes: absolute (abbreviated as a) or relative (abbreviated as r). These pen displacements can be performed with the pen either up (with the generic function name **pmovxx**) or down (with the generic function name **plinxx**) thus leading to the following functions: **pmovba(x,y)**, **pmovbr(x,y)**, **pmovsa(x,y)**, **pmovsr(x,y)**, **plinba(x,y)**, **plinbr(x,y)**, **plinsa(x,y)** and **plinsr(x,y)**. A simpler way to manage the pen movements consist to preselect a working space, either Benson (**pbenson()**) or user (**puser()**), and then to use one of the following functions: **pmovea(x,y)**, **pmover(x,y)**, to move the pen in the current working space, or **plinea(x,y)**, **pliner(x,y)** to draw a line.

These libraries have been used as the graphic routine libraries in our lab for two years. The source code and sample programs can be obtained upon request through the authors, and were submitted to the DECUS Library where they will be available soon.

# HARDWARE AND MICRO SIG

# Giving A PDP-11/73 A Better Image

**Clyde L. Tyndale and Richard B. Waltz**
Lab. of Biophysics, IRP, NINCDS,
National Institutes of Health
at the Marine Biological Laboratory
Woods Hole, Massachusetts

**Abstract**

A PDP-11/73 system performs image processing of images obtained from light and electron microscopy via a video camera. The system consists of the KDJ-11A processor with 4 megabytes of memory, magnetic tape drive, floppy and Winchester disk drives, array processor, and a video processing subsystem, all requiring three mounting boxes. The operating system is RSX-11M Plus. The array processor gives the system VAX like performance when performing vector calculations. The video subsystem with a dedicated pipe-line ALU can perform many image processing operations at video rates. We experienced several system integration problems due to the high speed of the CPU, the number of expansion cables required, and the mixed vendor nature of the system. The system will be described in detail along with some of the problems and solutions we encountered along the way.

## Introduction

At the Laboratory of Biophysics in Woods Hole (LB/WH), the Section on Neural Membranes is conducting research into the structure and function of nerves and nerve cells. Some of this work involves the use of both light and electron microscopes to observe cell tissue. For a number of reasons, which may be different for different types of observations or experiments, the observed images require various types of processing before useful data can be extracted from them.

This paper will describe the system we are now using to perform this image processing and analysis. We will describe both the hardware and software used. Because the system evolved over a fairly lengthy period of time, some of the intermediate stages will be described, including some of the problems and solutions we encountered.

## System Objectives

### Image Subtraction

One of the research projects involves the study of the transport of particles in tissue taken from the giant axon of the squid, Loligo pealei. When live tissue is observed using a light microscope, small particles can be seen to move in apparently ordered ways. Because of the nature of the tissue observed and also due to imperfections in the optical system of even the best microscopes, Image Subtraction becomes a useful technique for improving the observed images.

At high magnifications, a fairly complex structure is observed. The large quantity of detail makes it difficult to observe the particles which are moving. However, if a reference image is captured and is then subtracted in real time from the observed image, those parts of the image which are static in time will be canceled out, leaving only the particles which move to be observed against a plain background.

The image may also contain various defects, such as dust and stray particles which are in the optical system of the microscope. It is possible to generate a reference image with the microscope de-focused from the specimen but with the defects (which are in a different focal plane) still in the image. If this image is then subtracted from an in-focus image of the specimen, the clutter caused by the defects can be reduced.

### Image Enhancement

Due to the nature of the specimens observed, the contrast of the images is poor. This may be true for both light and electron microscopy. Various image enhancement processes can be used to improve the contrast of the images in order to better observe the structure present.

### False Color

Another way to improve the apparent information content of an image is the use of false color techniques, in which

color is used to represent a particular gray level, a region of particular density, or other similar representations. This can improve the ease of interpretation of the image.

### Image Averaging

Some of the image detectors used, especially for low light level optical microscopy or for some types of electron microscopy have an inherent high level of electronic noise associated with them. This noise level is often sufficient to obscure the image and the structure we wish to observe. Averaging is a well known technique for eliminating or reducing the random noise associated with a detection process and it can also be applied to image analysis of static samples. If a successive series of images is captured and added together, the noise will tend to cancel out of the resulting summed image. The degree of noise cancellation is related to the number of image frames added together.

### Tomographic Reconstruction

The last problem we wished to attack involves the attempt to understand the fine structure of nerve tissue in three dimensions. The problem is obviously complicated by the fact that the microscopes used to observe the tissue at the necessary high magnification are inherently two-dimensional devices with a limited depth of field of focus. Using an electron microscope, we wish to obtain images taken at a variety of angles from a relatively thick specimen and then use computational methods to derive the three-dimensional structure from these images. The computations involved in this tomographic reconstruction process require a lot of forward and inverse Fourier, in addition to other mathematical operations on a large array of data points.

## Putting the System Together

One of the major constraints we worked under was limited funding. This forced us to build the system in piecemeal fashion over a period of several years. This was an advantage in one respect - we were able to refine some of our goals as we proceeded. On the other hand, proceeding in the fashion we did caused us to sometimes get "locked in" to an approach which we might have otherwise not used.

In May of 1982 we were able to obtain funding for the first part of the system, which consisted of some analog image enhancement equipment (Colorado Video), along with the video camera (Dage/MTI), monitor (Panasonic), and a Video Cassette Recorder (VCR) (Sony U-Matic) for storing images for off-line processing. By the following December, the equipment arrived and we were able to begin using it.

This system is shown in figure 1. The analog processing equipment allowed two basic operations which proved useful to perform in the analog realm: contrast enhancement including gray scale expansion, and shading correction. The shading correction is designed to help correct an image which is not evenly illuminated over its entire area, which is a common problem with microscope images. These two processors have proved to be easy to use (the operator merely adjusts a few knobs while watching the monitor) and provide significant initial improvements to the light microscope images.

In March of 1983 we received funding for the next stage of our system. At this point we had determined that a computerized digital image processing system would be required to perform all the tasks which we required it to do. Do to funding limitations we were unable to purchase a turnkey system, so we decided to put it together in-house.

At this time, we chose to build a system around a set of image processing subsystem boards manufactured by Imaging Technology, Inc. (ITEK). The ITEK subsystem consists of Frame Buffers (FB), an Analog Processor (AP), and an Arithmetic Logic Unit (ALU), all of which interconnect by means of a high-speed video bus. The subsystem is shown in figure 2. (Although three frame buffers are shown here, the system was originally acquired with only two; the third was added later.)

Since funding (as usual) was limited, at this time we could afford to order only the AP, the ALU, and two FB's. There was no funding available for a computer to operate the subsystem, so we decided to use the LSI-11/2 embedded within the EDAX (X-ray analysis system) accessory connected to our electron microscope. The EDAX system has an LSI-11/2 microprocessor, 32KW of memory, and a dual floppy disc, along with a DLV-11J serial line interface and a number of specialized modules for the EDAX work. It was practical to mount our ITEK subsystem in an outboard BA-11 type expansion box and connect it to the QBUS of the EDAX system.

In December of 1983, the equipment arrived and we began the system integration process. Figure 3 shows the system block diagram. We chose a BA-11 type box from MDB Systems mainly because it supplied the three power supply voltages ( +5v, +12v, and -12v) at sufficient current as required by the ITEK modules. We ran the system under the RT-11 operating system and used a set of software driver modules along with a demonstration program obtained from ITEK. The driver modules are written in MACRO-11 and are FORTRAN callable. The demonstration program is written in FORTRAN.

The system integration process proceeded smoothly and by February of 1984 we had a running system which could do some of the processing functions we required but had a few limitations, most of which were due to the slow LSI-11/2 processor, the small memory size, and the lack of a large mass storage device. Due to these limitations, we opted to do any large calculations on our PDP-11/60 system, transferring image data via floppy disk. Because our 11/60 only had RX01 single density floppies (480 block capacity), two floppy disks were required to store a complete image since each image required 533 blocks at 512 bytes per block.

At this point we had a limited capability system which could do real-time image subtraction and other enhance-

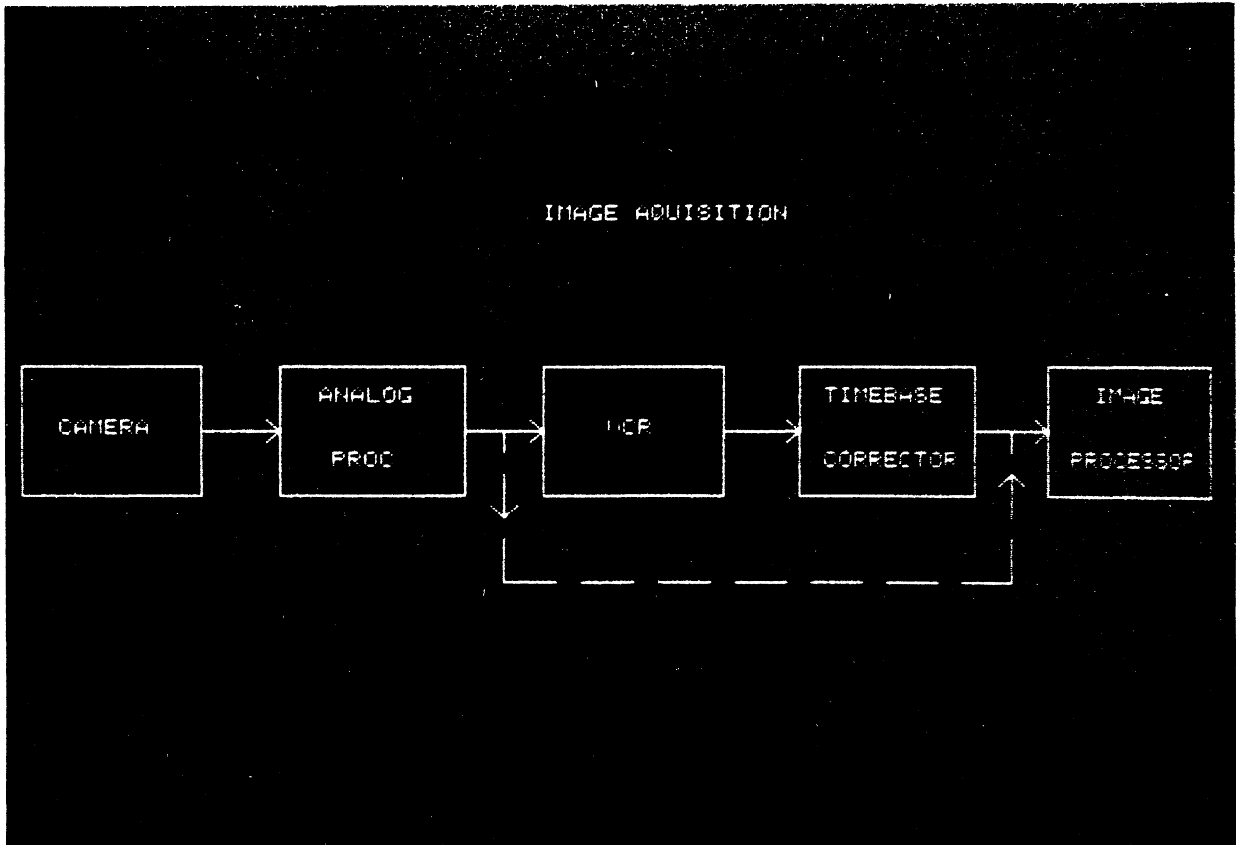



**Figure 1:** Block diagram of the Analog Video Processing subsystem

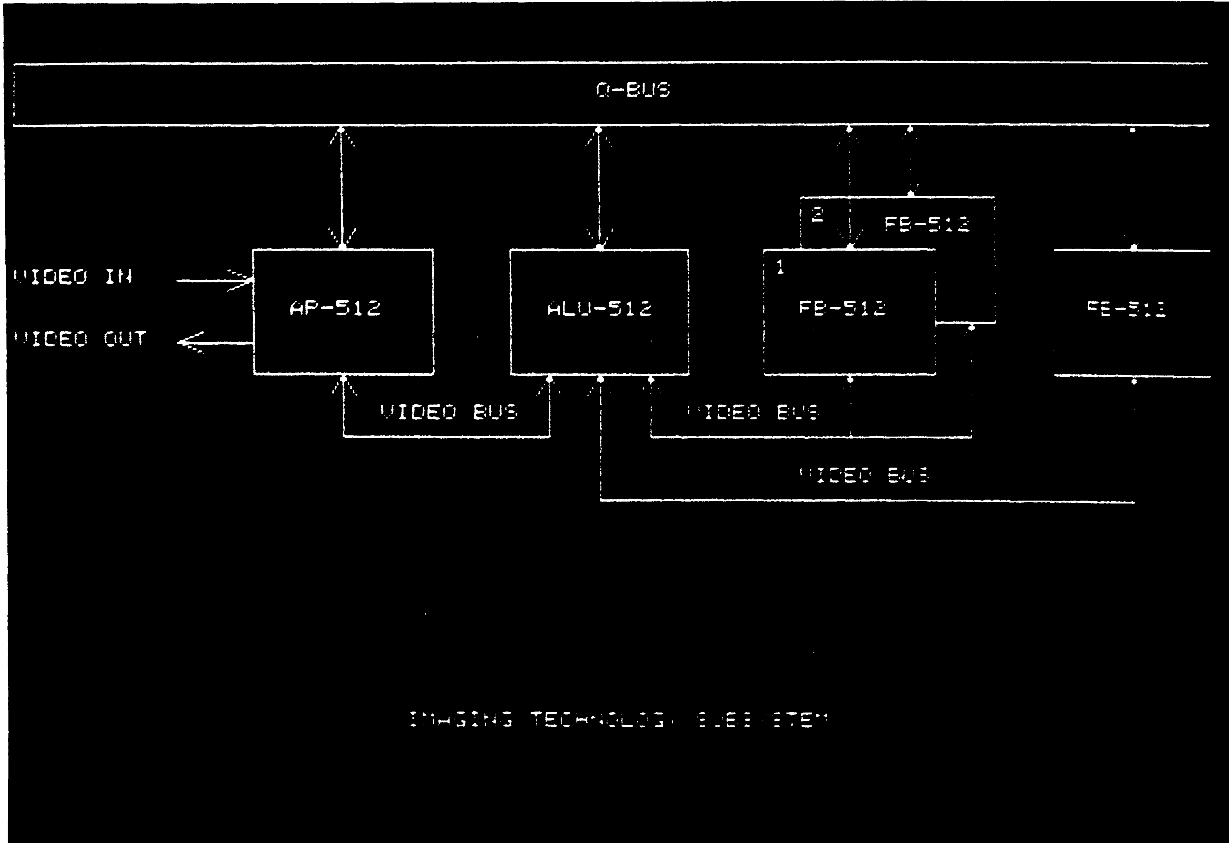



**Figure 2:** Block diagram of the ITEK image processing subsystem.

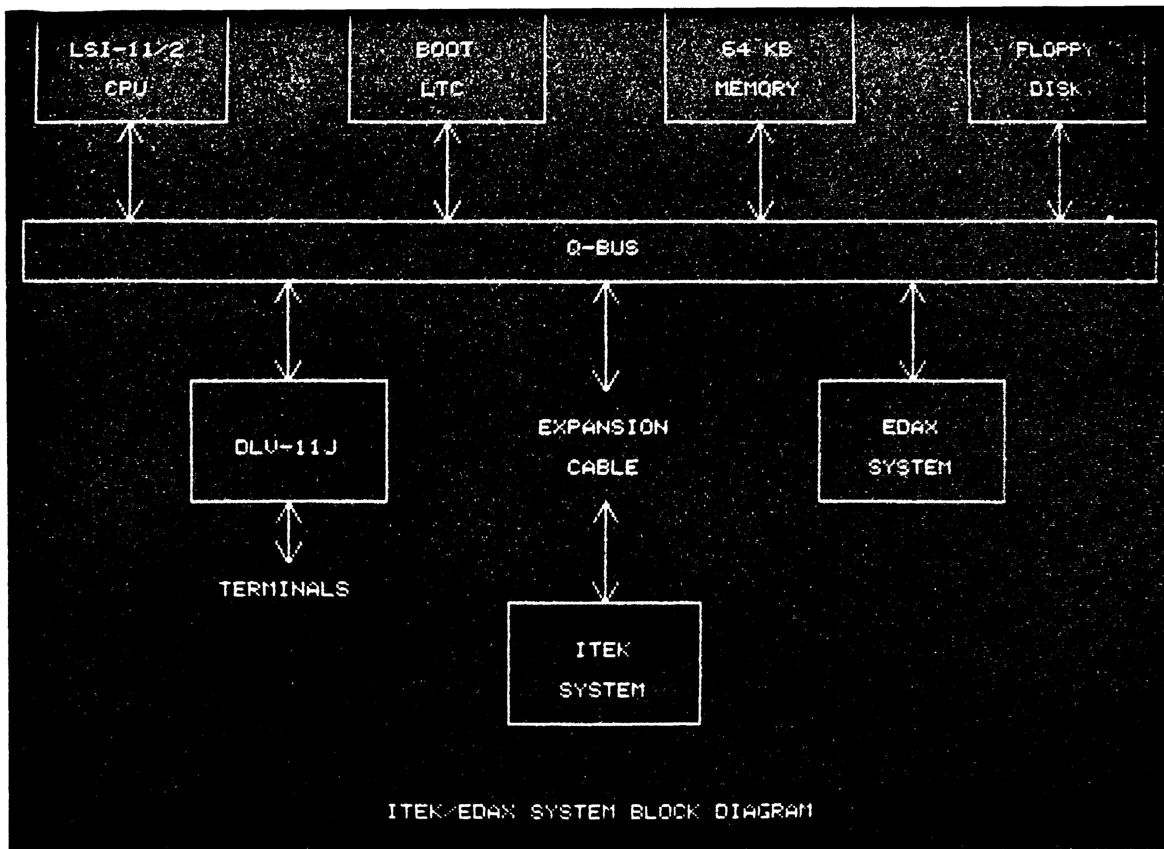



**Figure 3:** Block diagram of the LSI-11/2 based image processing system.

ments. Even with the limited CPU speed, some real-time processing was possible because of the high speed video bus inter-connection between the ITEK FB's, AP and ALU. In fact, this was the feature that attracted us to this subsystem; the amount of processing that it could do with minimal intervention from the QBUS CPU and minimal loading of the QBUS.

The major system limitations at this point included the lack of color (no color monitor), small memory, limited mass storage, and limitations on our ability to do image averaging and convolution due to FB overflows. The last would not have been a problem if we had been able to have three FB's on the system (figure 2). In addition, any processing which required much CPU processing or memory had to be done on the PDP-11/60, so we were limited by not being able to do these operations on-line. On the 11/60 we used a partial implementation of the Micrography Data Processing Package (MDPP, reference 1) for image processing software.

At this time it was obvious that in order to accomplish the rest of our project goals, we would need to acquire a dedicated computer to operate the ITEK subsystem, along with sufficient memory and mass storage, and additional FB and a color monitor. A major improvement in the operating software would also be required. In February of 1984, funding for this next stage became available.

At this time, a software package became available for the Multibus version of the ITEK subsystem which would have met many of our needs. If we had been able to start from scratch at this time we might have chosen to use the Multibus version, but since we already had a significant investment in the QBUS version of the ITEK subsystem, it was necessary to specify a QBUS computer system.

We specified a system consisting of the PDP-11/73 CPU, 4 Megabytes of memory, dual floppy disks (DSD), a 340 Megabyte Winchester disk (Fujitsu Eagle), a 9-track magnetic tape drive (Cipher), DLV-11J serial line interface and a SKYMNK array processor, as shown in figure 4. Because of the number of modules and the power required by them, the system was specified to be mounted in two BA-11 type mounting boxes.

We chose the 11/73 for several reasons: as a QBUS machine, it would operate with the ITEK QBUS boards on hand, with 22 bit addressing, it would operate with a large memory, it implements the full PDP-11 instruction set, it is the fastest QBUS processor, and with I and D space it would allow use of larger blocks of memory for program and data storage.

The SKYMNK Array Processor (Sky Computer) is a key part of the system. It has Direct Memory Access (DMA) to allow direct and fast transfer of data into and out of memory. It allows the use of very large blocks of memory to hold results of intermediate computations, a key point for doing large Fourier calculations; otherwise disk I/O would bog down the speed of the system. It can perform matrix computations at very high speed. It was supplied with a set of FORTRAN callable software drivers.

We originally planned to operate the system under RT-11, in order to use existing software, but the vendor could not supply a handler for the very large disk, so since we did not wish to write one ourselves, RSX-11M+ was substituted.

In August of 1984 the additional frame buffer arrived and was integrated into the LSI-11/2 based system with no problems. In December of 1984 the PDP-11/73 system arrived and the fun began!

### System Integration Problems

The first problem we experienced with the system was minor, although annoying: After we booted up the system, we found that the system time ran at twice normal speed. Because there were two identical mounting boxes (Sigma had been supplied), each with a full set of console switches, we learned that if both LTC switches were enabled, the clock ran at twice speed; if only one switch was enabled the clock ran at normal speed.

At this point we had an operating base level system. We performed the SYSGEN with no unusual problems. We then added our MDB mounting box containing the ITEK subsystem to the system.

One of the complications experienced in adding in the additional box was the bus terminations. In the Sigma box, the terminators are a resistor network soldered in place. In the MDB, they are a plug-in unit. For this reason, it was decided to implement the system by placing the MDB box in the middle between the SIGMA boxes.

The system refused to boot up! The boot routine crashed part way through. After a lot of head-scratching, consultations with Emulex (the boot ROM was theirs, on the SC03 Disc controller), and careful study of the DEC address scheme we discovered the problem. The first unexpected symptom we noticed was that the system refused to boot only when the ALU was present, leading us to conclude that there was an unexpected interaction between it and the boot program. The addresses used for the ITEK units were:

```
17770000-17770016: Frame buffer FB2,FB3
17770020-17770026: Analog Processor
17770040-17770056: Frame buffer FB0
17770200-17770216: Arith Logic Unit
```

We had used these addresses mainly because the units were supplied to us addressed this way. A check of Digital's address assignments showed that this address region was not assigned for the QBUS, so we had left them that way. They had operated successfully for some time on the LSI-11/EDAX system. However, one of our reference books (1980 Microcomputer Interfaces Handbook, DEC) did show that the region 17770000-17770376 was "Digital Reserved" for the UNIBUS. Further investigation showed that 17770200 is the starting address for the UNIBUS MAP Registers for the UNIBUS Memory Management Unit. A check with Emulex confirmed that this was the problem; the same PROM was used for both UNIBUS and

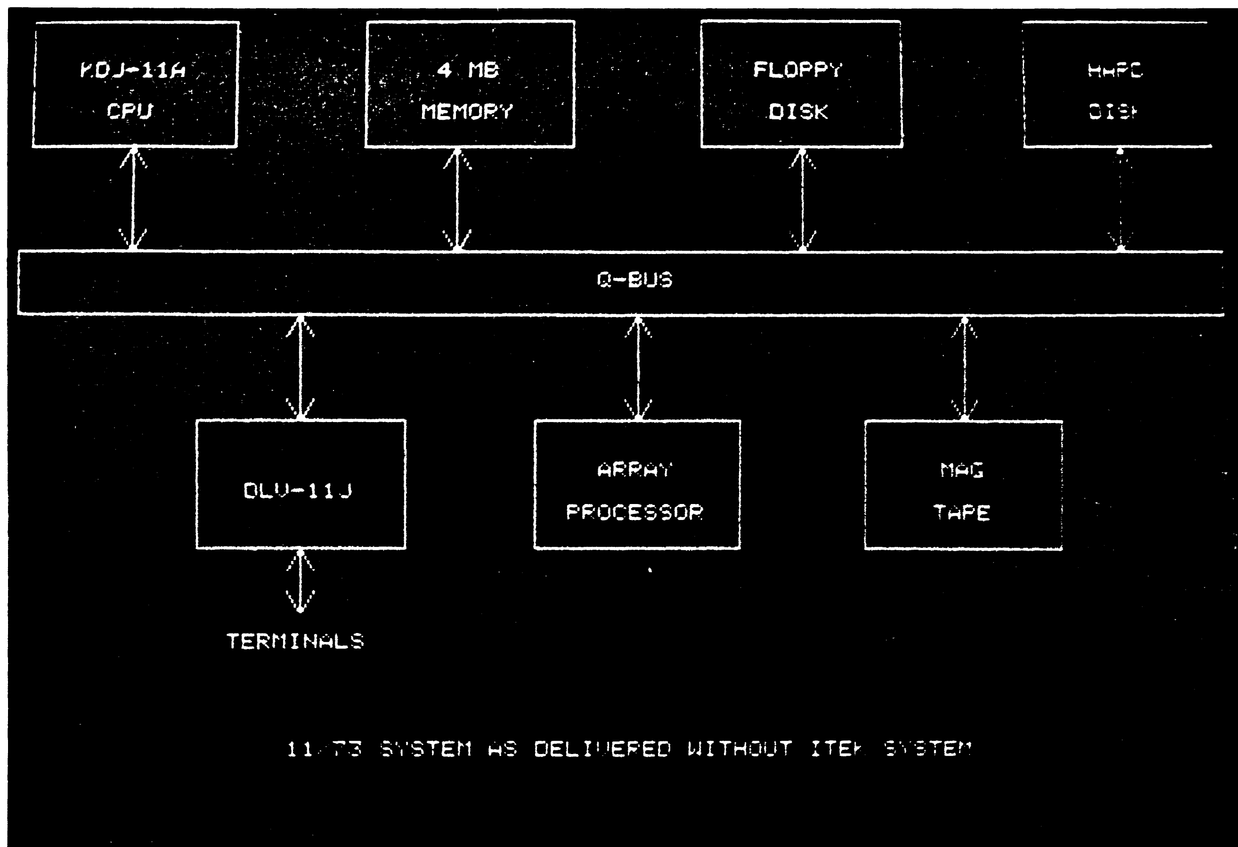



**Figure 4:** Block diagram of the PDP-11/73 system before the image processing subsystem was added.

QBUS and the presence of a device at 17770200 caused our problem. The solution was simple; we simply changed the ALU address to 17770060-17770076.

Now that we had a system that would boot properly, the next problem appeared promptly: the system crashed at apparently random times, sometimes even before the indirect command file at boot time completed. At other times the system would start up correctly and then crash during operation, in particular when the tape drive was used (it seemed to happen more often when running BRU).

After much trying of combinations of equipment on the QBUS, we learned that the problem was related entirely to the use of three mounting boxes; we had the problem whenever there were three boxes in the system, even if one of them was empty. Bus loading did not seem to be the culprit; only the number of mounting boxes (and consequently also the number of expansion cable sets). At this point we started trying to get some more information.

The bad news came first. Review of the DEC Micronotes showed that DEC did not support a multiple box 22-bit QBUS system with the KDJ-11A processor. Micronote #5 says that such a system is "Not currently configurable with DEC equipment". Micronote #35 shows how to properly configure a two-box 22-bit system with a KDJ-11A, but says that three box systems can be done for 18-bit systems only.

At this time (March of 1985), we were getting pressured to work on other projects for our laboratory. In order to gain as much use from the system as possible, the system was re-configured into a two-box system. Because of power supply and space limitations it was necessary to remove some options in order to do this. We chose to remove the floppy disc drive and 2 Mb of memory. The remainder of the system, enough to allow program development to proceed, was configured as a two-box system, using one Sigma and one MDB box, since only the MDB had the necessary power supplies for the ITEK system. Figure 5 shows the block diagram of the "two box" system.

About this time, in order to be able to install more terminals on the system, an additional DLV-11J was installed. One of the additional terminals was a LA-36 DECwriter. When this terminal was powered down at the end of the day, the system halted. The problem was unique to this terminal; terminals connected to other ports did not halt the system when powered down. We traced this problem to an easily overlooked "feature" on the DLV-11J: the HALT on BREAK option. A powered down terminal looks to the DLV-11J like it is sending a continuous BREAK signal. We fixed this problem by disabling the HALT on BREAK option.

The following December, it was possible to return to the three-box problem. Since the problem seemed to be related only to the number of expansion boxes used, we consulted with the manufacturers of the boxes for suggestions. The answer came from MDB, who also happened to be the manufacturers of the expansion cable sets: the problem was noise coupling between signal lines on the expansion cables, a problem that was due in part to the high speed of the 11/73. The suggested remedy was to use a re-designed version of the expansion cable assembly that changed the signal groupings so as to reduce coupling between critical signals.

By March of 1986, the new expansion cables arrived and the problem was solved! We now reinstalled the floppy disk controller and the 2 MB of memory that had been removed and the complete system worked without problems. The final system block diagram is shown in figure 6.

## Image Processing Software

We presently have three different image processing software packages that use the PDP 11/73 system. They are:

- Micrography Data Processing Package (MDPP)
- LUNG
- Imaging Technology Processing Package (ITPP)

### MDPP

The MDPP system (derived in part from P.R. Smith's University of Basel system) was put together to run under RSX. It was developed over many years in a number of different laboratories and has a relatively large number of users. The principal author is P.R. Smith, reference 1). Other versions run on a number of different DEC and IBM systems. The software is programmed in FORTRAN 77 with the capability to send a command line to the monitor (MCR) from a FORTRAN program (i.e., to request an additional task to run). MDPP is operational on our PDP 11/73 system.

In MDPP, the ITEK subsystem is used primarily for collecting and displaying images. Since the original designers of MDPP did not have an ITEK subsystem, the processing subroutines do not make use of its special features. The SKYMNK array processor is not yet fully integrated into the MDPP software for the same reason. The software is used primarily in the study of crystalline biological structures.

### LUNG

The LUNG program is a software package used for 3-D reconstruction from stained serial sections. Images are obtained by taking photographs of stained slices of biological structure and manually tracing the outline of the desired sub-structures using a digitizer to produce sections. The sections are processed to calculate volume, area, and sphericity index information. The traced data is processed, translated and rotated to obtain stereoscopic projections for output to a Tektronix 4010 graphics display for observing various perspectives. This program was developed in another laboratory (S.D. Chawla, reference 2) to study the volume and distribution of tumors in the lungs of experimental mice. It will require a significant amount of rework to fully utilize the features provided by our video and array processors.

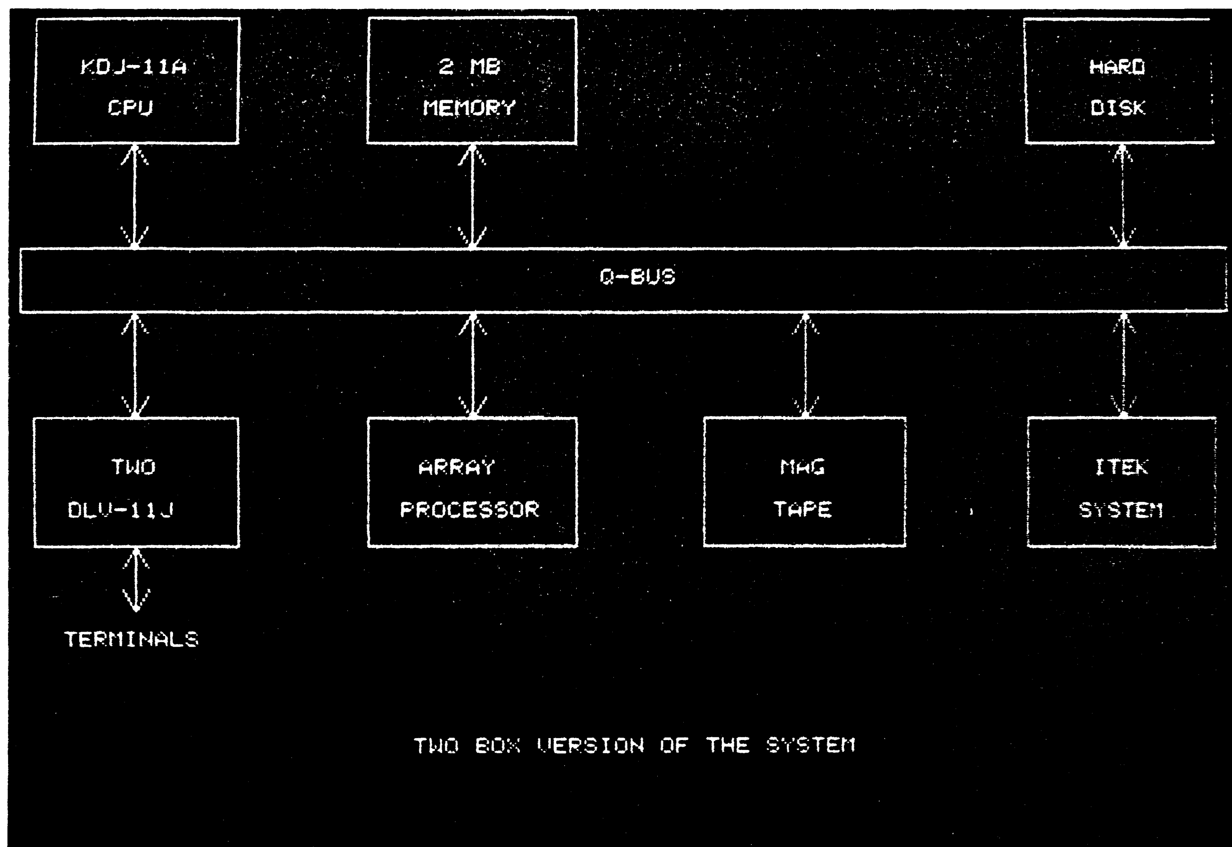



**Figure 5:** Block diagram of the "two box" limited image processing system.

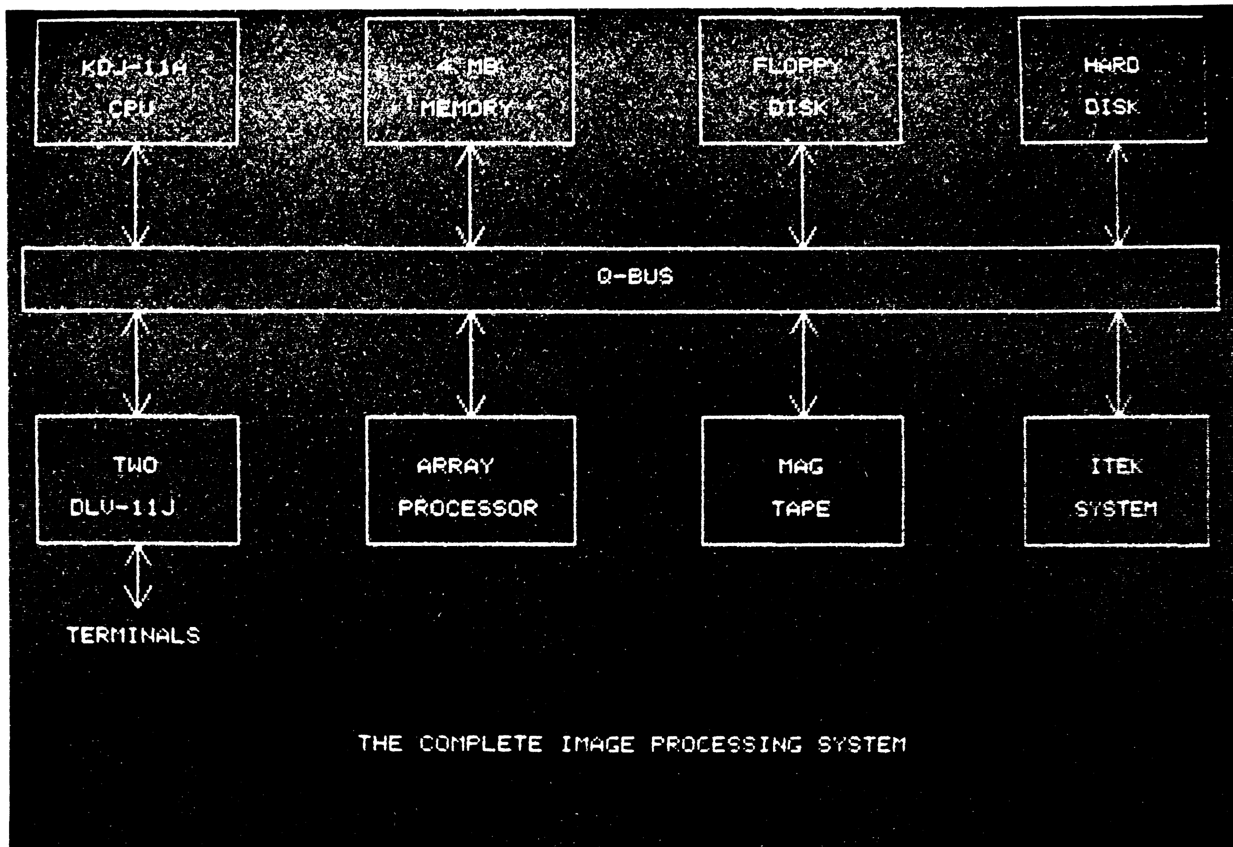



**Figure 6:** Block diagram of the complete image processing system.

## ITPP

The ITPP software package uses some of the features implemented in the MDPP software. ITPP is being programmed in our laboratory and will therefore be described in more detail.

The ITPP programs (tasks) are executed under the control of a main menu and sub-menus plus prompts as shown in figure 7. We took advantage of the multi-tasking features of RSX-11M+ and the ability of a task to activate a second task. The communication between tasks and sub-systems hardware is performed through subroutines supplied with the system.

The video processor subroutines were originally written for the RT-11 operating system and had to be modified for use with RSX-11M+. This was accomplished by generating a DEVICE COMMON and mapping the COMMON with each task. There were other minor changes required to make the original subroutines compatible with RSX-11M+.

The ITEK subsystem has one significant limitation for use in a multi-tasking environment. It lacks hardware interrupt logic. This requires that the drivers must monitor status conditions under program control. This presents problems when multiple users attempt to use the PDP 11/73 during tight timing loop execution (i.e. during real time image subtraction). When these real-time tasks run in the system they must run with a high priority in order to block other user's tasks from executing. This prevents data loss and other problems such as video display monitor flicker.

The SKYMNK array processor was delivered with a set of driver-level and data manipulation routines. It has logic for hardware interrupts and DMA interface to the 11/73 memory. It is tightly coupled to 11/73 memory because it has a small local memory. When large arrays are being processed, QBUS activity approaches the upper limit. This can cause problems for other system users, especially for the Magnetic tape which tends to exhibit data late errors under these conditions. Again, this has not been a serious problem on our system because of the limited number of concurrent users on the system.

As a result of the above limitations, we do not use the system as a true multi-user system, especially when real-time image processing is being performed. At other times, concurrent use by two or more users has not been a problem as long as the array processor is not being utilized. Usually, the only secondary use of the system is for program development and testing.

The ITPP programs are written in FORTRAN and MACRO-11. The program package is being used daily and various modifications and additions are evolving as experience is gained with use.

### Menu

The system as previously described is a tool used by the scientist in a research project. The primary task is to do research with a minimum burden from hard to use equipment during an experiment. Therefore we implemented the software for ease of use by employing a menu for program commands. A series of two letter commands specifies the operation or function (task) the user wishes to perform. The main menu is not normally displayed, but may be printed on the users terminal with the HE command. A list of Display Commands is given in table 1.

```
COMMAND   Meaning of command
---------------------------------
VO     VIEW FBO
IS     REAL TIME IMAGE SUBTRACT
V2     VIEW FB2
SN     SNAP (COPY VIDEO INPUT INTO FBO
           OR FB2)
V4     VIEW AP512 VIDEO INPUT (CHANNEL 4)
SS     SET SYNC
DL     DEFAULT LUT FILE TO LUT
           LookUp Table)
LD      LUT TO DEFAULT LUT FILE
LL     LINEARIZATION OF LUT
MT     MULTI-LINEAR TRANSFER FUNCTION
WL     WR LUTs TO DISK FILE
RL     WR DISK FILE TO LUTs
HG     COMPUTE HISTOGRAM
DH     DISPLAY HISTOGRAM (DEFAULT FILE)
EQ     HISTOGRAM EQUALIZATION
LS     HISTOGRAM LINEAR STRETCH
CV     CONVOLUTION
CO     COPY FBO TO FB2 FULL IMAGE
C2     COPY FB2 TO FBO
CS     COPY SUB-IMAGE CURSOR SELECTED
FC     PERFORM FUNC ON IMAG
AV     IMAGE AVERAGE
SP     SLOW PASS
II     INITIALIZE ITEX
LM     LUT MANIPULATION
VX     VITEX (write image to diskfile)
GR     GRAPHICS (Draw Line, Box,
           Circle, etc.)
SF     SKYMNK FFT(s) PROCESSING
FF     11/73 FFT(s)
DE     DILATE/ERODE AN IMAGE
MD     MDPP PROGRAMS (data capture
           programs)
PW     SPAWN COMMAND LINE (entered from
           console)
SK     TEST SKYMNK (MNK)
TI     TEST ITEX VIDEO PROCESSOR
HE     HELP
EX     EXIT
PI     INITIALIZE FB,ALU,AP (video
           processor modules)
UT     RUN PIP
MV     MOVE IMAGE
ZR     ZOOM OR ROAM
PX     PITEX (read image from disk)
```

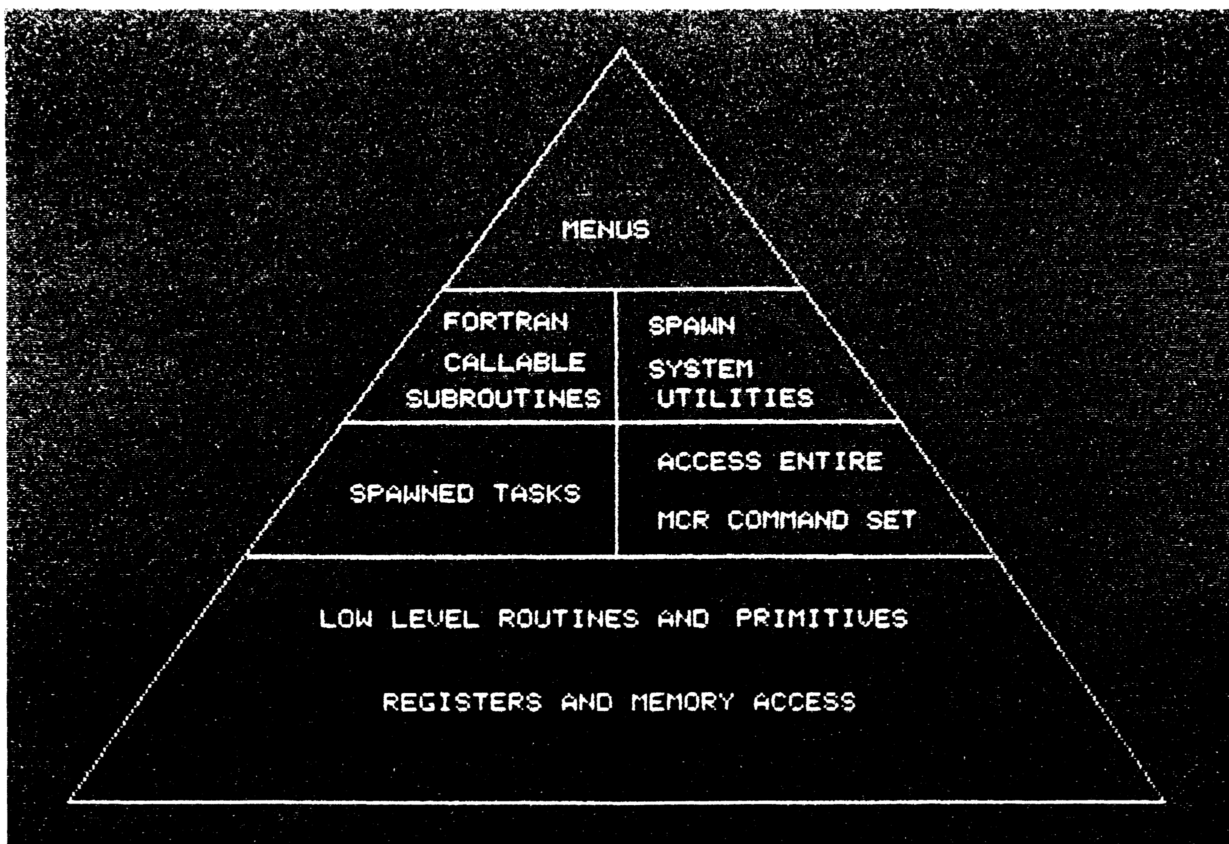



**Figure 7:** Hierarchy of the ITPP software package

```
SL    GREY LEVEL SLICING
YP    DISPLAY Y PLOT (pixel intensity)
          OF LINE CONNECTING 2 POINTS
YS    DISPLAY Y-SCAN PLOT (pixel
          intensity) OF SELECTED ROW
CC    COLOR CHART (display of LUT(s)
          on monitor)
```

In a typical session, the user logs in, selects ITP by typing ITP¡cr¿, then normally types a two character command to select a function to be executed (i.e. AV for image AVerage command). The command is decoded and then the appropriate task is SPAWNed or, in some cases, a second menu may be displayed to allow the user to select an operation from a larger group of tasks to be SPAWNed. When the OFFSPRING task is SPAWNed the user is presented with a series of prompts requesting entry of the required parameters for the task. A prompt ("*") is displayed on the user's terminal after the OFFSPRING task exits, to indicate readiness to accept the next command.

The program does not allow a user to jump directly from one sub-menu to another sub-menu. The only allowable path is back to the root or main menu and then back down the tree to the appropriate node or sub-menu. The tasks have been written to permit a reasonable amount of flexibility in organizing the analysis but the user should bear in mind that the order is important if sensible results are to be obtained.

The main menu is processed with the FORTRAN statements given in Program 1.

General FORTRAN statements for spawning a task or or another subroutine with a sub-menu take the following form:

```
nnnn    CALL SPAWND (NAME,NBYTES)
               OR
        CALL Sub-Menu

        GOTO 5000
```

## SPAWN(ed) task

An active task may activate another task by issuing a SPAWN directive. The task issuing the spawn directive is called the PARENT and the SPAWNed task is called the OFFSPRING. The two FORTRAN directives

```
CALL SPAWN (MCR,,,IEFN,,,,NAME,
     NBYTES,,,,IDS)
CALL WFLOR(IEFN)
```

worked satisfactorily in our system. The parent task must be installed as `...tsk` where tsk is a three character symbol (i.e. `...ITP`). The byte array NAME contains the offspring task to be activated (i.e. `RUN CONVOL` for the command "CV" which allows the user to perform convolutions on an image stored in a frame buffer). RUN is required for all non-installed tasks and `tsk` for a task installed as `...tsk`. NBYTE designates the number of characters in the command line (NAME). The Executive directive "CALL WFLOR(IEFN)" waits for the event flag IEFN to be set upon completion of the OFFSPRING task CONVOL and the exit of `MCR...` task. There is only one OFFSPRING task running in a given time interval. As a result, the WFLOR is sufficient for inter-task communication by the SPAWN directive. Chapter 5 of the Executive Reference Manual (reference 3) gives a more comprehensive example of activating one or more offspring tasks.

NAME and NBYTE is obtained by an appropriate FORTRAN statement.

```
        READ (5,100)NBYTES,(NAME(K),K=1,45)
100     FORMAT(Q,45A1)
        CALL SPAWND (NAME,NBYTES)
```

If PW is selected from the main menu, the user will be prompted to type a command line. For example, if "RUN CONVOL" or "FLX" is typed, the call to SPAWND will be:

```
CALL SPAWND ('RUN CONVOL',10)
```

or

```
CALL SPAWND ('FLX',3)
```

The subroutine SPAWND contains the two Executive directives SPAWN and WFLOR as given in Program 2.

## Batch processing

Tasks may be run using the Batch Processor, allowing a sequence of commands to be processed. Batch processing is performed by submitting a JOB to the Queue Manager with the appropriate Batch Commands plus the image processing commands and terminal input data. The terminal data is handled via a virtual terminal input as obtained from the JOB list. These two approaches give the scientist a choice of operating procedures to allow for the command and terminal data requirements of the specific experiment.

## Task to task communication

Sub-system parameters and flags, along with program control data are passed from task to task via default disk files. Some typical data passed by default files are LookUp Table (LUT) values, histogram values, and convolutions kernel files. This technique was chosen rather than using the SEND/RECEIVE directive or a shared (COMMON) region because of the ease of implementation and the amount of information that can be passed. Commands (i.e. WL) are available for storing LUT on a disk by other than default files.

## Image storage

Image data to be passed from task to task is normally kept in Frame Buffers (FB), but sometimes is stored in 11/73

main memory. This communication is transparent to the user. We found this to be an acceptable method since only one user has access to the video and vector processors during a session. Commands are available for copying images or sub-images from one FB to another, between a FB and main memory. Image data management for secondary storage is limited to writing (command VX) and reading (command PX) between frame buffers and disk storage. The hard (Winchester) disk has 340 Mbytes of storage, thus providing enough space for all our accumulated images to date. DEC utilities are used to backup disk data files. The user must organize the data within these constraints.

## APPLICATIONS

The ITPP software is presently used for image enhancement. The package also includes graphics capability to mark or label features in an image and to label each image for proper identification. The graphic subroutines can also be used to produce illustrations. With a color photographic unit (N.I.S.E., Rembrandt) attached to the system, 35mm slides may be easily made of the results. The block diagrams in this paper were created using the graphics subroutines of the system. Listed below are some of the image enhancement features which are now operational on the system.

- Image Enhancement by Histogram Modification
  - Histogram Equalization
  - Histogram Linear Stretch
- Image Smoothing
  - Neighborhood Averaging
  - Lowpass Filtering
  - Averaging of Multiple Images (see figures 8 and 9)
- Image Sharpening
  - Sharpening By differentiation
  - Highpass Filtering
- Pseudo-color Image Processing
  - Density Slicing
  - Gray-level to color Transformation
- Image Subtraction
  - Removal of microscope artifacts from the image (see figure 10)
  - Detection of movement in a series of images

### Graphics Features

The graphics subroutines operate by means of a cursor displayed on the video monitor. The operator controls the cursor position by using the numeric keypad. The even number keys are used to move left, right, up or down. The odd number keys are used to move diagonally, except for key number 5 which is used to terminate cursor movement. The menu for the graphic features is:

```
Command  Operation
-------
SF       Select Framebuffer
CI       Set Cursor Intensity
PI       Set Pen Intensity
DL       Draw Line
DR       Draw Rectangle
RF       Rectangle Fill
DC       Draw Circle
IT       Insert Text
EX       Exit
HE       HElp
SC       Set Cursor
CC       Clear Cursor from monitor
```

### Arithmetic operations

The ITEK subsystem has an ALU for performing arithmetic operations at video rates (i.e. subtract at a 10 megahertz rate). ITPP provides a menu for selecting the follow operations:

```
COMPLEMENT FBm; copy results
        into FBn (FBm ==> FBn)
AND   FBm ^ FBn ==> FBn
OR    FBm v FBn ==> FBn
ADD   FBm + FBn ==> FBn
SUB   FBm - FBn ==> FBm
ADDM  FBm + c   ==> FBm,
        where c is a scalar constant
SUBM  FBm - c   ==> FBm
```

### Hardware Tests

The ITEK subsystem and the SKYMNK array processor can be exercised using software supplied by the manufacturers. These test are accessible via the main menu. The IOX task may be run by entering the command PW and typing `Run $IOX`, to provide a means of exercising system peripherals.

## VECTOR PROCESSING SPEED COMPARISONS

The tomographic three dimensional (3-D) reconstruction problem requires a considerable amount of array processing. The PDP-11/73 without special hardware would be to slow to process the amount of data required for our particular application in a reasonable amount of time. The

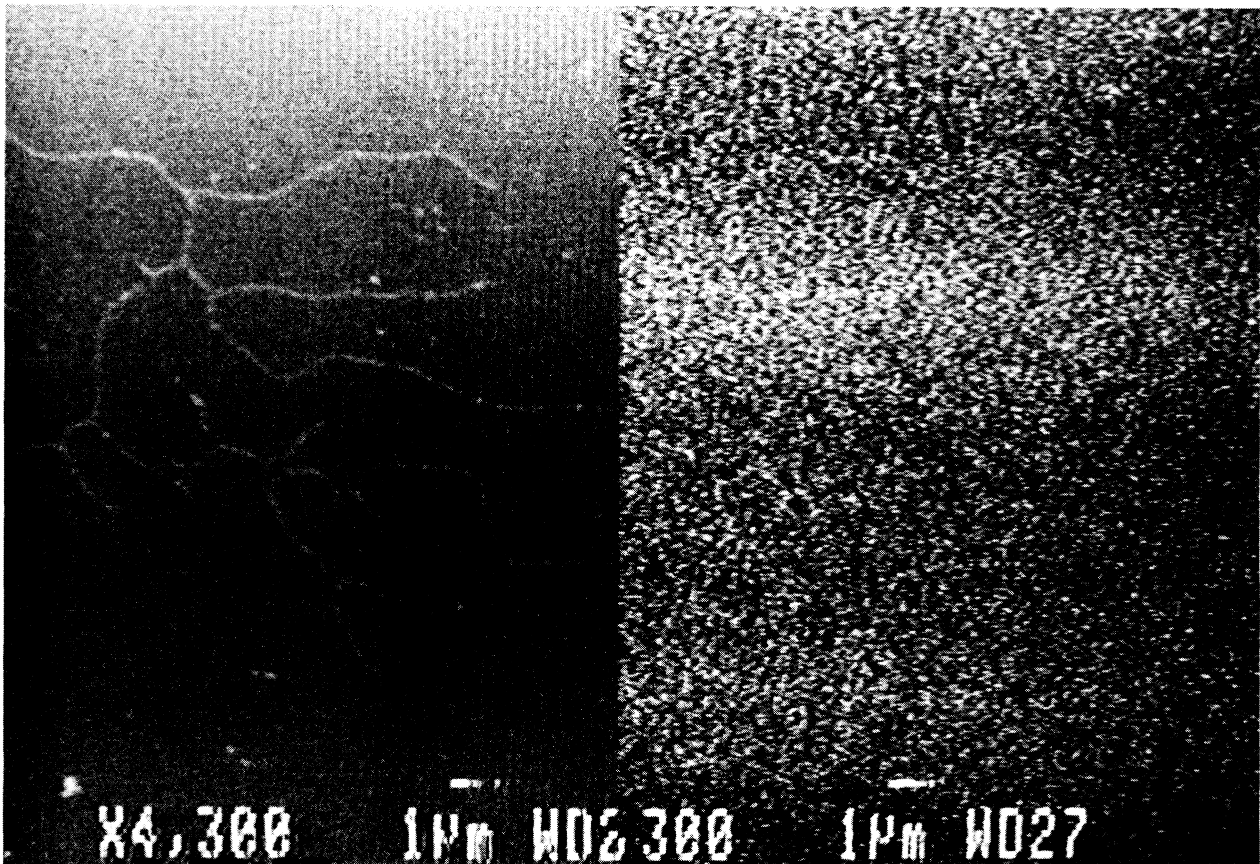
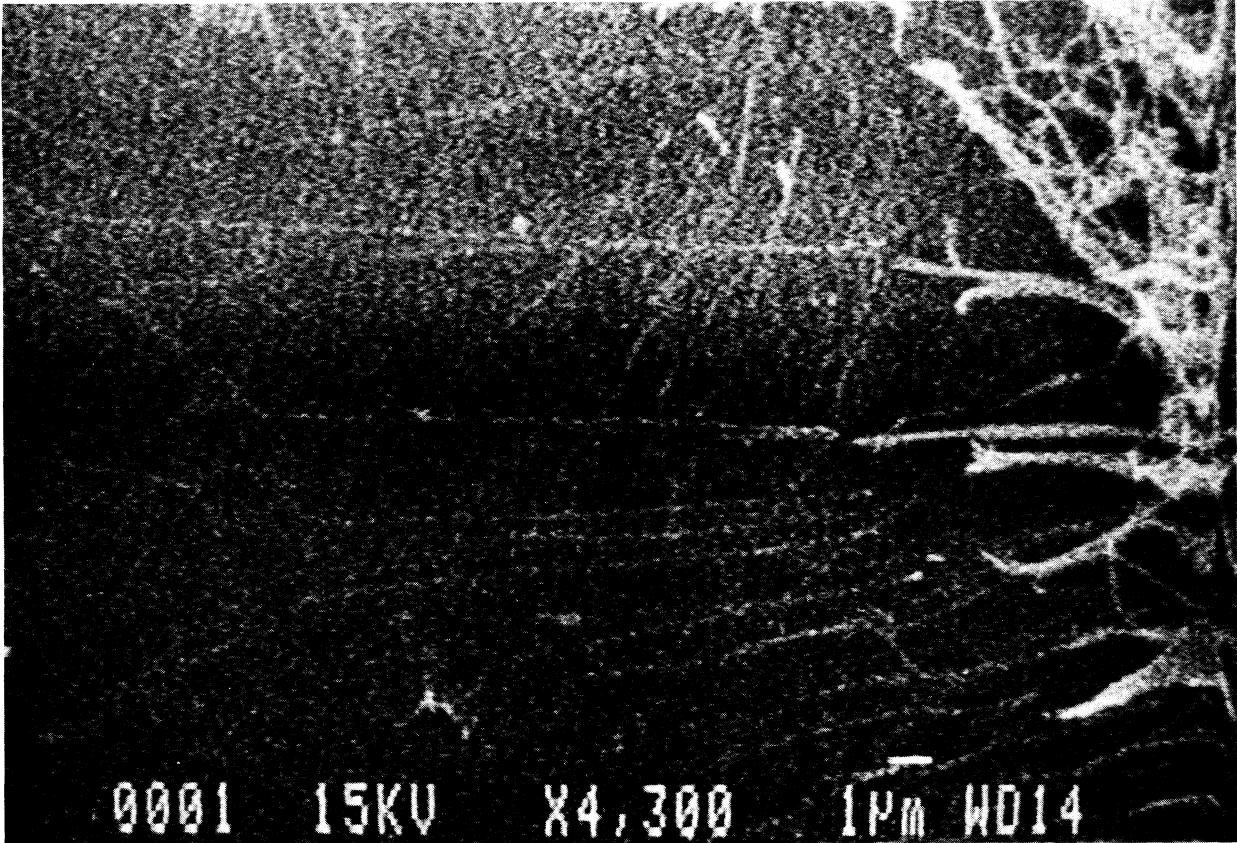
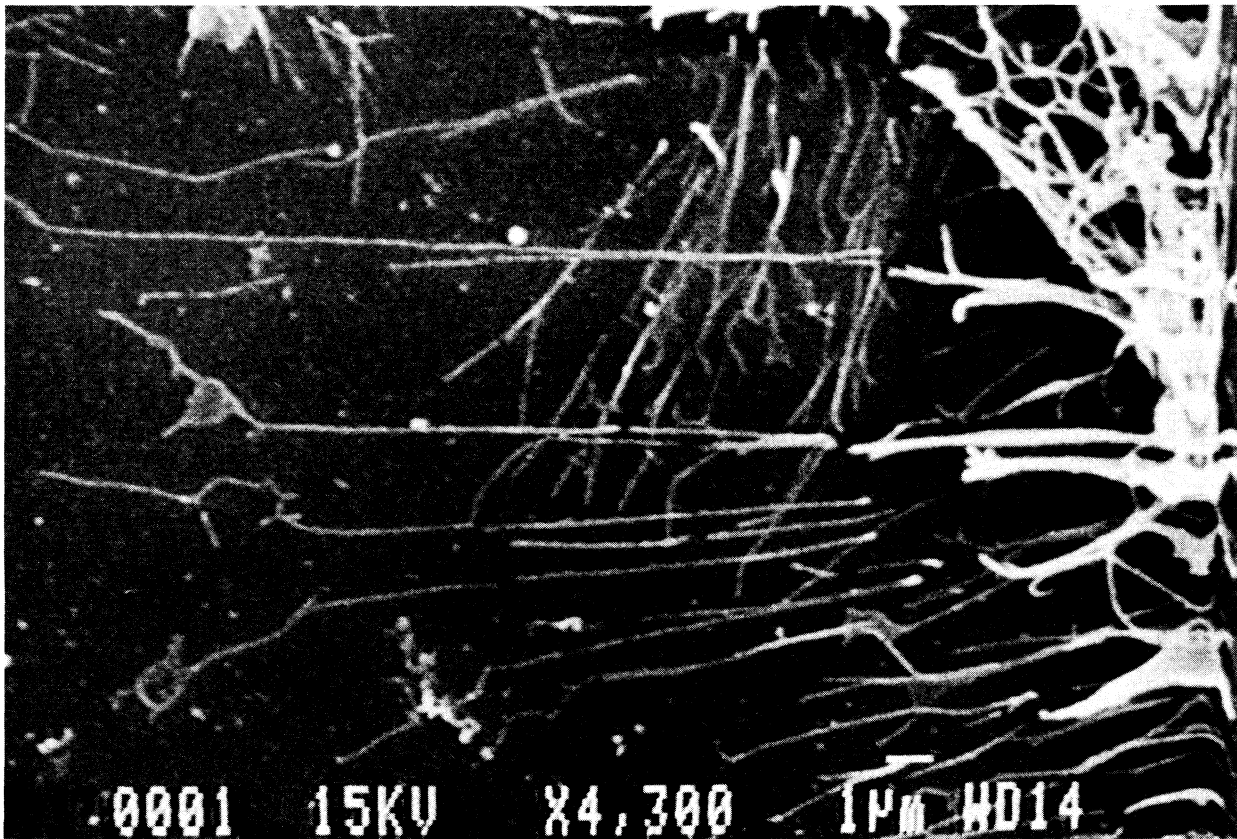
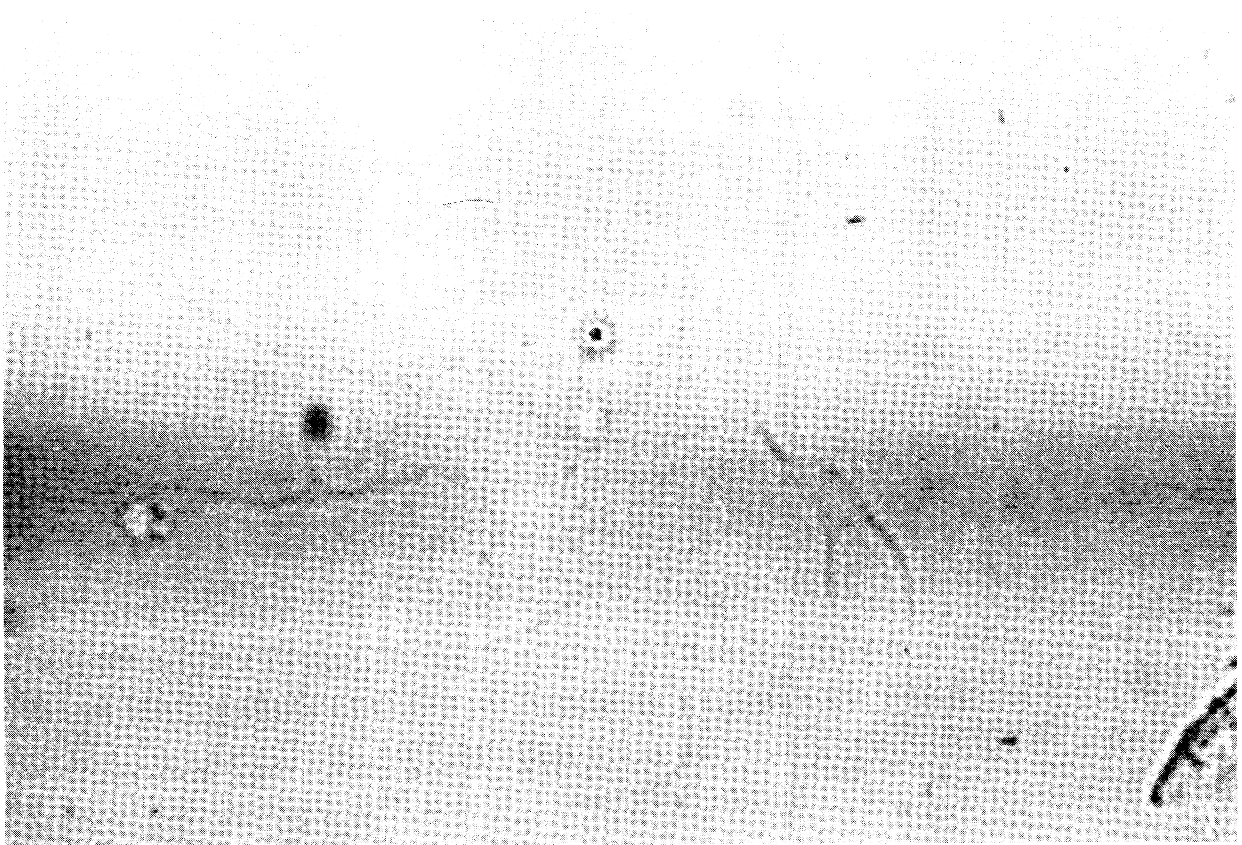
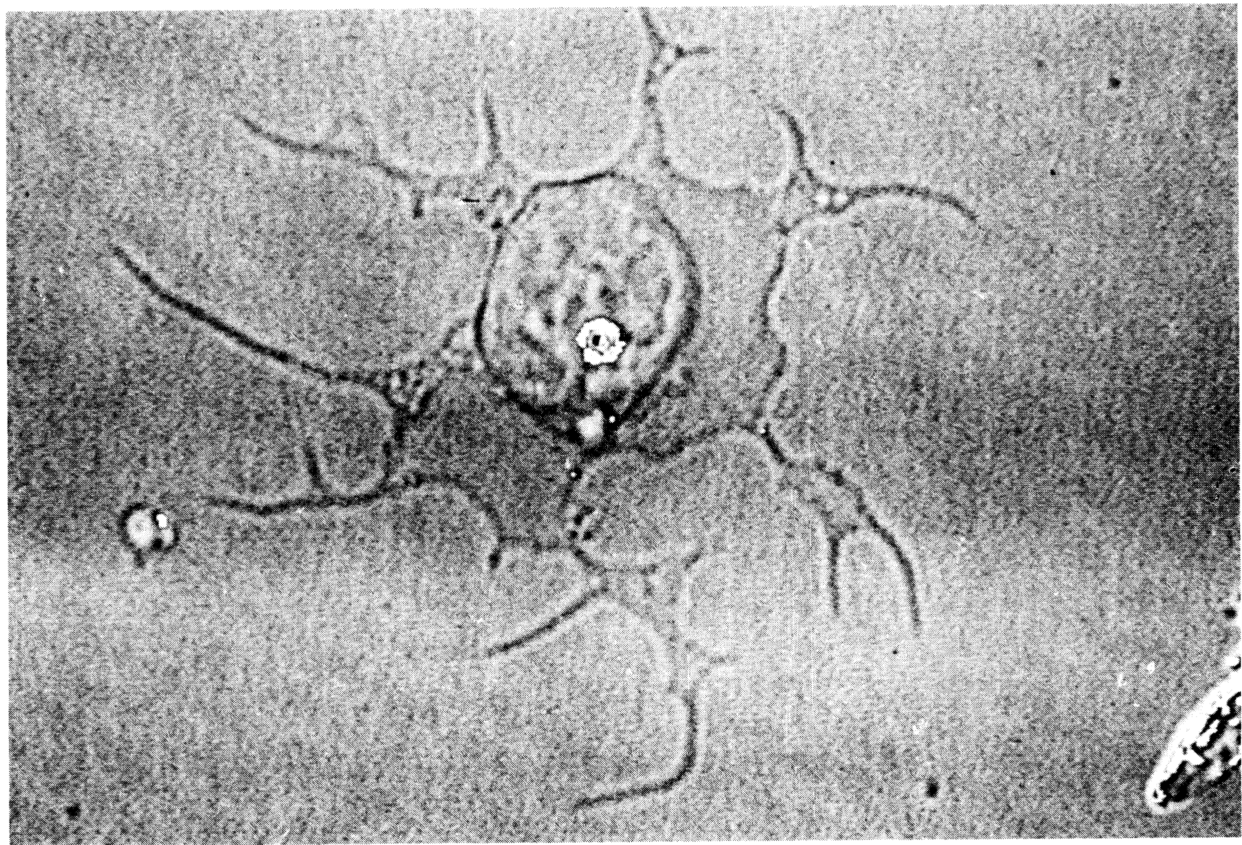



**Figure 8:** An example of image enhancement by averaging of multiple images. The image on the right side is a Scanning Electron Microscope (SEM) image before processing. The image on the left is the same micrograph after multiple image averaging, in which the inherent detector noise is greatly reduced.


0001 15KV X4,300 1μm WD14


0001 15KV X4,300 1μm WD14





Program 1

```
      INTEGER*2 COMAND(45)      ! Array storage command table
C
      DATA NCOMND/45/           ! Number of 2 char command
C
      DATA COMAND/'V0','IS','V2','SN','V4','SS','DL','LD'
      1,'LL','MT','WL','RL','HG','DH','EQ','LS','CV','C0'
      1,'C2','CS','FC','AV','SP','II','LM','VX','GR','SF'
      1,'FF','DE','MD','PW','SK','TI','HE','EX','PI'
      1,'UT','MV','ZR','PX','SL','YP','YS','CC'/
C
 5000 CONTINUE                  ! Read command from terminal
C                                 Decode for task selection
      TYPE *,'*'                ! Provide a prompt
      READ (5,5011,ERR=5580,END=5591)INSTR  ! Read Cmd
 5011 FORMAT (A2)
      DO 5013 I=1,NCOMND
      IANS = I
      IF(INSTR .EQ. COMAND(I))GOTO 5028  ! Get the index
 5013 CONTINUE
      TYPE *,'Command not in table. Type "HE" for help'
      GOTO 5000
 5028 CONTINUE                  !  Select task to SPAWN
C                                  Select Sub-menu

      IF((IANS .LT. 1) .OR. (IANS .GT. 45))GOTO 630
      GOTO (101,102,103,104,105,106,107,108,109,110
      1,      111,112,113,114,115,116,117,118,119
      1,      120,121,122,123,124,125,126,127,128
      1,      129,130,131,132,133,134,135,136,137
      1,      138,139,140,141,142,143,144,145),IANS

                  ...
                  ...
                  ...
```

Program 2

```
C
C
C       FORTRAN SUBROUTINE TO SPAWN A COMMAND LINE
C
C
        SUBROUTINE SPAWND (NAME,NBYTES)
C
C  REQUIRED DATA STATEMENTS
C
        BYTES NAME(1)             ! BYTE ARRAY FOR COMMAND LINE
        REAL MCR
        DATA MCR/6RMCR.../        ! CONVERT MCR... TO RADIX-50
C
C  SPAWN OFFSPRING TASK USING EVENT FLAG IEFN
C
        IEFN=9
        CALL SPAWN(MCR,,,IEFN,,,,NAME,NBYTES,,,,IDS)
        CALL WFLOR(IEFN)          ! WAIT IEFN BIT TO BE SET
        RETURN                    ! RETURN TO CALLER
        END
```

---

| PROGRAM * | 11/03 | 11/73 | SKYMNK | 11/780 ** |
|---|---|---|---|---|
| SIEVE (in seconds) | 5.50 | 0.75 | ---- | 0.21 |
| FLOATING-POINT ADD (in microseconds) | 246.0 | 25.2 | 10.9 | 7.8 |
| FLOATING-POINT MULT (in microseconds) | 298.0 | 31.3 | 10.9 | 11.4 |
| MULT/ADD (MATRIX) (in microseconds) | 685.9 | 53.5 | 13.2 | 15.6 |

* Programs copied from reference 3 (pg 41) and modified to run on our computers.
** 11/780 data from reference 3 (pg 38).

Table 2 - Benchmark times for various operations and systems

cost of very fast array processors exceeded our budget. We chose to trade some speed for affordable cost and purchased a SKYMNK array processor. The SKYMNK gives us performance similar to a VAX 780 as shown in table 2. In actual applications this was confirmed. The SKYMNK processor is able to compute a 2-D 512 by 512 FFT in 60 seconds, producing a result in complex numbers.

## References


[1] Dr. P.R. Smith, Dept of Cell Biology,New York University Medical Center, New York, NY 10016 (Note: MDPP is copyrighted by P.R. Smith)

[2] Chawla, S.D., Glass, L., Freiwald, S., Proctor, J.W.: Interactive Computer Graphic System for 3-D Stereoscopic Reconstruction from Serial Sections; Analysis of Metastatic Growth. *Computer Biomedic.* 12 (3) 223-232, 1982.

[3] Digital Equipment Corporation, RSX-11M/M-PLUS Executive Reference Manual, Order No. AA-L675A-TC (For RSX-11M+, version 2.0).

[4] Cohler, Jonathan: Time Trials. *Hardcopy.* 2 (2): 36-41, November 1984

# A Simple Bootstrap Prom Programmer

Frank R. Borger
Michael Reese Medical Center
Chicago, IL

Recently we installed a second disk drive controller on our PDP11 system. Since the second controller had to be installed at a non-standard address, we were unable to use standard DEC bootstrap proms. We devised some simple programs and a hardware programmer to enable us to program blank proms to fit the standard device.

The first thing we wrote was a simple program to print out the contents of the external page. (Note that the following program did not do file output. We submitted the program as a batch job, and saved the spooled batch output file to create an editable file.

```
;         PRINT EXTERNAL PAGE
;
        .MCALL  DIR$,QIOW$,EXIT$S,SVTK$S
;
;
START:  MOV     #160000,R5 ;Set the start
        SVTK$S  #SSTTAB,#1 ;Set trap vector
        MOV     #1,R2      ;for $CBOMG
        MOV     #10000,R3  ;Set loop count
REGO:   MOV     R5,R1      ;address in R1
        MOV     #MESS,R0   ;point to buffer
                           ;no zero supp.
        CALL    $CBOMG     ;convert address
        CLR     R4         ;clear flag
        MOV     (R5),R1    ;try address
        ADD     #2,R5      ;next address
        TST     R4         ;is it there?
        BEQ     EXISTS     ;branch if yes
        SOB     R3,REGO    ;go again
EXISTS: MOV     #YESCON,R0 ;point to buffer
        CALL    $CBOMG     ;convert address
        DIR$    #YESQIO    ;print it
        SOB     R3,REGO    ;go again
        EXIT$S
;
;SST SERVICE ROUTINE TABLE
;
SSTTAB: .WORD   NONEX
;
;SST SERVICE ROUTINE
;
NONEX:  INC     R4         ;show not there
        RTI
;
MESS:   .ASCII  /NNNNNN        /
YESCON: .ASCII  /NNNNNN                ;/
LEN=.-MESS
.EVEN
;
YESQIO: QIOW$ IO.WVB,5,1,,,,<MESS,LEN,40>
        .END    START
```

Since we knew from our hardware manuals that the disk bootstrap code occupied external page locations 173200 thru 173376, we edited the batch run to just contain the contents of those locations. This gave us a file of the form:

```
173200  042120          ;
173202  000042          ;
.
.
.
173376  111612          ;
```

Armed with a PDP11 programming card we then examined this output to determine what the RM03 bootstrap code was doing. We also new the following about DEC proms, (from the M9312 Technical Manual)

The first 9. words are a ROM HEADER.

| | |
|---|---|
| Word 0 | Device Name (2 Ascii characters, reversed) |
| Word 2 | Offset to second ROM HEADER (If a multi device ROM, else 0) |
| Word 4 | Power up entry - unit 0 Without CPU diagnostics |
| Word 6 | Power up entry - unit 0 With CPU diagnostics |
| Word 10 | 0 (Unit number for above) |
| Word 12 | Entry point from console emulator (Dev # in R0. |
| Word 14 | Address of device CSR |
| Word 16 | Entry point for unit >0 |
| Word 20 | BCC to diagnostic code |
| Word 24 | Must contain 173000 |
| Word 26 | Must contain 340 |
| Word 376 | 16-Bit CRC for the whole ROM (Only used by ROM diagnostics) |

After editing the output and figuring out what the bootstrap code did, we had an output as shown in figure 1. We then wrote a program to translate the octal data as it appears on the bus to 4-bit prom data. We had to take into account the following:

1. A 16-bit DEC word is stored in 4 consecutive 4-bit prom words.
2. Bits 10, 11 and 12 are inverted.

3. The following table describes the mapping information used to store the data. The contents of this table are the bit numbers, (0 thru 15) of a 16-bit DEC word.

| ROM Output Bit number | Rom Address offset 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 4 | 3 | 7 | 11 | 15 |
| 3 | 2 | 6 | 10 | 14 |
| 2 | 1 | 5 | 9 | 13 |
| 1 | 8 | 4 | 0 | 12 |

The basic program shown in figure 2 reads in the octal bootstrap program listing, and outputs the actual prom data. Outout looks like this.

```
0       0 0 0 0
1       0 1 0 1
2       1 0 0 0
3       0 1 0 1      .ascii PD
4       0 0 1 0
5       0 0 1 0
6       1 1 0 0
7       0 0 0 0     ;offset
```

We then constructed a simple hardware programmer. Programming the proms required the following:

1. Select the 4-bit byte address via the 9 address lines.
2. To program a "1", apply a 15 volt programming voltage to the appropriate data line and to the VP control line of the PROM. Only 1 data bit should be programmed at one time.
3. The programmer was designed with an address and data readout system. This enabled us to verify the corrrectness of our software by taking a known prom and applying the decoding software to generate the prom data, and then comparing the computer output with the actual prom data.

The unit consists of:

1. A Clock and strobe unit, which continuously strobes the 4 data lines to provide a "program" pulse.
2. An adjustable voltage supply, providing 12 volts when the program button is pushed.
3. 4 Data switches with appropriate programming logic.
4. A 3 digit address selection switch.
5. A 3 digit Address readout.
6. A 4-bit data readout.

Having built and testea the programmer, Hans Goebel had the honors of programming the first PROM. Although we had bought a half dozen blank proms, (expecting the worst from Murphy,) hans got it right the first time. About 1 hour was required to program the prom.

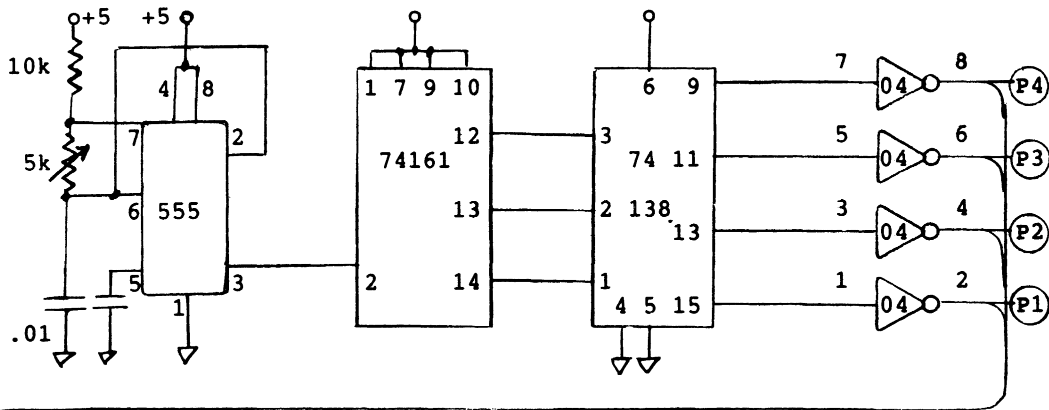
In retrospect, had we thought about it more, we probably could have programmed only a couple of bit changes to an existing prom, if we chose our alternate address correctly. Had we done that, zapping a one or two bit change could be done using a breadboard, pulse generator and variable power supply. One would just hard jumper the appropriate address and pulse the correct data line to blow the internal prom fuse.

```
173200  042120          ;.ASCII PD              RP03/RM03 BOOT
173202  000042          ;                   ;offset to next device
173204  000261          ;SEC                ;show no diags to run
173206  012700          ;MOV    #0,R0       ;clear all RMER1 flags
173210  000000          ;
173212  012701  >------>;MOV    #176714,R1  ;address of RMER1->R1
173214  176714  ^       ;
173216  010704  ^       ;MOV    PC,R4       ;point retries here
173220  103060  ^ <-----;BCC    173362      ;br if must ret to ucode
173222  000402  ^ v  <--;BR     173230      ;otherwise...
173224  173000  ^ v  v  ,.WORD  173000      ;Must be 173000
173226  000340  ^ v  v  ;.WORD  000340      ;Must be 000340
173230  010003  ^ v  -->;MOV    R0,R3       ;RMER1 flags->R3
173232  000303  ^ v     ;SWAB   R3          ;put in upper byte
173234  010311  ^ v     ;MOV    R3,(R1)
173236  012702  ^ v     ;MOV    #5,R2       ;seek, go bits for RMCS1
173240  000005  ^ v     ;
173242  000425  ^ v  <--;BR     173316      ;go start the load
173244  042102  ^ v  v  ;.ASCII BD          ;device name
173246  000132  ^ v  v  ;                   ;offset to next device
173250  000261  ^ v  v  ;SEC                ;show no diags run
173252  012700  ^ v  v  ;MOV    #0,R0       ;start with unit 0
173254  000000  ^ v  v  ;
173256  012701  ^ v  v  ;MOV    #176700,R1  ;point R1 to 1st register
173260  176700  ^ v  v  ;
173262  010704  ^ v  v  ;MOV    PC,R4       ;point retries here
173264  103036  ^ v<----;BCC    173362      ;br if no error
173266  010061  ^ v  v  ;MOV    R0,10(R1)   ;set unit bits in RMCS2
173270  000010  ^ v  v  ;
173272  012702  ^ v  v  ;MOV    #71,R2      ;read, go bits for RMCS1
173274  000071  ^ v  v  ;
173276  012711  ^ v  v  ;MOV    #21,(R1)    ;set read-in preset
173300  000021  ^ v  v  ;
173302  012761  ^ v  v  ;MOV    #14000,32(R1);set FMT,ECI bits in RMOF
173304  014000  ^ v  v  ;
173306  000032  ^ v  v  ;
173310  016161  ^ v  v  ;MOV    16(R1),16(R1);clear any DRIVE ATA bit
173312  000016  ^ v  v  ;
173314  000016  ^ v  v  ;
173316  012761  ^ v  -->;MOV    #177000,2(R1);set 1k in word count
173320  177000  ^ v     ;
173322  000002  ^ v     ;
173324  011103  ^ v     ;MOV    (R1),R3     ;get RMCS1
173326  042703  ^ V     ;BIC    #377,R3     ;clear lower byte
173330  000377  ^ v     ;
173332  050203  ^ v     ;BIS    R2,R3       ;set seek and go bits
173334  010311  ^ v     ;MOV    R3,(R1)     ;get RMCS1 in R1
173336  105711  ^ v  -->;TSTB   (R1)        ;check READY bit
173340  100376  ^ v  ^--;BPL    173336      ;wait for READY bit
173342  005711  ^ v     ;TST    (R1)        ;check SPEC COND bit
173344  100003  ^ v  <--;BPL    173354      ;br if ok
173346  000005  ^ V  v  ;RESET              ;else do unibus reset
173350  000164  ^ v  v  ;JMP    2(R4)       ;and do full retry
173352  000002  ^ v  v  ;
173354  042711  ^ v  -->;BIC    #377,(R1)   ;clear command, int enable
173356  000377  ^ v     ;
173360  005007  ^ v     ;CLR    PC
173362  000137  ^ ----->;JMP    165564      ;go back to console ucode
173364  165564  ^       ;
173366  000261  ^       ;SEC                ;show ret to console ucode
173370  012700  ^       ;MOV    #1,R0       ;with 1 in R1
173372  000001  ^       ;                   ;do full retry
173374  000706  ^<------;BR     173212
173376  111612          ;                   ;CRC word
```

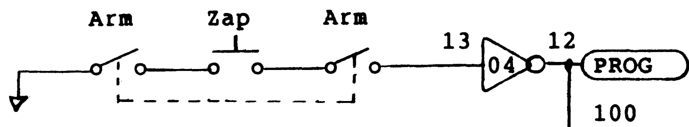
```
10 ! decode boostrap as 16 bits
11 dim a$[8],b$[6],c$[132]v,x(16),xx$[132]v
12 dim de(16) : ! Decoding bits for bootstrap bit versus rom bit
13 de(16)=3 : de(15)=2 : de(14)=1 : de(13)=8 : de(12)=7 : de(11)=6
14 de(10)=5 : de(9)=4 : de(8)=11 : de(7)=10 : de(6)=9 : de(5)=0
15 de(4)=15 : de(3)=14 : de(2)=13 : de(1)=12
16 ! de(n)=x means rom bit n = data bit x
20 open #3, "BOOTDP.DAT/RO/LN:132"
25 if end #3 then 200
30 open #4,"BOOTDP.OUT/WR/LN:132"
40 input line #3, xx$ : zz=zz+1
41 if len(xx$)>5 then 45
42 print #4, xx$
43 goto 40
45 a$=sbs$(xx$,1,8) : b$=sbs$(xx$,9,6) : c$=sbs$(xx$,15)
46 ad=oct(sbs$(a$,5,3))
47 if ad<128 goto 49
48 ad=ad-128 : goto 47
49 ad=ad*2
50 b=oct(b$)
51 for i=1 to 16
52 x(i)=b-int(b/2)*2
54 if i=11 then  let x(i)=1-x(i) : ! invert data bits 10 thru 12
55 if i=12 then  let x(i)=1-x(i) : ! (11-13 counting from 1 instead of 0)
56 if i=13 then  let x(i)=1-x(i)
57 b=int(b/2)
58 next i
60 for xx=1 to 4
61 print #4, oct$(ad+xx-1);"      ";
65 for i=1 to 4 : print #4, frmt$(x(de(21-xx*4-i)+1),1);" "; : next i
70 if xx=4 then  print #4,c$;
71 print #4
72 next xx
80 goto 40
200 close
204 print xx$
```

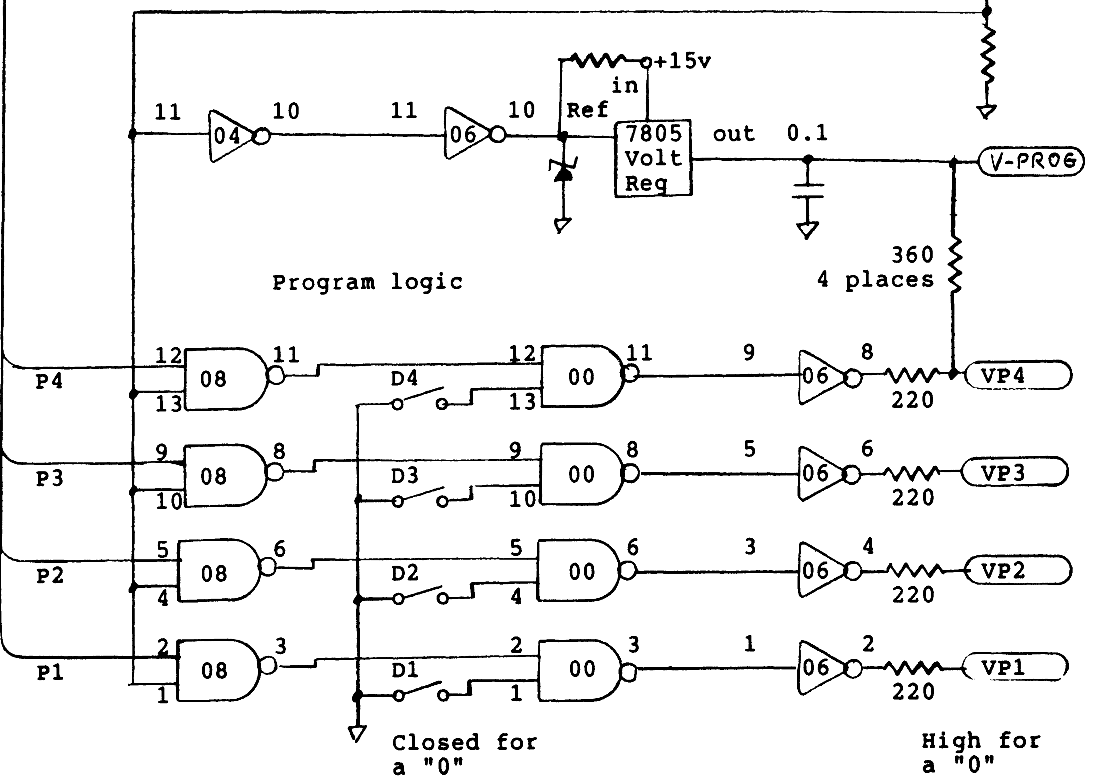
PROM PROGRAMMER

Clock and Strobe

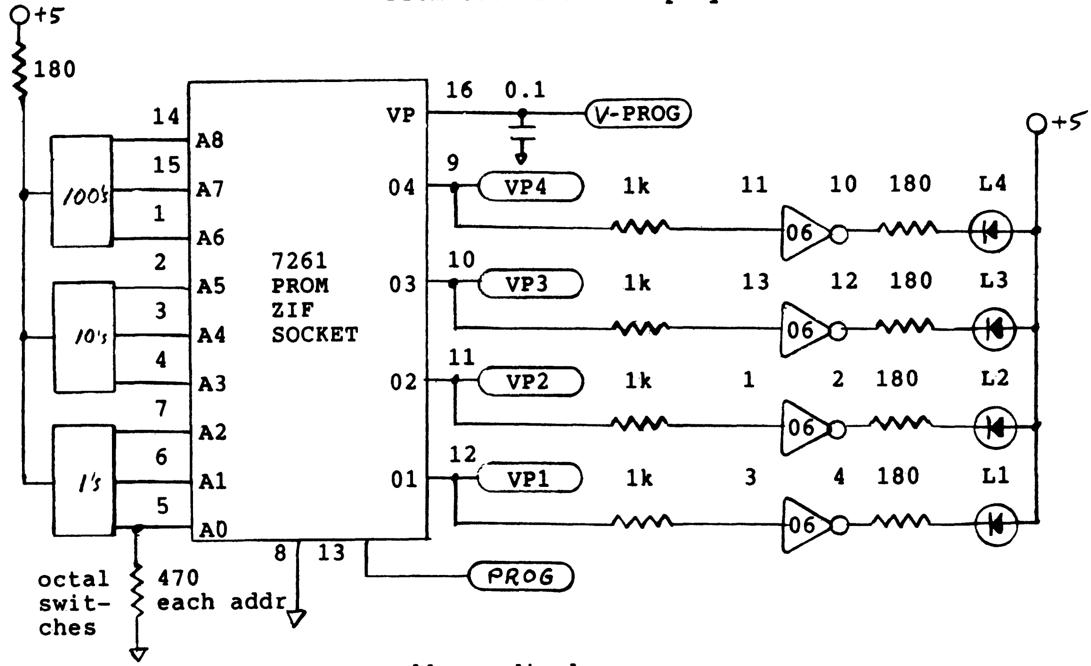



Arm and V-Prog supply

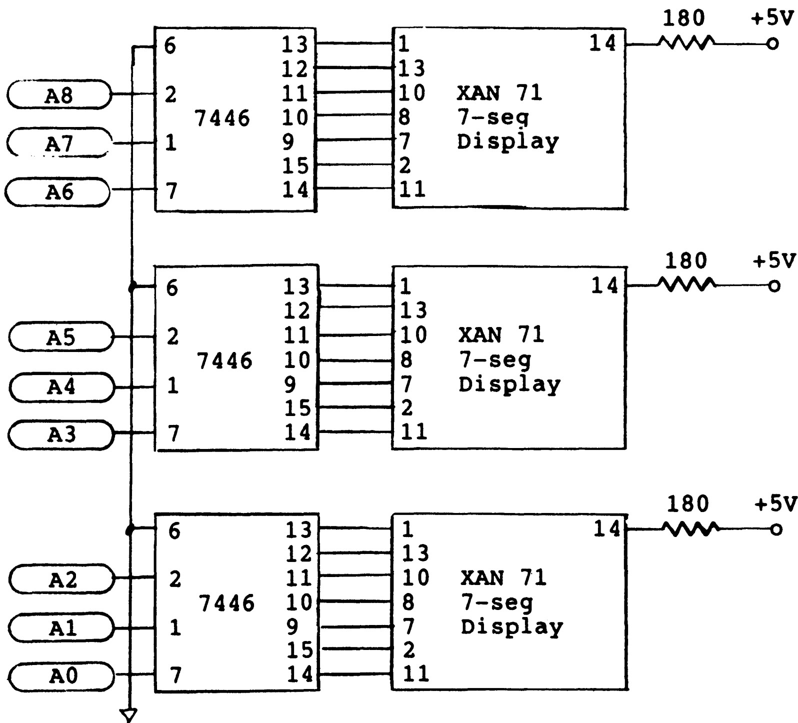


Prom socket and display




Address display



# IAS SIG

# REESE BASIC (The Other Basic)

Frank R. Borger
Michael Reese Medical Center
Chicago, IL

Everyone knows about BASIC+2 for RSX11 and IAS, but there is another BASIC available for 11's which has features that make it a valuable addition to any RSX/IAS system. It's Reese Basic. It's available from DECUS or from various SIG tapes.

It's also an Interpreter rather than a compiler. (For those who may not understand the difference between an interpreter and a compiler, the basic differences are:)

1. Instead of assembling and then task building your source program, you run the BASIC operating system and tell the system to read in your program.

2. The program always stays in memory in text form.

3. The BASIC operating system effectively does a continuous compilation of your source text into actual program operations.

Although interpreters can not be as efficient as compilers, they offer unique advantages, mainly in program development and debugging:

1. Any program statement typed in from the terminal without a line number is executed immediately. With the program stopped you can examine variables, reset loop counters, etcetera.

2. Any incorrect code can be replaced by just re-typing the offending line. The program can be re-run immediately.

3. Diagnostic halts and/or variable print-out statements may be entered easily.

4. Program execution may be continued at any line, single stepped, etc.

5. A trace mode is available, whereby any goto, computed goto statements are reported.

6. Programs that have been corrected in memory can then be saved back to disk.

Program development and debugging is incredibly faster for an interpreter. As an example, assume your basic program just bombed because you made a simple error and used A(N) instead of A(M). You wish to correct this error and test the new version. Modification time for REESE BASIC compared to BASIC PLUS 2 are:

| | | |
|---|---|---|
| REESE BASIC | 1 Type new line | |
| | 2 RUN or CON nnn | 5-10 seconds |
| BASIC PLUS 2 | 1 Edit source | 10-30 seconds |
| | 2 Compile | 20-40 seconds |
| | 3 Link | 3-5 minutes |

The above compile and link times are for an 11/44 running IAS, with the large TKB, separate work disk, etc. With a smaller CPU, smaller TKB, one disk, the difference would be even more dramatic. If you don't think you can debug programs faster with this difference in time, your programs always work perfectly the first time. (Mine don't.) But you never get something for free, the cost is that interpreters are slow. If we compare execution for a simple tight loop we find the following:

```
10 FOR I=1 TO 10000 Reese Basic 15 seconds
20 A=LOG(I)         Basic Plus 2 4 seconds
30 NEXT I
40 EXIT
```

A factor of 4 in speed seems like a very high price to pay. But nobody I know writes programs like the above simple test. If we compare the two basics using more day to day examples things even out. The following is an example of what we use Basic for a lot. We are searching a fairly large data base (1400 70-character fixed length records.) and wish to print any records containing a given 6-character string.

```
 10 dim a$[70]
 20 open #3, "POLOG.F84"
 30 open #4, "POLOG.SEL/WR"
 40 if end #3 then 100
 50 input line #3, a$
 60 if pos(a$,"NEWARK") < 1 then 50
 70 print #4, a$
 80 goto 50
100 close
110 exit
```

| | |
|---|---|
| REESE BASIC | 14 seconds<br>60% CPU usage |
| BASIC PLUS 2 | 12 seconds<br>52% CPU usage |

We initially bought BASIC PLUS 2 because we thought we would get a large speed increase over our "home made" basic. A 10% faster operation didn't really buy us much.

As a minor digression, the above example illustrates the real strength of of a basic interpreter. It is so easy to write a

quick 5 to 10 line program like the above that we do it all the time. That program was entered, saved to disk and completed execution before DEC's basic compiler finished compiling, to say nothing of TKB time.

Another thing they didn't tell us was that BASIC PLUS 2 programs are large. Incredibly large. That small program above only takes one block of storage, but the minimum BASIC PLUS 2 program task image takes up 186 blocks. I did a wild card search of our main user's disk and found just under 1700 basic programs. (That surprised me. I had no idea we had that many basic programs around.) A comparison of file storage requirements for the two basics yields:

```
REESE BASIC  (Source text only)
             1676 programs - 11K blocks

BASIC PLUS 2 (Minimum 186 blocks)
             1676 programs - 312K blocks
```

We have the equivalent of 5 RM05's on our system, and those disks would be pretty full if we kept every basic program as a B+2 task image. I guess DEC likes to sell RA80's.

While we are looking at relative sizes, lets compare memory requirements. At first glance, the 11K words of interperter code would at first seem to be a significant restriction on program size and/or variable storage. In fact, DEC's basic hauls in scads of RMS11 file access routines. This combined with the fact that (for IAS,) the interpreter code is a shared library, results in both Basics taking up about the same memory. DEC basic wins if only one basic is in memory, Reese basic wins if two or more are present:

```
REESE BASIC
     Pure code library (shared) 53500
     Minimum Program            16300

     Total for ONE program      72000
     Total for TWO programs    110300

BASIC PLUS 2
     Minimum task size          50200

     Total for ONE program      50200
     Total for TWO programs    120400
```

Likewise a comparison of maximum variable storage yields about equal variable storage capabilities, (with small program sizes.) The following program was executed by incrementing the size of the dimensioned variable until an error occurred. Results for Reese Basic versus DEC basic follow:

```
10 dim a(64,102)
20 print "Max =";64*102;". variables"
30 print "    =";64*102*4;". bytes"
40 exit

MCR>bas sizetest/rn  (Reese Basic)
Max =  6528 . variables
    = 26112 . bytes
MCR>run sizetest$   (DEC B+2)
Max =  6720 . variables
    = 26880 . bytes
```

Lets look at some more of the differences between the two basics. The first thing that is totally different is the file open command. Early in the development of Reese basic it was decided to not use the standard basic "OPEN" command syntax, but to use a format compatible with the standard DEC file name parser, including various access mode switches. Comparing the two types of open statements we see:

```
  REESE BASIC
20 OPEN #3, "POLOG.F84"
25 OPEN #4, "POLOG.SEL/WR"

  BASIC PLUS 2
20 OPEN 'POLOG.F84' FOR INPUT AS FILE #3
25 OPEN 'POLOG.SEL' FOR OUTPUT AS FILE #4
```

Reese Basic file specifications are essentially just String variables, but with several 2-character switches that provide the full range of files11 access modes:

| | | | |
|---|---|---|---|
| /FX | Fixed Length | /RO | Read Only |
| /RN | Random Access | /WR | Write Access |
| /LN:n | Length = n | /UP | Update Access |
| /EN:n | goto n on EOF | /MO | Modify Access |
| /BN | Binary file | /AP | Append to file |
| /BL | Virtual Array | /SH | Shared Access |

Note that since file names are string variables, one has great capability for name parsing. For example, it is very easy to check for a file extension and supply a default one if one is not there:

```
10 dim na$(25)v
20 input "Name of Input File ";na$
30 if pos(na$,".")<1 then na$=na$+".DAT"
40 open  3, na$+"/RO/SH"
```

All calculations are done using 2-word floating point format, (and in fact require a Floating Point unit, or use of a floating point emulator.) Variables are stored however in several different formats, both to save space and to provide for better mapping to virtual files. The following types are supported:

```
     A&             8-bit Integer
     B1%            16-bit Integer
     CC             32-bit floating point
 DIM A[n,n]         Arrays may be 2 dim.
 DIM A$[n]          Fixed Length String
 DIM A$[n]V         Variable Length String
 DIM A$[n]V(x,y)    Dimensioned String
```

Reese basic also supports virtual array storage. Special forms of the DIM statement and the OPEN statement are used. These allow one to directly access any file. The following example shows how to access any word of a 400 block file. (Note that for virtual arrays, the first unit of any array is N(0) ).

```
120 dim #4,DD%(399,255)
.
204 open #4,"[11,17]TEST.DAT/BL/MO"
.
300  ! Print octal value
305 print oct$(dd%(bl,by))
```

The above example prints the contents of offset by into block bl of file test.dat.

Reese Basic has a full set of standard string functions, (one of the strong points of any basic,) with some added ones that make a programmers life easier.

```
INX/POS  Position of substring
LEN      Length of a string
SEG$     Substring from main string
SBS$     Substring from main string
RIGHT    Right-most N characters
LEFT     Left-most N characters
MID      Same as SBS$
PIECE$   Substring between dividers
LTR$     Leading blank trim
TRM$     Trailing blank trim
RJS$     Right justify string
LJS$     Left justify string
SPACE$   String of n spaces
STRING$  String of n characters
```

It has a large set of internal and I/O conversion routines:

```
VAL      String to numeric conversion
OCT      String to octal conversion
AR5      Ascii to RAD50 conversion
ASC      Ascii character to numeric
CHR$     Numeric to ascii character
STR$     Numeric to string conversion
OCT$     Octal to Ascii (unsigned)
OCS$     Octal to Ascii (signed)
R5A$     Rad50 to Ascii conversion.
FRMT$    Print using, FORTRAN style
TAB      Tabulate to position n
```

Some very nice DEC system based functions

```
NRC     Number of records in file
ERR     Number of last error
ERL     Line number at last error
FCS     Last FCS related error number
COR     Free space available in bytes
DAT$    MM/DD/YY from day-of-century
DDAT$   DD-MMM-YY from day-of-century
TIM$    Time-of-day from seconds
DCEN    day-of-century from date string
SEC     seconds from time-of-day
```

And some non standard commands also tailored to DEC system:

```
STEP          Single step program
CON           Continue after stop or error
ON ERROR GOTO Transfer if error occurs

BREAK      Print using write-pass-all
SET PROMPT Turn on/off question mark
SET TRACE  Turn trace on/off
SET UPPER  Turn case conversion on/off
SET READ-PASS-ALL
SET WRITE-PASS-AL

SLEEP n units  Do mark-time
WAIT n units   Terminal read with time-out
TRACE on/off   Turn trace on/off
```

Reese basic contains a simple but effective method for doing various types of question processing. Options include single and multiple line questions, single and multiple choice answers, and automatic linking to further questions based upon question answers. The text of the questions themselves is contained in a special macro source file. The command:

```
OPEN LIBRARY #N, FILENAME
```

Opens a macro question source file containing question text. Questions are of three basic types:

```
.MACRO   MULTXT  0
This is an example of a
multiple line question.
.ENDM

.MACRO  SINGLE  1
Type your name, (last,first)
.ENDM

.MACRO  MULCHO  2
Select your state of mind
GOOD\MULCH1
FAIR\MULCH2
TGIF\MULCH3
.ENDM
```

Various routines are available to automatically output these questions and return ascii (or numeric for multiple choice questions) answers. The basic question display routines are as follows:

Three basic forms of question processing calls exist:

```
x=M  Reference question by NAME
x=C  Reference by link to previous question
x=R  Reference by last question accessed

CALL "xDIS"(LUN,STAT,NAME)
CALL "xPOS"(LUN,STAT,NAME)
CALL "xQTXT"(LUN,STAT,NAME,STRING)
CALL "xATXT"(LUN,STAT,NAME,STRING)
CALL "xQNAM"(LUN,STAT,NAME,STRING)
```

The above question handling routines are just one of a series of machine language subroutines that can be loaded on command into the program area of the basic interpreter. Although the mechanism for passing arguments from basic to the machine language routine is somewhat complicated, (since these routines must do what TKB does when it links a program,) machine language routines are just as flexible as they are with DEC basic, Fortran, etc. Machine routines are handled by three special commands:

```
LOAD "PROGNA"        Loads macro routine
UNLOAD               Unloads all routines
CALL "PROGNA"(...)   Calls macro routine
```

Some of the more noteworthy loadable routines include:

```
SPAWN   spawns an MCR command
BINSRC  Fast binary search
DIRECT  Issue any system directive
EXITST  Exit with status
INSTAL  Install a task
LOWCAS  Convert ascii to lower case
REVSTR  gnirts txet a esrever
UPPCAS  Convert ascii to upper case
```

The spawn directive has proved to be a very powerful one. With it, BASIC has the capability of acting as a alternative to MCR. Many of our less sophisticated users interact with a menu program written in BASIC, and never see or need to use MCR or DCL. The combination of math capabilities and string manipulation with spawning capability provides a much greater capability than batch, indirect MCR or DCL.

# A MULTI-TERMINAL TASK

Ted Smith
Division of Medical Physics
Department of Radiation Therapy
Hospital of the University of Pennsylvania
Philadelphia, PA 19104

**ABSTRACT**

This is a discussion of an application task which collects and displays data on seven or more terminals. Methods for maintaining adequate response time while simultaneously processing different requests are demonstrated by an analysis of a patient information system used in our department. Features of the IAS terminal handler and facilities of the IAS executive that support the multi-terminal task are described. Specific examples from our "treatment area status" task illustrate these features.

## Introduction

Some applications require a facility to efficiently process concurrent requests from various users in an asynchronous order. The IAS Version 3.2B operating system provides the mechanisms needed to create an ideal environment for multi-terminal tasks. The design and implementation of a multi-terminal task is illustrated by an analysis of the Radiation Therapy Department's "Patient Tracking System". The tracking system replaces an intercom system which follows the patient's progress through the department.

The Department of Radiation Therapy provides many services at various locations within the department. Outpatients arrive and await scheduled services in the reception area. During a course of treatment, the patient's status may alternate between outpatient or inpatient depending on the condition of the patient. Frequently, patients are scheduled for services at mulitple locations. An intercom system had been used to connect the service areas. When a patient completed a service, the technician "buzzed" the reception area for the next patient.

Although functional and simple to operate, the intercom system lacked the pliability to ascertain the progress of patients through the department. Factors affecting the technicians' ability to provide services efficiently, required a tedious survey of the reception and service locations. Some prevailing factors are:

- Who is waiting?
- Which patients are available?
- Where is the patient?

Each day, the reception desk received a treatment schedule of patients and services. As patients arrived, the receptionist marked the treatment schedule. Service locations must habitually "buzz" the reception desk to obtain the names of waiting patients. Patients scheduled for multiple services became unavailable to other services when selected by a service location. Upon completion of a service, the technician must know if the patient has any remaining appointments to properly direct him either to the reception area or send the patient home. Locating a patient required the technician to "buzz" the reception and service locations scheduled until the patient was found.

## Objectives of "Patient Tracking System"

The primary objective of the "Patient Tracking System" is to follow the patient's progress through the department. A description of each patient is maintained and provided to the scheduled service locations (Table 1). When a patient is selected for a service, his description is updated to reflect his new location and he becomes unavailable for his other scheduled services. After the service is provided, information about the service is collected and the patient becomes available for the other services scheduled.

**Table 1. Patient Description**

- o Name
- o Status (Inpatient, Outpatient)
- o Availability for service
- o Location
- o Staff physician
- o Type of service scheduled
- o Time scheduled
- o Time patient arrived in dept

Other objectives include the collection of service information, a message utility and the ability to track unscheduled services. The collection of information about each service when provided allows the system to automatically generate billing charge information. The patient's file is also updated providing staff with immediate access to the progress of the patient. When an available patient is selected for a service, the system sends a message to the patient's current location. The message contains a request for the patient to be sent to the selected service location. Users may send multi-line free text messages to one or more service locations. Special instructions or additional information about a patient is transmitted using this message facility.

## Design of the "Patient Tracking System"

The tracking system consists of three tasks to interface with the terminals, lookup patient schedule information and to update the patient's file with completed service information (Figure 1). Each task maps to an installed region providing concurrent shared access to the system data structures. The IAS Send/Receive message facility is used to provide communication links between the three tasks.

The terminal interface task, TRKHUP, maintains the system data structures in the installed region and controls the flow of information within the system. This is a menu driven terminal interface providing the user with a simple command structure to display and collect patient data. A VT220 terminal is installed at each service location. Each function key on the VT220 transmits a unique escape sequence allowing an application to assign and interpret single keystroke commands. The physical terminal characteristics are set to "VT220" with "7 bit controls". These features enable the use of the entire keyboard and proper interpretation of ANSI 7 bit character codes.

Normally, the "escape" character, ESC {1}, is considered an input terminator. Thus, two or more QIO$ reads [1] are necessary to acquire the escape sequence (e.g. ESC "[28~" for the HELP function key). The first read will contain the escape sequence inducer ESC and a second read with a timeout of zero will contain the escape sequence "[28~". The IAS terminal handler was reconfigured to allow "escape sequence support" [2] enabling the recognition of escape sequences as the input terminator. Now, the entire escape sequence, ESC "[28~" is captured using a single read. Greater throughput with reduced operating system overhead was accomplished since most commands require pressing only a single function key.

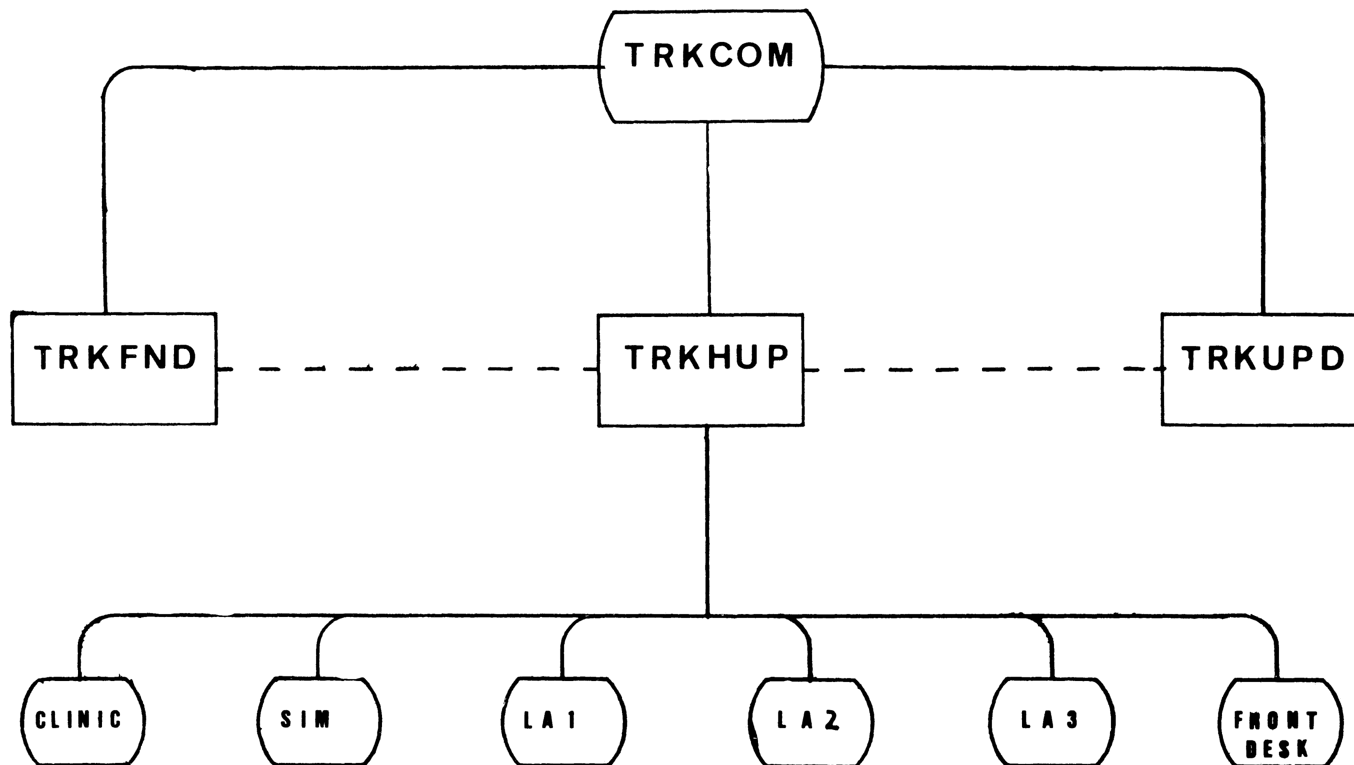



Figure 1. "Patient Tracking System"

The terminals used in the tracking system are setup as devices instead of terminals by IAS {2}. This was done to give the tracking system complete control over the terminals. Users are now prevented from interrupting the system by pressing <Control/C> {3}. Messages from terminals outside the system are inhibited because the IAS MESSAGE utility [3] can only transmit messages among terminals and not to devices. Each terminal is assigned a unique I/O buffer, local event flag [4] and logical unit number by the terminal interface. When an input terminator is detected, the QIO$ sets the corresponding local event flag. The command is then extracted from the associated input buffer for processing and the event flag is cleared. When all input is processed, the terminal interface issues a "Wait for Logical Or of Flags" [5] to suspend itself until an event flag is set. Processor time is saved by not performing a round robin test of event flags when the interface is idle.

The patient lookup task, TRKFND, searches our patient database [6] for the patient and all scheduling information for today. A patient may be located using either name, social security number or medical record number {4}. TRKFND and TRKHUP communicate using the Send/Receive with AST [7,8] intertask message facility. An AST trap occurs in TRKFND whenever a message is received. The AST trap routine sets a predetermined local event flag enabling TRKFND to run. TRKFND then receives the message containing the operation to be performed for TRKHUP. When completed, TRKFND sends the results back to TRKHUP using the same mechanism described above. When all message are processed, TRKFND issues a "Wait for Single Event Flag" [9] to suspend itself until another message is sent and the AST trap again sets the event flag.

If a patient cannot be uniquely identified using the information supplied by the user, TRKFND will collect those patients with similar information and a unique identifying database key {5}. The list of patients is then sent to TRKHUP. The user will be prompted to select the actual patient from this list, continue searching for more matches or abort the search. If a patient is selected, TRKHUP returns the unique key to TRKFND and the associated patient and schedule is extracted.

The updating task, TRKUPD, adds to the patient's file all information about the provided service. Soon TRKUPD will generate the billing charge information based on the service information entered by the user. Communication between the TRKHUP and TRKUPD tasks is accomplished using the facility for Send/Receive messages. When a service is completed, the user is asked to provide information describing the service. Also, the user selects the type of charge to be generated (e.g. Simple, Intermediate or Complex). The information collected is then sent to TRKUPD. Upon receiving a message, TRKUPD updates the patient database to contain the collected service information. The billing charge is added to a charge file for later processing. At the end of each day, the billing charge file is transmitted to Central Data Processing by tape and the charges are then posted to the patient's account. When not processing messages, TRKUPD suspends itself by issuing a "Wait for Single Event Flag" [9].

**Send/Receive Messages**

The Send/Receive messages used by the tracking system have a standard format consisting of a fixed message header and a variable length message. The message header (Table 2) is always the first ten words of the message. The message is limited to 250 bytes including the header. When a message is sent, an AST trap occurs in the receiving task which sets a local event flag is set informing the receiving task about the message. The terminal interface, TRKHUP, uses event flags 23 and 24 for messages from TRKUPD and TRKFND respectively. Both TRKUPD and TRKFND use event flag 24 indicating a message from TRKHUP.

**Table 2. Send/Receive Message Header**

| Location (Words) | Length (Words) | Description |
|---|---|---|
| 1 | 1 | Function to be performed (e.g. Lookup patient by name) |
| 2 | 1 | Service location making request (Address of Corresponding Post record) |
| 3 - 4 | 2 | Hospital assigned to Post |
| 5 - 6 | 2 | Last patient found by primary search |
| 7 - 8 | 2 | Last patient found by secondary search |
| 9 | 1 | Not used |
| 10 | 1 | Lenght of message (in bytes) |

**Terminal State**

The terminal interface is required to allow the independent operation of terminals while concurrently processing different requests made in an asynchronuous order. To provide the needed flexibility, the state of each terminal is maintained. Terminal state consists of the unique characteristics of the terminal (Table 3) and the current display menu level being used by the terminal. Prior to servicing a terminal, the terminal's state characteristics are fetched from the installed region. The command is fetched from the terminal's input buffer and tested for validity. The command is processed only if valid for the current menu. Otherwise, an appropriate error message is displayed.

**Table 3. Terminal Characteristics**

- o Name of Location
- o Address of input buffer
- o Address of mail buffer
- o Logical Unit Number
- o Terminal unit number (TTnn:)
- o Event flag number
- o Terminal type (e.g. Service)
- o Services performed at location
- o Current menu level
- o Data pointers

**I/O buffering and Event Flags**

Special considerations for the handling of terminals are needed when a single task controls more than one terminal (Table 4). A task controlling multiple terminals must have the ability to accept input from one or more terminals while processing completed input from another terminal. Input entered at different terminals must be kept separate and a method determining which terminals have completed input must be developed. The FORTRAN READ statement causes the task to suspend and wait until the input is terminated. This prevented TRKHUP from performing simultaneous reads from different terminals or processing another terminal's terminated input.

Each terminal is allocated to TRKHUP using a FORTRAN OPEN statement. A unique 80 byte input buffer and local event flag is assigned to each terminal. Instead of using the FORTRAN READ statements, an assembly language macro was written to issue a QIO$ read to each terminal specifying the local event flag to be set when input was terminated. After input is terminated, TRKHUP extracts the command from the corresponding input buffer for processing and issues another QIO$ read to the terminal.

**Table 4. Comparsion of input processing between single and multi-terminal tasks**

| Single Terminal | Multi-terminal |
|---|---|
| o One input buffer | o One buffer per terminal |
| o FORTRAN READ sufficient (equivalent to QIOW$) | o QIO$ macro with event flag |
| o Prccess input | o One event flag per Terminal |
| | o Which terminal completed |
| | o Lookup terminal state |
| | o Select input buffer |
| | o Process input |

**Command Processing**

The terminal interface defines many of the function keys on the VT220 keyboard. The input terminator informs TRKHUP which function key was pressed or how the input was terminated. Currently, three groups of input terminators exist: Normal, Timeout and Escape sequences. Normal input termination consists of commands enter by the user and terminated with a carriage return, line feed or form feed. Timeout occurs when a read fails to complete in the specified time (3 minutes). Upon timeout, the terminal is reset to the main menu. Escape sequence termination occurs when the user presses a function key. An optional command may also precede the function key.

The command fetched from an input buffer is parsed into the command line and input terminator. The input terminator is then converted into a command code (Table 5). The command line is used to modify the defined operation of the command code. For instance, a user enters a "5" and presses the <Down Arrow>. The command line will contain the "5" and command code is 1. The operation is modified to move the cursor five lines down instead of only one. This modifying feature allowed the user to "jump" to the desired location with a single command.

**Message Facility**

A message facility is provided by the terminal interface allowing users to send free format text messages to one or more other users within the tracking system. The facility is also used by the system to inform the receptionist when a patient is selected for a service. The message facility provided a more favorable mechanism of communication than the intercom system.

Messages can contain up to four lines of text and may be sent or received from any menu level. A mail buffer, like the input buffer, is assigned to each terminal. Each line of the message is transferred from the input buffer into the message buffer as entered by the user. This process is necessary for two reasons: the input buffer is too small to contain multiple lines of text and is always initialized with spaces before the next QIO$ read is issued at the terminal. After the message is completed, the contents of the mail buffer are transferred to the installed region and the message address is placed in the mail queue of all the receiving terminals. At each receiving terminal, the "You have mail." message is displayed and highlighted. The receiver decides when to read the message. This allows the reciever to complete a function without interruption. The message is erased and the space deallocated when the last receiver reads the message.

Table 5. Input Command Summary [11]

| Code | Command | Definition | Code | Command | Definition |
|---|---|---|---|---|---|
| -1 | | Timeout | 22 | ESC [23~ | <F 11> |
| 0 | Normal | <CR> <LF> <FF> | 23 | ESC [24~ | <F 12> |
| 1 | ESC [B | <Down Arrow> | 24 | ESC [25~ | <F 13> |
| 2 | ESC [A | <Up Arrow> | 25 | ESC [26~ | <F 14> |
| 3 | ESC [D | <Arrow Left> | 26 | ESC [31~ | <F 17> |
| 4 | ESC [C | <Arrow Right> | 27 | ESC [32~ | <F 18> |
| 5 | ESC [1~ | <Find> | 28 | ESC [33~ | <F 19> |
| 6 | ESC [2~ | <Insert Here> | 29 | ESC [34~ | <F 20> |
| 7 | ESC [3~ | <Remove> | 30 | ESC Op | <Keypad 0> |
| 8 | ESC [4~ | <Select> | 31 | ESC Oq | <Keypad 1> |
| 9 | ESC [5~ | <Prev Screen> | 32 | ESC Or | <Keypad 2> |
| 10 | ESC [6~ | <Next Screen> | 33 | ESC Os | <Keypad 3> |
| 11 | ESC [28~ | <Help> | 34 | ESC Ot | <Keypad 4> |
| 12 | ESC [29~ | <Do> | 35 | ESC Ou | <Keypad 5> |
| 13 | ESC OP | <PF 1> | 36 | ESC Ov | <Keypad 6> |
| 14 | ESC OQ | <PF 2> | 37 | ESC Ow | <Keypad 7> |
| 15 | ESC OR | <PF 3> | 38 | ESC Ox | <Keypad 8> |
| 16 | ESC OS | <PF 4> | 39 | ESC Oy | <Keypad 9> |
| 17 | ESC [17~ | <F 6> | 40 | ESC OM | <Keypad Enter> |
| 18 | ESC [18~ | <F 7> | 41 | ESC On | <Keypad "."> |
| 19 | ESC [19~ | <F 8> | 42 | ESC Ol | <Keypad ","> |
| 20 | ESC [20~ | <F 9> | 43 | ESC Om | <Keypad "-"> |
| 21 | ESC [21~ | <F 10> | | | |

### Installed Region and Data Structures

The installed region [10], TRKCOM, is an area in main memory used as a control and communications common area for the "Patient Tracking System". The region is eight "Kw" {6} in size and is shared by the three tracking system tasks. The area is divided into 511 nodes of sixteen words. These nodes are dynamically allocated and deallocated to form one of five logical records: Post, Free, Rumor, Case and Task. All records are formed using contiguous nodes. Therefore, a record address is the address of the first node allocated to the record. Pointers within each record contain the address of related records providing fast access without the need to search the entire region for the desired record. These pointers are grouped together forming set relationships. In addition to the node pool described, the communications area also contains 16 one word registers for control and fast lookup of selected records.

### Post Records

One Post record is allocated for each terminal in the tracking system and is composed of six nodes containing the terminal characteristics (Table 6a) and pointers for set relationships with other records associated to the Post. Four sets are maintained within the Post record: Location, Service, Mail and Dispatch sets. The Location set connects all the Post records in the installed region. Location of another terminal's characteristics when sending messages is the primary function of this set. The Service set connects the Post to all of the services (Task records) scheduled for the Post. The Mail set contains the address of the messages (Rumor records) sent to the Post. The Dispatch set contains the address of all the patients (Case records) being treated or waiting at the Post.

**Table 6a. Contents of a Post record:**

| Location (Bytes) | Length (Bytes) | Description |
|---|---|---|
| 1 - 2 | 2 | Record type ID (Always 1) |
| 3 - 4 | 2 | Next Post in Location set |
| 5 - 6 | 2 | Prior Post in Location set |
| 7 - 8 | 2 | First Task in Service set |
| 9 - 10 | 2 | Last Task in Service set |
| 11 - 12 | 2 | First Message in Mail set |
| 13 - 14 | 2 | Terminal unit number (TTnn:) |
| 15 - 16 | 2 | Assigned LUN (channel) |
| 17 | 1 | Post Type (Service, Reception) |
| 18 | 1 | Status flags |
| 19 - 28 | 10 | Post Name (e.g. Clinic) |
| 29 - 30 | 2 | Services performed at Post |
| 31 - 32 | 2 | First Case in Dispatch set |
| 33 - 192 | 160 | Mail buffer |

**Table 6b. Contents of a Free record:**

| Location (Bytes) | Length (Bytes) | Description |
|---|---|---|
| 1 - 2 | 2 | Record type ID (Always 2) |
| 3 - 4 | 2 | Next Free record in Vacant set |
| 5 - 6 | 2 | Length of Free record (in nodes) |

**Table 6c. Contents of a Rumor record:**

| Location (Bytes) | Length (Bytes) | Description |
|---|---|---|
| 1 - 2 | 2 | Record type ID (Always 3) |
| 3 - 4 | 2 | Next Rumor record in Mail set |
| 5 - 6 | 2 | Sendor's Post address |
| 7 - 8 | 2 | Length of Rumor record (in nodes) |
| 9 - 10 | 2 | Length of Message (in bytes) |
| 11 - nn | nn - 10 | Message |

### Free Records

Free records mark unallocated nodes in the area and have a variable number of nodes assigned (Table 6b). A Vacant set is used to connect all Free records in the region in ascending order by record address. Adjacent Free records are always appended to form a single contiguous Free record. The tracking system uses a "first fit" algorithm when allocating a record (e.g. Rumor). The Vacant set is scanned until the first Free record is found containing at least enough nodes to create the record contiguously. Any remaining nodes are used to create a new Free record. When a record is deallocated (e.g. message read), one of four operations is performed based on the type of records adjacent to the deallocated record:

- If neither of the adjacent records are Free records, then a new Free record is created.
- If only the previous record is a Free record, then the deallocated record is appended to the Free record.
- If only the next record is a Free record, then a new Free record is created containing both the deallocated record and the next Free record.
- If both of the adjacent records are Free records, then a new Free record is created containing the deallocated record and both of the adjacent Free records.

The benefit of the "first fit" and deallocation procedures insure the formation of the largest possible contiguous Free record.

### Rumor Records

Rumor records are used to transmit messages between Posts and use a variable number of nodes depending on the length of the message (Table 6c). The message is built from both the Post's mail buffer and input buffer. This restricts the length of the message to 240 bytes (160 byte mail buffer and 80 byte input buffer) or, four lines. Line terminators are included in the buffer. If the message fills the mail buffer, then one additional line may be entered into the input buffer after which the message is considered complete. A Rumor record containing the message is created for each receiving Post. The Rumor record address is added to the Mail set of each recieving Post. When completed the sending Post's mail and input buffers are erased.

### Case Records

Case records contain the demographic information about a patient currently in the tracking system (Table 6d). Only one case record is created for each patient who is currently a part of the system. Case records require three nodes and contain pointers for three sets: Dispatch, Patient and Schedule. The Dispatch set contains all the patients currently at a Post. When a patient moves to another Post, he is removed from the current Post's Dispatch set and placed in the new Post's Dispatch set. A Patient set connects all the patients in the region. The Schedule set connects the patient to all of his scheduled services (Task records).

### Task Records

Task records contains specific information about each service scheduled for the patient (Table 6e). For each service the patient is receiving, one Task record is created. A Task record requires one node and has pointers for two sets: Schedule and Service. In addition to forward and backward pointers, an owner pointer is maintained for each set. An owner pointer contains the address of the owner of the set: Case address for the Schedule set and Post address for the Service set.

### Installed Region Registers

In addition to the node pool described above, the communications area also contains sixteen "registers" used for control and fast access to selected records (Table 7). Only five of the "registers" are used by the tracking system: Vacant, Posts, Cases, Nodes and Wait. The other eleven "registers" are reserved for future use. When the region is installed into memory, all of the terminals are allocated and Post records created at the top of the region. A single Free record will contain all remaining nodes. At the end of each day, the region is re-initialized to the startup state.

**Table 6d. Contents of a Case record:**

| Location (Bytes) | Length (Bytes) | Description |
|---|---|---|
| 1 - 2 | 2 | Record type ID (Always 4) |
| 3 - 4 | 2 | Next Case record in Patient set |
| 5 - 6 | 2 | Prior Case record in Patient set |
| 7 - 8 | 2 | First Task record in Schedule set |
| 9 - 10 | 2 | Last Task record in Schedule set |
| 11 - 14 | 4 | Database Key of patient |
| 15 - 34 | 20 | Patient's name |
| 35 - 42 | 8 | Time patient entered system (arrival) |
| 43 - 50 | 8 | Patient medical record number (Id) |
| 51 - 53 | 3 | Staff physician initials |
| 54 - 56 | 3 | Resident physican initials |
| 57 - 58 | 2 | Status flags (e.g. Available) |
| 59 - 60 | 2 | Not used |
| 61 - 62 | 2 | Next Case record in Dispatch set |
| 63 - 64 | 2 | Current location (Post) of patient |
| 65 - 68 | 4 | Patient Status (e.g. Outpatient) |
| 69 - 70 | 2 | Current service (Task) being provided |
| 71 - 72 | 2 | Critical Care flag |
| 73 - 96 | 24 | Not used |

**Table 6e. Contents of a Task record:**

| Location (Bytes) | Length (Bytes) | Description |
|---|---|---|
| 1 - 2 | 2 | Record type ID (Always 5) |
| 3 - 4 | 2 | Next Task record in Schedule set |
| 5 - 6 | 2 | Prior Task record in Schedule set |
| 7 - 8 | 2 | Owner Case record of Schedule set |
| 9 - 10 | 2 | Next Task record in Service set |
| 11 - 12 | 2 | Prior Task record in Service set |
| 13 - 14 | 2 | Owner Post record of Service set |
| 15 - 22 | 8 | Scheduled time of service |
| 23 - 24 | 2 | Type of service scheduled |
| 25 - 28 | 4 | Database Key of assigned Rx course |
| 29 - 30 | 2 | Schedule time in Tocks since midnight (One tock = 2 seconds) |
| 31 - 32 | 2 | Status Flags |

**Table 7. Installed Region Registers**

| Register | Description |
|---|---|
| Vacant | Address of first Free record in region |
| Posts | Address of first Post record in region |
| Cases | Address of first Case record in region |
| Nodes | Pool size (in nodes) |
| Wait | Address of Reception area Post record (console) |

## Summary

The "Patient Tracking System" has proven to be more useful and versatile as the intercom system previously used by the Department. Users can now see their current schedules and patient locations by glancing at their terminals. Soon automated billing of services will be completed, which reduces the amount of paperwork flowing through the service areas. Using an installed region to contain all the tracking system data structures allows the flexibility to add new functions without major rewriting of the system tasks. For instance, terminals may be added or removed by a utility task attaching to the installed region. This is useful when a terminal line fails and a new line is added while the tracking system is running without loss of information.

## Footnotes

{1} The escape character ESC has the ASCII 7-bit code 27 decimal.

{2} Upon startup of the IAS timesharing system, terminals may be designated as either a general purpose timesharing terminal using the "SET TERMINAL" command or as a device using the "SET DEVICE" command.

{3} The IAS timesharing system, PDS, interprets the <Control/C> as a command to interrupt and suspend the current task and allows the user to abort the task.

{4} Medical record number is issued by the Medical Records Department of the Hospital and used to uniquely identify the patient throughout the hospital.

{5} Database key is the page and line number where a record is located in the patient database.

{6} "Kw" is constant equivalent to 1,024 words of memory.

**References**


1. Digital Equipment Corporation, IAS System Directives Reference Manual, September 1984 Order No. AD-H002B-T1, pp 4-84 to 4-88.

2. Digital Equipment Corporation, IAS Device Handlers Reference Manual, September 1984 Order No. AD-H004A-T2, pp 2-33 to 2-37.

3. Digital Equipment Corporation, IAS PDS User's Guide, September 1984 Order No. AD-H003B-T1, pp 14-120 to 14-121.

4. Digital Equipment Corporation, IAS Executive Facilities Reference Manual, December 1980 Order No. AA-H005A-TC / AD-H005A-T1, pp 2-2 to 2-4.

5. Digital Equipment Corporation, IAS FORTRAN Special Subroutines Reference Manual, December 1980 Order No. AA-H001A-TC / AD-H001A-T1, p 5-28.

6. Smith, Ted, Baren, Jill M. and Curley, Robert F., "A Radiation Therapy Patient Information Management System", Proceeding of the Digital Equipment Computer Users Society, May 1985, pp 83 to 95.

7. Digital Equipment Corporation, IAS System Directives Reference Manual, September 1984 Order No. AD-H002B-T1, pp 4-125 to 4-126, 4-154 to 4-156, 4-168 to 4-170.

8. Digital Equipment Corporation, IAS Executive Facilities Reference Manual, December 1980 Order No. AA-H005A-TC / AD-H005A-T1, pp 2-8 to 2-10.2.

9. Digital Equipment Corporation, IAS Fortran Special Subroutines Reference Manual, December 1980 Order No. AA-H001A-TC / AD-H001A-T1, p 5-27.

10. Digital Equipment Corporation, IAS Executive Facilities Reference Manual, December 1980 Order No. AA-H005A-TC / AD-H005A-T1, p 2-13 to 2-14.

11. Digital Equipment Corporation, VT220 Programmer Pocket Guide, Order No. EK-VT220-HR-001, pp 10-13.

# LANGUAGES AND TOOLS SIG

# Filling Some Holes In The VAXC Run-Time Library — The "system" function and related support routines

**Wayne E. Baisley**
Rockwell International / Graphic Systems Division
Lombard, Illinois

LT108, DECUS Fall 1986 Symposium
San Francisco, California

## Abstract

The VAXC Run-Time Library provides VMS support for a large subset of the UNIX system support routines. One notable exception is the "system" function, which passes a command line to a shell or Command Line Interpreter (probably the easiest way to make any utility "callable"). This paper describes an implementation of the "system" function using VAXC. Also described are a number of related and general support routines, and header files.

UNIX[1] has been the source of many very useful programs and ideas. Some of the better known examples are the *Make* utility, which automates compilation and linking based on dependency rules and file time-stamps; the Source Code Control System (SCCS), for managing multiple versions of source files; the ***shells*** which are programmable Command Line Interpreters (CLI); ***pipes*** which allow the output of one program to be fed directly into another without having to create an intermediate file; Grep, Who, and other utilities; and, of course, the C language.

Fortunately for VAX users (the majority of whom use VMS), VMS is flexible enough to capitalize on, or at least to accommodate, such useful ideas. Corresponding to the UNIX facilities mentioned above, VMS has the Module Management System (MMS), which is very similar to *Make*; the Code Management System (CMS), which is analogous to SCCS; the DEC/Shell product which is the UNIX Bourne shell, complete with pipes and many UNIX utilities on VMS; Grep, Who, and other utilities, which are available from DECUS; and, of course, VAXC along with the VAXC Run-Time Library.

All these, and others, are readily available VMS equivalents of UNIX concepts and programs. The extent of this cross-pollination leads me to conclude —

### VMS can do anything UNIX can do better

It is left to the reader to interpret that conclusion as best fits his or her experiences and philosophical bent.

[1]UNIX is a trademark of AT&T Bell Laboratories

### The VAXC Run-Time Library provides most of the useful UNIX C features

Let's take a closer look at the VAXC Run-Time Library, which is DEC's name for their collection of C support routines. The VAXC Run-Time Library provides most of the useful features of UNIX C, including

- The character string manipulation functions
- The standard (*i. e.* generic) I/O functions
- The UNIX-specific I/O functions
- The ***curses*** CRT window management functions
- And many others

The VAXC Run-Time Library also provides unique VMS features, such as DECnet and RMS support, and the VAXC calling standard, which allows VAXC to call and be called by any other VAX language.

The VAXC compiler, besides all the standard language features, also supports the VAX Symbolic Debugger, Language Sensitive Editor, Common Data Dictionary and other software tools. It even goes UNIX one better by producing listings, if desired.

### The VAXC Run-Time Library omits at least one of the commonly-used UNIX C features

One particularly useful UNIX C feature has yet to be exploited by the VAXC developers, however.[2]

[2]The "VAXC Futures" talk at the Symposium indicated that system will likely be supported in the next major release.

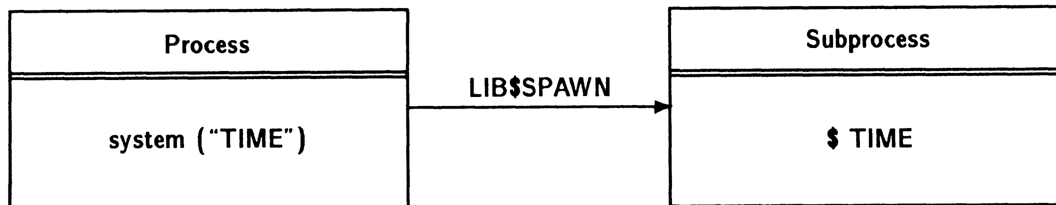



Figure 1

The system function —

is a means of easily passing a command to the shell or CLI.

What do I mean by "easily"? Well, certainly not necessarily easy to implement, at least not on VMS. But an example will show how easy it is to use. The following C statement

```
status = system ("TIME");
```

shows a static command string being passed to the system function for execution, with the exit status to be stored in the variable "status". It's that easy to use. And the command strings can be constructed "on the fly", rather than statically defined, as is the case in this example. Since string manipulation is one of C's stronger points, this combination of features is quite powerful.

### The system function provides a very useful service

I first used system on RSX-11M to integrate separate Whitesmiths Ltd. C compiler components. Rather than having to invoke a command file or run the individual components separately, a single compile command can be issued. For example, the command line —

```
>pcc -fps xyz.c
```

produces a sequence of commands which are passed by the system function to the RSX CLI known as MCR —

```
pcc> cpp -xi qr:[301,376] -o xyz.tm1 >xyz.er0 xyz.c
pcc> del xyz.er0;*
pcc> cp1 -cemn6 -o xyz.tm2 >xyz.er1 xyz.tm1
pcc> del xyz.tm1;*
pcc> del xyz.er1;*
pcc> cp2 -fo xyz.tm3 >xyz.er2 xyz.tm2
pcc> del xyz.tm2;*
pcc> del xyz.er2;*
pcc> cp3 -i 6no07n -t xyz -o xyz.tm4 xyz.tm3
pcc> del xyz.tm3;*
pcc> mac xyz.obj/en:lc,xyz.tm5/nl:toc/-sp=xyz.tm4
pcc> del xyz.tm4;*
pcc> cls -fz 02.14 -i 6no07n -t xyz -o xyz.lst xyz.c
pcc> del xyz.tm5;*
```

The use of system or its equivalent is perhaps the easiest way to make a "callable" interface for a software package, and has the advantage of being "portable" in the sense that it doesn't depend directly on RSX system directives. Incidentally, the "-fps" part of the PCC command line stands for "produce a full listing", "don't submit the listing to the print queue", and "show me the CLI commands as they are issued". This terse, even cryptic approach to command line options epitomizes the UNIX philosophy. Form follows (or, rather, is subsumed into) function. VMS is nearly the opposite, with every option spelled out. At least DCL allows abbreviations.[3]

### The system function was reinvented for VMS

A year or so after using system in the PCC program, I began using a VAX for RSX development, and had to write a system function to work with VMS. This proved to be much more difficult than had been the case for RSX, which had involved little more than the Spawn (SPWN$) and Wait For Single Event Flag (WTSE$) directives. The main reason for this is that a VMS "process" can execute only one image at a time[4] (not counting DCL which executes in Supervisor Mode).

Fortunately, the LIB$SPAWN routine in the VMS Run-Time Library does nearly all that system requires, and certainly all the hard parts. As represented in Figure 1, it creates a subprocess, copies logical names and DCL symbols, handles many other messy details, and then

[3] I don't wish to leave a negative overall impression; whatever VMS may lack in brevity is certainly more than compensated for in its documentation and error messages.

[4] It should be noted that RSX enjoys the same comparative advantage over UNIX here as it does over VMS; executable images (tasks) are run directly, and there is no "process" context to be established.

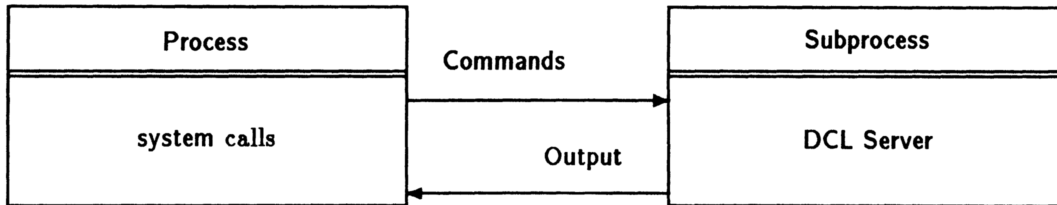



Figure 2

passes the command to the CLI. Implementing system as a straight interface to LIB$SPAWN was fairly simple, but even a "simple" use of LIB$SPAWN is not as easy as it first appears. This is due to restrictions imposed whenever SYS$INPUT or the *input-file* parameter passed to LIB$SPAWN is not a terminal, as is the case for command procedures, MMS processing, and batch jobs, each of which can produce different results.[5] In addition, such a simple interface is very slow since a new process is created and then deleted for each call to LIB$SPAWN.

### And then performance was improved greatly

Performance was improved greatly by calling LIB$SPAWN to create a subprocess just once, and then reusing it as a "server" for DCL commands, as MMS does.

Mailboxes are used to feed commands to the "server" and to retrieve output and status information, as shown in Figure 2. System writes commands to the input mailbox, as represented by the upper arrow, which are handled by the subprocess's CLI. Any output or error messages are written to the output mailbox represented by the lower arrow, read by system, and echoed to SYS$OUTPUT. System also uses the mailboxes to obtain the exit status of the command, which is discussed more fully below.

Since a typical PCC command calls system 6 times, the overhead of creating and deleting 5 subprocesses is avoided, a savings of about 9 CPU-seconds on a VAX-11/750 with this scheme, which can be a large fraction of the entire compilation process. Besides the performance improvements, the use of mailboxes resolves the *input-file* problems mentioned in the previous section.

### The implementation of system on VMS was complicated, though straightforward

As indicated previously, the implementation of system on VMS was complicated; however it was straightforward. The following factors contribute to the difficulty —

*Setting up mailboxes*

First, we must create mailboxes. This is not terribly difficult, but something of a nuisance, particularly since we wish to avoid inadvertant mailbox (and process) name conflicts. I chose to derive the mailbox names from the name of the calling image and the process id (also called the P-I-D, or *pid*). For example, a PCC command might produce a subprocess named PCC_0063, and mailboxes named PCC_0063_INPUT and PCC_0063_OUTPUT.

*Passing arguments by descriptor*

Next, arguments to the VAX Run-Time Libraries and System Services are usually passed by a means of a general-purpose object-oriented data structure called a descriptor, which is very flexible but more bother than strings (from C at least). This is another example of the difference between UNIX and VMS. UNIX typically uses NUL-terminated strings, which are adequate for most but not all situations. Both have arguable advantages, as well as vocal detractors.

*Obtaining the image exit status*

Next, we must obtain the image exit status for each command we issue to the "server". As mentioned above, system uses the mailboxes to do this. The trick here is to wait for a read to be posted on the input mailbox by the subprocess's CLI, issue another command to the server to write the current value of the $STATUS symbol to the output mailbox, read the result, and finally return the severity portion (the low order 3 bits) to the caller. This trick works like magic, and isn't confounded by attempts of an image in the subprocess to read from the mailbox,[6] since DCL always appropriates commands lines which start with a dollar sign, returning an End-Of-File status to the reader.[7]

I could have used other means to obtain the command status, such as intercepting image accounting messages, but this scheme is easy to understand and implement.

[5] One interesting but annoying variation is to specify the null device (NLA0:) as the *input-file* parameter to LIB$SPAWN, which produces log-out messages from the subprocess after each call.

[6] See the Futures section below for more on this topic.

[7] The exception to this rule is the $DECK ... $EOD sequence, which disables DCL's normal input processing. An attempt to use this sequence would probably give system severe indigestion. MMS probably wouldn't like it either.

There are a few other less interesting but important items which system must take care of, such as

- handling unexpected server process termination,
- handling ^Y and ^C aborts, and
- cleaning up on the exit of the calling image.

So in spite of some clumsiness, VMS proves quite capable of supporting a well-behaved, low-overhead implementation of system.

### The system function serves as a useful example

Besides the service it performs, the system function serves as a useful programming example of

- how to use the LIB$SPAWN routine,
- calling VMS System Services from C,
- using descriptors in C,
- writing AST service routines,
- writing exit handlers,
- how VMS can be difficult for programming.

### Other related functions were developed along with system

Other related functions were developed along with system which are of some utility on their own.

- *cimageid* returns the name of the currently executing image as a string. It is used by system to construct the mailbox and subprocess name strings.
- *complain* is a simple way to produce VMS-style error messages.
- *crembx* creates a mailbox with a given name and certain fixed parameters such as buffer length.
- *chain* executes a single DCL command and exits, using the LIB$DO_COMMAND Run-Time Library routine. It is significantly faster than system for a single command since it doesn't create a subprocess.

### A small UNIX C annoyance was fixed, too

Since I use RSX and VMS in my work, I have little exposure to and consequently little criticism of UNIX. But because VAXC, which I use constantly, owes so much to UNIX C, I will register a small complaint about a "feature" of the UNIX C string functions. The string functions

- *strcat* - concatenate two strings
- *strcpy* - copy a string

and their bounded counterparts

- *strncat* - concatenate up to 'n' characters
- *strncpy* - copy up to 'n' characters

return the address of the beginning of the output string. The ending address is far more useful, since one almost always has the beginning address beforehand.[8] Having the string functions return the beginning address results in many unnecessary calls to *strlen* to determine the current length of a string under construction, or the use of *strcat* (which is nearly the same) where *strcpy* would have sufficed, which seems mildly unUNIXlike.

So some new string functions were invented, namely

- *strzcat* - concatenate two strings
- *strzcpy* - copy a string

and their bounded counterparts

- *strxcat* - concatenate up to 'x' characters
- *strxcpy* - copy up to 'x' characters

which perform exactly as do the standard versions, but return the address of the terminating NUL byte, rather than the beginning address of the output buffer.

### These new functions make constructing a string so easy ...

The code fragment below shows how the PCC program uses the *strzcpy* function to construct the string "cpp -xo xyz.tml" in the buffer named *cmdbuf* which it later passes to the system function. The character pointer *cp* is used as a place-holder at each statement, and then as the starting address for the next copy operation.

```
cp = strzcpy (&cmdbuf, "cpp -xi ");
cp = strzcpy (cp, includes);
tml_p = cp = strzcpy (cp, " -o ");
cp = strzcpy (cp, outname);
tml_end_p = cp = strzcpy (cp, ".tml");
   :
```

[8]The exception is the case where the address passed to the string funtion is that returned by a memory allocation routine such as *calloc*. But it is trivial to record the address simultaneously. For example, *strcpy ((address = calloc (size)), "string");*.

This fragment also shows the addresses of intermediate points in the output string being recorded as they are encountered, rather than determined separately. PCC uses this feature to isolate substrings for later use in deleting the temporary file "xyz.tm1", as in the following code section.

```
*tm1_end_p = '\0'; /* terminate the string */
do_delete (tm1_p);
```

### Future enhancements looking for a volunteer

System by definition doesn't allow for the commands it executes to perform any direct I/O with the calling process. Such a feature is the domain of the UNIX C *popen* (and *pclose*) function, which could be implemented as a variation of the system function. The difference between system and *popen* is that *popen* establishes the "server" subprocess and mailboxes, and *popen*'s caller can then write and read whatever it may desire to and from the subprocess. Are there any volunteers?

**d e c u s has it now.**

This software is available on the Symposium Languages and Tools, and VAX tapes, in the directories [RIGS . CCLIB . . . ], including

- Sources
- Command Procedures
- Object Library
- Documentation
- A Simple Test Program

### References

VAX/*VMS Run-Time Library Routines Reference Manual*, Digital Equipment Corporation, Maynard, Massachusetts,
VAX/VMS Version 4.4, April, 1986,
pp. RTL-294 – RTL-298 for the LIB$SPAWN routine

VAX/*VMS DCL Dictionary*,
Digital Equipment Corporation, Maynard, Massachusetts,
VAX/VMS Version 4.4, April, 1986,
pp. DCL-610 – DCL-614 for the DCL SPAWN command which uses the LIB$SPAWN routine

*ULTRIX-11 Programmer's Manual Volume 1*,
Digital Equipment Corporation, Maynard, Massachusetts,
or any other UNIX Programmer's Manual,
Volume 1, Section 3 (Subroutines), for the system routine

# CHOOSING AUTOMATED STRUCTURED ANALYSIS (SA) TOOLS

June Baker
Computer Sciences Corporation
Falls Church, Virginia

## ABSTRACT

In pursuit of increased productivity and quality, organizations are seeking automated tools to assist in the design and development process. A plethora of automated structured analysis tools currently is available, and organizations must determine which one(s) fulfill their needs.

Our research shows that there are vast differences in the capabilities and usefulness of commercially available products. We evaluated several during a six-month period in order to recommend sets of tools to ongoing and startup projects at CSC. This paper discusses the criteria we used to study a subset of available tools and what we found.

## 1.0 INTRODUCTION

During the past several years automated structured analysis tools have been introduced in the commercial marketplace. When we began our evaluation there were approximately six viable products announced and distributed. Currently, there are innumerable products available that operate on IBM* PCs and compatibles, several that operate on the Digital Equipment Corporation (DEC*) VAXstation II* and several that operate on APOLLO* and/or Sun* workstations. At least one product uses a UNIX* type operating system. We chose four products for our initial evaluation, and these products operate on IBM PCs. Although the evaluation and subsequent report were completed early this year (1986), we have obtained and reviewed other products which appear to have innovative features. We have tested, reviewed literature and have seen demonstrations of twelve (12) automated structured analysis tools. The report currently is being updated to be distributed within CSC to appropriate management.

This paper discusses our goals, basic requirements, our evaluation approach, comparative results and conclusions. Although we limited the evaluation to tools that operate on IBM PCs and compatibles, the methodology can be used to evaluate any structured analysis tool.

## 2.0 OVERVIEW

Organizations responsible for developing software within time and cost constraints are seeking ways to both increase the productivity of their software engineering staff and improve the quality of the completed software. Studies have demonstrated the high cost of fixing problems uncovered during testing and maintenance, and this has fostered a concern for more complete, verified designs in the early stages of development. Specifying complete, accurate requirements and designs early in the development process is most economical and will produce better results during the later development stages.

The current surge of interest in increased productivity and quality of software products has encouraged creative companies to develop automated structured analysis tools. These tools are supposed to enable users (professional software engineers) to develop complete specifications based upon proven structured analysis methodologies developed by DeMarco or Gane and Sarson.

## 3.0 OUR GOALS IN EVALUATING STRUCTURED ANALYSIS TOOLS

There were three major reasons that caused us to evaluate some commercially available structured analysis tools. First, CSC management believes that automated tools will enable us to perform better on software development projects, and it is necessary to have in-depth knowledge of available products to demonstrate and recommend throughout our Division. Certain tools are applicable to the environment of one project and not to others, and it is vital to recommend the best tool or an appropriate selection of tools for a specific project. From the start of the evaluation process we have been consultants to project management and have recommended tools that currently are being used on projects. We have received some feedback on the value of these tools to the individual projects, and eventually we will accumulate feedback on a formal basis. Furthermore, we are training project members in the use of tools in order to get them started using the tool properly.

Second, CSC must demonstrate to current and potential customers its knowledge of and familiarity with those automated tools that it proposes to use on future projects. We have participated actively in current proposal efforts by recommending specific tools, describing them in documentation and creating data flow diagrams and analysis reports to be included in the formal proposal.

*IBM is a registered trademark of International Business Machine Corp.; DEC and VAXstation II are registered trademarks of Digital Equipment Corp. APOLLO is a registered trademark of Apollo Computer, Inc.; Sun is a trademark of Sun Microsystems, Inc.; UNIX is a trademark of AT&T.

Third, we are involved actively in a Division IR&D project to develop an Integrated Software Engineering Environment (ISEE) to be used internally as a software development workstation. The workstation consists of tools spanning the life-cycle development phases (requirements analysis, initial design, implementation and testing). We are using commercial off-the-shelf tools with some interfaces that we are developing ourselves. It was necessary to determine which structured analysis tool provided us with the most applicable capabilities to include in the ISEE. Several structured analysis tools will be resident in the workstation, but we are providing interfaces to one only.

## 4.0 EVALUATION METHODOLOGY

In an attempt to be as objective as possible we developed a straightforward strategy to use in evaluating the automated tools. The evaluators were senior software engineers with experience using IBM and DEC equipment. One of the evaluators is a "hacker" and easily learns how to use diverse products. The other evaluator has been a manager for the past eight years and has not programmed recently, but has used VAX- and PC-based tools on a daily basis.

First, product literature was read to determine if the tool met a significant subset of our requirements. Much product literature transports the reader to fantasyland. Indeed, at first we believed that the "next" tool would fulfill all our desires and needs. We finally did determine that although vendors promised tremendous increases in productivity and quality by using their tools, no one tool satisfied all of our requirements. We chose two mature, stable tools, one inexpensive tool which appeared to have good capabilities and a new tool marketed by a well-known name in the area of structured analysis. At the time we began the evaluation there were only one or two other PC-based tools available, so we believe that we effectively surveyed the marketplace as it existed at the time.

Next, we acquired the software (we purchased some and borrowed some from eager vendors), and we installed the software using directions specified by the vendor. Hardware adjustments were made also following vendor instructions. When all else failed we contacted a vendor representative for help.

If a tutorial was available it was accessed and completed (occasionally, the tutorial "bombed"). The user became familiar with the tool's major capabilities by self-study.

All product documentation was read thoroughly to learn the complete set of capabilities and to answer questions about functionality. Also, we determined the manual(s) usefulness, accuracy and approach to good English. We had preconceived ideas that a sloppy manual often reflected a sloppy approach to a product.

We contacted the vendor's technical support group whenever there were problems using the tool and whenever we found software bugs.

A standard set of data flow diagrams was produced using the DeMarco methodology. The diagrams had previously been drawn manually for an ongoing CSC project. We created a context diagram, a top-level (level 0) diagram and one exploded diagram (level 1).

Dictionary entries were made to define all objects and data elements for later reporting and validation. We created reports based on the dictionary contents.

The data flow diagrams were validated according to the capabilities of the tool. We used a fourteen-point checklist to determine the thoroughness of the validation process.

Text capabilities were used whenever possible to create mini-specifications and to add information to the dictionary and diagrams.

Additional functions of the product were reviewed and their usefulness evaluated. For example, we created structure charts and presentation graphs when the tool permitted, and we added information to the data dictionary whenever possible.

We gave demonstrations of the product to CSC personnel, and we noted people's questions and opinions.

We revisited the products that we had evaluated early in the study in order to eliminate biases caused by increasing familiarity with use of an automated tool and a mouse.

Finally, we designed a matrix to include all tools that were evaluated with an entry for each requirement. For example, under the requirement of "User Friendliness" we rated each of the following criteria separately: easy to learn and use, documentation, HELP facility, tutorial, hot line service and user defined function keys. We used a scale of 0-3 for the ratings. When the feature was not available the rating was 0. When the feature was available, but didn't perform as advertised or performed poorly, the rating was 1. When the feature worked adequately the rating was 2, and when the feature worked well and helped the software engineer do the job well the rating was 3. We did not give weights to any features, although in retrospect we realize that certain features are more important than others. We will give weights to criteria in future evaluations.

The underlying principle in our methodology was our belief that a product could not be evaluated simply by reviewing the documentation and sales literature or by viewing a vendor-presented demonstration. We had to have hands-on access to the tool in order to feel comfortable with its features. In addition, we believe that a tool must be used on a daily basis in a true project environment for its worth to be revealed. We were not operating in a project environment for this evaluation; however, we will be collecting information and statistics from project members as the recommended tools begin to be used throughout our Division.

## 5.0 OUR REQUIREMENTS AND WHAT WE FOUND

1. First, all tools had to operate on IBM PCs and

compatibles using the MS*-DOS operating system. Since there are a number of tools that operate on VAXstations and other workstations, it is conceivable that our prototype ISEE will be moved to a VAX-based workstation once the initial IR&D project has been completed and evaluated.

All the tools that we evaluated ran on our IBM PC/AT, which operated under MS-DOS version 3.0, had 640 KB memory, a hard disk (20 MB), floppy disk, Hercules* Monochrome Graphics Card, monochrome monitor and an Epson* FX printer. All the vendors gave instructions for using the tool with other hardware configurations and with compatibles; however, we did not attempt to confirm such capabilities. The tools required much storage (up to 3 MB) which quickly ate up the capacity of our hard disk. The data flow diagrams were stored as MS-DOS files and combined with the data dictionary gobbled up more storage. Also, the products tend to be memory hogs, so we needed all of the 640 KB that we had. Some of the products were sloppy about cleaning up after themselves, so occasionally when we moved from one tool to another the system couldn't find enough free space and we had to reboot.

2. The term "user friendliness" is a popular buzzword and is prevalent in product literature to encourage purchase of the vendor's software. A friendly product is not "chatty" and verbose; rather, it is unobtrusive and easily accessible. We believe that software engineers will use automated structured analysis tools only if they can be mastered easily and if modifications can be made quickly as well. Under the basic requirement for user friendliness we included the ability to learn the basics of the product quickly and without extensive training, good user documentation, an on-line HELP facility, a vendor "hot line" for software and hardware problems, understandable prompts and messages as guides throughout the software and a tutorial to enable the user to learn to use the product quickly. In addition, we looked for both expert and novice modes to appeal to a wide range of users.

Generally, the less complex and sophisticated the tool, the easier it was to learn quickly and effectively. The tools with the greatest range of capabilities were more difficult to learn, and users had to relearn functions if they did not use the tool constantly. We were able to use some functions and create data flow diagrams after one session with the tool, regardless of its complexity. However, in all cases it took much longer to learn to use the dictionary and reporting capabilities.

The documentation was necessary to learn how to use all the features of the tools, but after a while the evaluators were able to bumble along with the pop-up menus and the icons. When all else failed we looked at the documentation. The manuals provided by the vendors ranged from poor to adequate. Some were written in barely intelligible English, which tended to detract from the user's confidence in the product. All of the documentation would have benefited from the inclusion of detailed and numerous examples.

*MS is a registered trademark of Microsoft Corp. Epson is a registered trademark of Epson America, Inc. Hercules is a registered trademark of Hercules Computer Technology.

On-line HELP facilities were available in most of the products. None of them was particularly useful, and we used the manuals to research information or called the customer technical support group.

All tools had error messages and prompts, and these ranged from cryptic, repetitive, and confusing to adequate. They were no better nor worse than most software error messages and prompts.

Each of the products we evaluated had tutorials to help the user get started. These ranged from poor to excellent. The most mature products had detailed tutorials and step by step procedures in the manuals. The users were able to use the products after working with the tutorials for one or two sessions.

One of the products allowed the user to define function keys to store repetitive operations. This is an excellent and sophisticated feature, but it is not for the novice nor casual user. One of the products had good facilities for both novice and expert users. In fact, the user could combine both modes to create an interface most comfortable for him or her.

Overall, the products made an attempt to be user friendly, but definitely there is room for improvement. Almost all the tools we evaluated recently have distributed new releases, but the emphasis has been on increased functionality and bug fixes rather than on an improved user interface.

3. All products had to include interactive graphics manipulation enabling the user to draw and modify data flow diagrams and other diagrams quickly and easily. We normally use the Tom DeMarco structured analysis and system specification methodology, and we looked for products that supported this methodology.

All the products allowed us to create and modify data flow diagrams following DeMarco methodology. It is apparent that production of nifty graphics has become routine, because every product we have seen does a reasonably good job. The tools differ in enabling the user to move objects singly, with adjacent data flows or with multiple objects. Also, with some tools the user had complete control over placement of objects and text and with size of objects, and with others the system determined placement and size. Some tools allowed the user to vary the graphics on data flows and arrows. Some of the tools allowed the user to draw diagrams with symbology other than that required for DeMarco data flow diagrams. For example, several allowed the user to draw structure charts, some had many different symbols and some had icons, such as terminals, people, disks, etc. Therefore, the tool had uses beyond its original purpose. Several of the tools also allowed the user to define his or her own symbols and save them for later inclusion in diagrams.

4. Next, the product had to include an integrated text facility. Text must be included within graphic symbols on the diagrams and separately as mini-specifications for a structured specification. Templates to guide the user in entering text are useful, and these templates should be capable of being customized.

All of the tools required that the objects (process, source/sink, data store, data flow) be labeled, but one product did not allow additional text on the diagram. This product automatically titled a child diagram with the label appearing in its parent. Most of the tools allowed the user to control the placement of the text within the objects, but one tool gave the user no control over such placement. One of the tools restricted amount of text within objects based on the system printer fonts. One tool required a label to be affixed to an object before it allowed the user to continue.

One product automatically numbered process bubbles in accordance with DeMarco rules. One product required that a number be placed in a process bubble, but it never checked beyond determining that numbers 0-9 were used exclusively. We entered identical numbers on several process bubbles and never received an error message. The other products did not require numbering of processes.

The capability to add a mini-spec or other project documentation varied from product to product. Two had no capabilities whatsoever. The others had full word processing capabilities, and the user could link a mini-spec to a primitive process easily. None of the products had templates to assist a user to enter a mini-spec.

5. In order to have a complete structured analysis tool it was essential that the product create and maintain an integrated data dictionary. We wanted the mechanics of entering and modifying entries in the data dictionary to be as transparent to the user as possible. Also, the user needed the capability to query the data dictionary easily.

Integrated data dictionaries were present in three of the products. The other product claimed to have a data dictionary capability, but the user had to access a separate database management system to populate the data dictionary. In addition, the database management system had to be purchased separately from another vendor. The process was so cumbersome and so faulty that the functionality was nil. Including objects in the data dictionary was automatic in one product as was the capability to make global changes to the data flow diagrams and dictionary by making a change in one place only. It was more tedious to add or change objects to the data dictionaries in the remaining two products, but more information could be stored and query facilities were good.

6. The capability to analyze the structured specification in terms of conformance to the DeMarco methodology, consistency, completeness and accuracy was an essential function for the products we evaluated. A checklist of fourteen (14) validation requirements was generated to determine how thoroughly each of the products adhered to the methodology.

No product provided automated support of all structured analysis functions and conventions. One product provided almost no analysis (this was the same tool that had no integrated data dictionary). The other three provided analysis of most of the structured analysis functions and conventions and reported with error messages and sometimes highlighted the errors. One product brought net input and output data flows to the child diagram from its parent when the user "exploded" to the next level. Thus, the user could not forget accidentally to include these data flows in the child. The same tool provided many syntactical checks as the user created the diagram, and it was useful to catch errors as the diagram was being created rather than during analysis.

7. Because we needed the capability to interface inputs to and outputs from the product, we looked at the capability to extract data into an ASCII file describing the data flow diagrams and/or the contents of the data dictionary.

All the tools allowed the user to extract data into an ASCII file describing the data flow diagrams and/or the contents of the data dictionary. The formats and information captured on the file differ from product to product. We have not investigated the individual formats to determine if the information is useful, but we will do so later in order to use the files to interface to other programs.

8. We were concerned with product stability as well as the length of time the product has been available to the market. We were looking for mature companies with a growing customer base in order to ensure technical support and upgrades to the product.

Two of the products have been on the market for at least two years, and the companies have created new versions to include additional functionality as well as bug fixes. The companies themselves are well established, have good technical support staffs and have provided us with advice and help. One company is a new startup, probably run by two or three programmers in the backroom. The product has many good features, but it has had a tendency to crash while being used by an actual project and during evaluation. New releases are sent out, and the vendor is eager to help, but the company is far from mature and stable. There have not been any releases based on increased functionality from this company. The fourth product comes from a mature, stable company, but the product has many bugs and we have had hard crashes while using it. A new version currently is being distributed, so we will have the opportunity to determine if it has become more stable.

9. Since CSC normally develops software for medium to large systems, multi-user and networking capabilities scored high in our list of requirements. In addition, it was essential that a product support more than one project (and therefore more than one data dictionary) within a single computer or a network.

Two of the products had access protection that could be established by a system administrator even though these products operate on single-user non-secure PCs. Not only were user IDs and pass-

words (optional) required, but the system administrator could restrict access to various functions, most notably access to the data dictionary.

All of the products allowed multiple projects (multiple data dictionaries) to be established. Thus, one copy of a product could be shared by a number of projects. However, since PCs are single-user workstations, just one person at a time could access a single project. For projects with more than two analysts, this could prove to be an effective barrier to production. Therefore, we looked for networking capabilities to enable a product to be used in mediumto large-scale projects. One product had full networking capabilities, and the vendor will work with a user to set up a network facility. Another product had the capability of being included in a network, but access to a project data dictionary was limited to a single user at one time. The remaining two products had no multi-user capabilities.

10. Standard formatted reports and user-defined reports based on dictionary contents were essential. We wanted to be able to view the reports immediately on the computer terminal as well as via the printer.

All products provided some reporting capabilities. Two allowed the user to define contents of the reports as well as accept standard system-defined reports. Unfortunately, creating user-defined reports is complex, and much effort has to be expended to become at ease with this function. Three tools allowed the user to view reports on the terminal screen before deciding whether to save the report and/or print it.

11. Since the graphic material would be used in formal contract-specified documentation, printing and/or plotting capabilities were important features. We required the ability to combine text and graphics in single documents.

Three of the products provided capabilities to combine text and graphics in files and documents. All provided printing capabilities and two provided plotting capabilities. The range of printers varied, but even though we used the inexpensive, low-range Epson for most of our printing, the results were reasonable and acceptable for direct reproduction. One tool gave us a hassle because we used the Epson printer, and we could not get good graphic output unless we tinkered with placement of text and objects. In our report we recommended that users of this tool invest in a high-quality printer for best performance.

12. Once the software engineer has developed a structured specification (data flow diagrams, data dictionary, mini-specifications), he or she must begin initial design in a structured manner. One approach to structured design is to develop structure charts with control and data flow specified. Ideally, the outputs from structured analysis automatically would create structure charts. None of the tools that we examined had this capability, but we looked for some that allowed the user to create structure charts. We believed that if this capability was present, eventually there could be automatic generation of the structure charts using the data dictionary as a central source of information.

None of the tools we examined provide an automatic transformation from structured analysis to initial design. Two currently offer the capability to draw structure charts, but there is no relationship to the objects created during analysis. A new release of one of the other tools promises to provide the capability to create structure charts, but again there will be no automatic transformation. The fourth tool provides no design capabilities whatsoever.

13. Finally, we looked for structured analysis tools that were part of an integrated set of life-cycle software development tools, including project management tools, design, development and testing tools.

The vendor of one tool is dedicated to building tools for the entire software development life cycle, and it currently has project management tools that can be linked to its structured analysis tool, and it provides bridges to application generators. It is possible that one additional vendor will be teaming with a vendor of application generators and code generators to enhance the capabilities of their products. No vendor had tools for testing, and the design capabilities were not tightly-coupled with the analysis capabilities.

## 6.0 CONCLUSIONS

The use of an automated tool will speed up the analysis process by focusing on mechanical efforts and freeing the analyst of tedious, repetitive chores. All too often even strong adherents of structured analysis methodologies balk at modifying data flow diagrams manually, thereby reducing accuracy in documentation. It should be noted that the identical effort must be expended in the concept and original thought processes of individual analysts who use automated tools as when analysis is done manually.

The people who use the tools to develop a structured specification must be aware of structured analysis and design approaches in order to understand why and how the tools should be used. Otherwise, the end result will not be better, just faster. An abundance of attractively packaged specifications can easily disguise a bad design.

All the tools we evaluated had good graphics capabilities. It is easy to draw some nifty graphics with the tools. However, it is tempting to fall into the trap of producing pretty pictures without carefully validating and balancing the diagrams. The graphics output alone is but one part of a structured specification, and the analyst must be careful to do the whole job.

The more mature products with better functionality were more difficult to learn. It took constant reinforcement to use the tools; for example, if we left one of the more complex tools for some time we had to relearn many of the functions. Documentation varied, but none was as accurate nor complete as it should have been.

Adherence to DeMarco rules varied from product to product. Although all vendors claimed to adhere closely to the DeMarco methodology, only the least slick product truly followed the rules. We have noticed that some of the newer products pay

greater attention to the methodology.

In most cases it was not simple to learn to use the dictionary nor to create meaningful reports. Any number of reports could be generated at the stroke of a key or the click of a mouse, but interpreting the contents could be mindboggling. Users will have to concentrate most on accessing the dictionary, storing information within the dictionary and creating and understanding reports.

Since only one product allows multi-user access, the choice of a product could depend upon the size of the project (how many analysts ) as well as price and functionality. The prices varied by a factor of 8 for a single-user copy. On a project about to begin we currently are trying to determine whether to set up a network with an expensive product or buy four or five copies of a single-user tool with excellent functionality and keep a separate controlled dictionary that would be updated procedurally. The price differential is significant, especially when counting costs for networking the PCs.

## 7.0 SUMMARY

The crop of commercially available structured analysis tools are proven productivity and quality aids. Their greatest impact is on automated documentation of data flow diagrams and maintenance of a data dictionary that provide a positive impact on configuration management.

With all the differences among the tools in functionality, maturity and cost, once you have used a reasonably well-automated structured analysis tool, you never again will create a structured specification manually.

# A GENERALIZED CODING STANDARD AND SOME ASSOCIATED TOOLS

E. J. Straub, A. L. Slavich, and C. Winter
Pacific Northwest Laboratory
Richland, Washington

## ABSTRACT

This paper describes a general coding standard which was developed to provide neat, uniform and complete documentation at the module level and some software tools which were developed to support the coding standard. The general coding standard is intended to be used as a tool for writing language specific coding standards. Although several language specific coding standards have been developed with this GCS, only the GCS is discussed in this paper (although tools for the FORTRAN coding standard are discussed).

## INTRODUCTION

In recognition of the increasing need for standardized coding methods among a group of 20 programmers at Battelle, Pacific Northwest Laboratories, a committee of staff members was assigned the task of recommending formal coding practices that would be used by all members of the group. Specific issues addressed by the committee were the needs for better documented code, standard formats for code to ease software review and maintenance, and improved software quality control. Furthermore, any coding schemes that were recommended would have to require minimal effort and cost to implement and must be easily adaptable to the many programming languages used by the group. As a result the committee, utilizing recommendations from the staff members, defined the concept of a generalized coding standard. Language specific coding standards, or implementations of the generalized coding standard to specific programming languages, were designed and special software tools were developed on the VAX 11/780 to aid users in using these standards, as well as to take advantage of the parsability of code documentation which occurs when the standard is used.

## THE GENERAL CODING STANDARD

The General Coding Standard (GCS) is intended to be used as a tool for forming language specific coding standards (LSCS). The GCS describes what information should be provided at the routine level. The goals of the standard are to promote localization of all definitions, information hiding, clarity, and uniformity.

The objectives of the localization goal included localizing the definition of the data structures and global variables used by a system (group of routines), the explicit definition of named constants for values used throughout the system, and the standardization of local variable definitions and declarations for those routines.

The methods of attaining localization differ considerably among various programming languages. While localization methods will be different for each language specific standard, two general types of languages can be considered: compiled languages and interpreted languages.

For most compilers, include files can be used to achieve localization. Depending upon the specific language, these include files may contain declaration statements for global and local variables, definitions of data structures, and definitions of named constants.

Although interpreted languages vary considerably, they tend to offer fewer opportunities for explicit localization than do compiled languages. Their tendency to allow global access to variables once they are defined precludes the local declaration of variables within subroutines, but also removes the concern that variables used by more than one routine will be defined inconsistently. Some interpreted languages allow for the naming of global constants. If this is not the case, named constants may be self imposed by setting a symbol's value to the constant value and never changing it. If the interpreter allows, constants which are used by more than one routine should be defined in include files.

Clear and uniform documentation, as well as information hiding, is achieved by dividing each routine into two blocks: the Definition Block and the Implementation Block (a third block called the Code Block may also be useful). Each block consists of a specified set of information sections. These sections may all be in one location in the source code, or they may be located at different locations (depending on the order in which the specific language requires).

The Definition Block provides all the information that is necessary for one to use (call) the routine, but does not include implementation specific information. When someone wants to use a routine, they only need to look at the Definition block. The details on how the routine is implemented are separated from the details on how to use the routine. This is information hiding, and increases code maintainability by reducing dependency on implementation details.

The Implementation Block provides information about how the routine is implemented. This information is

useful for maintenance and modification the routine. The implementation block may contain the code, or a separate block may be added which contains the code (the code block). Whether the code is given its own block is up to the language specific standard.

The Definition Block

Following is a brief discussion of the sections of information required in the Definition Block.

The Routine Name - The name of the routine. If the specific programming language requires the routine name as part of the code, then no additional information is necessary here. If the language does not, then the routine name must be included in inline documentation.

The Arguments - The data type and a description of each argument to the routine. This description should also state whether the argument's value is used when the routine is invoked, or is modified within the routine.

The Purpose - What the routine does; what problem the routine solves.

The Limitations and Prerequisites - Describes any limitations which apply to the routine, as well as any events which are required to occur prior to invoking the routine. For example, if another routine must be called prior to calling this one, that routine would be mentioned in this section.

The Logical Names and Files Used - Description of all logical names, files, and other devices which are explicitly used by the routine. If a logical unit number is passed to the routine as an argument, or if a filename is passed as an argument, then these files do not need to be documented in this section (since they are already documented in the Arguments section).

The Routines Used - The names of all external routines used (explicitly called) by this routine. Does not list routines which are called by routines which are called by this routine (ie, only routines which are explicitly called by this routine need be referenced here).

The Implementation Block

Following is a brief discussion of the sections of information required in the Implementation Block.

The Development Information - Identifies who created the routine, when and where it was created, and the machine, operating system and compiler/interpreter used to develop the routine.

The Modification History - Describes the who, what, when and why of any modification to the routine made after its development phase.

The Variable and Constant Definitions and Descriptions - The data types and descriptions of all variables and named constants. In the FORTRAN standard three sections are used to achieve this:

- The Variables Section. Contains all local variable and array declarations and descriptions.
- The Common Block Section. Contains only include statements which include common block files.
- The Constants (parameters) Section. Contains only parameter statements and include statements which include files which FORTRAN parameter statements.

The Algorithm - A step by step description of the algorithm used by the routine to achieve its purpose. The algorithm should be described in pseudo code.

The Coding Standard Tools

Following the coding standard can be burdensome. To help alleviate this problem, some software tools were developed to support the standard. Other software tool are being developed which will utilize the parsability of inline documentation which is provided by the standard. Some of the tools work for all languages supported by the standard, while others only work for one language. Following is a discussion of the software tools.

Coding Standard Tools Which Work for All Supported Languages

PUTHEAD - PUTHEAD is probably the most used tool. It puts a coding standard template on the front (or back if the language is interpreted such as DCL) of a source file. PUTHEAD will create the source file if it does not exist. After executing PUTHEAD, the programmer edits the file and completes the template. PUTHEAD examines the filetype specified to determine both what language and what type of template to put on a file. Supported languages and templates are shown in the table below

| Language | Filetype | Template Type |
|---|---|---|
| DCL | .COM, .LNK | program template |
| FORTRAN | .FOR, .F, .FTN | subroutineorprogramtemplate |
| FORTRAN | .CMN | common block template |
| FORTRAN | .PRM | parameter template |
| FORTRAN | .VAR | variable file template |
| PASCAL | .PAS, .P | subroutineorprogramtemplate |

As other languages are supported by the coding standard, Puthead will be modified to recognize other filetypes.

BLDMANUAL - BLDMANUAL reads a group of files and generates a new file which contains the Definition Block from each file. Since the Definition Block contains information about how to use the routine, this has the effect of building a users manual.

Coding Standard Tools Which Only Support FORTRAN

GETINFO and PUTINFO - GETINFO and PUTINFO are currently being developed. They will allow the user to automatically move documentation between external documents and source code. GETINFO is given the name of an entity (a block, section, or variable/symbol definition) to retrieve from one or more source files. The information is written to a specially formatted (Coding Standard Tools (CST) format) ASCII output file. When given the name of

an entity, a CST format file, and one or more source files, PUTHEAD will add/replace the information which is specified for that entity in the CST format file, into the source files.

GENROUT - GENROUT reads a FORTRAN listing file with cross reference (file must have been created with DEC VMS FORTRAN/LIST/CROSS) and generates the Routines Used section. This is a list of the routines explicitly called by that routine. The list has data type declaration statements for all functions called by the routine and lists all of the explicitly called subroutines in comments.

Furthermore, the routines listed are sorted by package. When invoking GENROUT, zero or more package files may be specified. These files contain lists of routines which belong to a certain package. GENROUT will then determine which package a routine belongs to. If the routine does not belong to any package specified, then it is put in the list before any of the packages are listed.

GENVAR and Global Naming Strategies - When used in this text, the phrase "variable name" includes both variable and array names. The phrase "variable description" includes the variable name, data type, and text description of what it is used for.

Global Naming Strategy: What and Why - It is good practice to use a global naming strategy (GNS) when developing a software system. GNS means using variable names consistently throughout a system. That is, any variable in the software with a specific name is used for one purpose and one purpose only. Do not confuse GNS with global variables. This discussion pertains to local variables. The idea is to use the same name for local variables which are used for the same thing in different routines. For example, in FORTRAN, ISTAT is a commonly used name for a variable which receives return statuses from function or subroutine calls. If a GNS was being used, each routine in the system would not use variables with the name ISTAT for anything other than receiving return statuses.

GNS offers several advantages. Some of the advantages are:

- Reduction of the learning curve when becoming acquainted with a new system.
- Elimination of confusion resulting from inadvertent, unrelated uses of the same variable name in different routines.
- Variable names may be documented at the system level. Allowing easy creation of a "programmer's dictionary" of variable names.

The GENVAR Utility

GENVAR may be run in one of two modes: 1) single routine mode, and 2) multiple routine mode. When used in single routine mode, GENVAR's sole function is to help generate the Variables section of the routine's coding standard template. GENVAR's multiple routine mode is used to implement global naming strategies (GNS), and to generate the Variables section of the template. Both of these modes are described.

Nomenclature used in this discussion:

SOURCE - FORTRAN source file. This is a file which contains one (and only one) fortran routine. This file is compiled to generate the LISTING file.

LISTING - FORTRAN listing file with cross reference section. Must have been generated by DEC FORTRAN compiler with /LIST and /CROSS switches on.

VSF - Variables Section File. This is the file which is generated by GENVAR to be included in the SOURCE file. It contains the Variables Section.

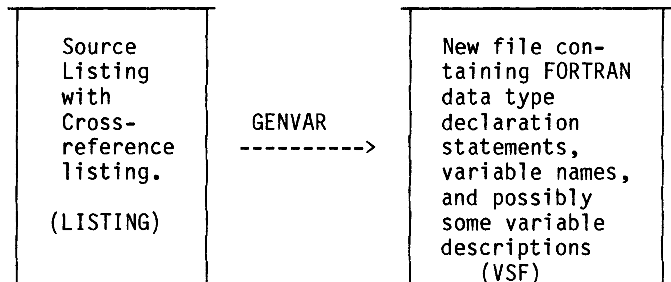
VDF - Variable Definition File. This file contains definitions for each variable used in the system. The definitions includes the variable name, the data type of the variable, what routines use the variable, and a description of how the variable is used. If a variable name is used differently in separate routines, then multiple records will exist in the VDF. This file is essentially a dictionary of variable names.

VLF - Variable List File. This file is generated by GENVAR. It is intended to be edited and then merged with the VDF using program GENVDF to form a new VDF.

Single Routine Mode

This mode is useful for generating the Variables section for one routine when no GNS is being used. GENVAR will get all variables and descriptions from the source file and generate the Variables section, complete with explicit type declarations and comments containing variable names ready for descriptions (if a description for the variable already exists in the source file, then that description will be included here).

Single Routine Mode Processing by GENVAR




LISTING ---> VSF

VSF is generated from LISTING file using GENVAR

Multiple Routine Mode

This mode is useful when a Global Naming Strategy (GNS) is used. The goal is to have all variables defined and described consistently in both source code, and in the Variable Definition File (VDF). Each file used by GENVAR will be discussed, followed by a discussion of how to use GENVAR in GNS mode.

The Variable Definition File (VDF)

The VDF is intended to have descriptions for all variables. Following is a sample record from the VDF.

```
;     TESTNM     GENTST,GETTST,EXTST,PRTTST
CHARACTER*80     Used to hold the name of the test
being performed.
```

The above example illustrates a record for the variable TESTNM which is used to hold the name of a test being performed. TESTNM is a CHARACTER*80 variable and is used for this purpose in routines GENTST, GETTST, EXTST, and PRTTST.

The Variable List File (VLF)

The VLF is generated by GENVAR and contains records for each variable which was used in the SOURCE file but is inconsistent with the VDF (either no definition in the VDF for this variable when used in this routine; or different descriptions in the SOURCE and in the VDF). The VLF should be examined and modified such that it correctly represents the variables used in SOURCE. Then it is merged with the VDF (via program GENVDF) to form a new VDF which contains the new and modified definitions.

GENVAR writes a record to the VLF whenever any of the following conditions occurs for a variable in the LISTING file:

- Different descriptions exists for the variable in LISTING and the VDF for this routine.
- The variable was described in LISTING, but was not described in the VDF.
- The variable was not described in LISTING and was described in the VDF for other routines, but not for this routine.
- The variable was not described anywhere (this causes only the variable name and type to be written to the VLF).

Excluding the final case, each of the above instances would cause one (or more) records containing descriptions of the variables use to be written to the VLF. The final case would cause a record to be written to the VLF, but the record would not contain any description of the variable's uses. The VLF should then be modified and merged with the VDF (using program GENVDF) to form a new VDF.

If more than one record is written to the VLF for one variable, the user must examine the VLF and either select one of the descriptions supplied, or enter a new description. All of the descriptions which are not correct must then be deleted from the VLF. After doing this, the VLF may be merged with the VDF to form a new, current VDF. Program GENVDF will perform this merge. Following is a discussion of the format of the VLF.

The Variables Section File (VSF)

This file is generated by GENVAR to be included in the source code as the Variables section. It contains explicit data type declarations for all variables which should be declared in the Variables section. Furthermore, it contains comments which have descriptions for each variable which was described consistently between the SOURCE file and the VDF. In this context, consistently means that one of the following is true for the variable:

- The variable is described in the SOURCE file, and not described in the VDF for this routine.
- The variable is described in the VDF for this routine and is not described differently in the SOURCE file.

Anytime the second case is not true, a record will be written to the VLF.

The format of this file is the same as the format described for the Variables section in the FORTRAN Coding Standard.

How To Use GNS Mode GENVAR

Typically GENVAR is run twice. The first run generates the VLF which is then modified to correctly describe all of the variables which were used in the source code. Then GENVDF is run to merge the VLF with the VDF. Finally GENVAR is run again. This time all of the variables are described, so the VSF is generated and inserted into the source code. Now all of the variables are described consistently in both the source code and in the VDF. Figure 1 provides a more detailed description of this process. Figure 2 shows how information is moved between the various files by GENVAR.

## SUMMARY

The coding standard has been in use for over a year now. All of the people who have used the standard like it. It is mainly a documentation standard, but we are currently in the process of adding discussion on control structures. The software tools BLDMANUAL, PUTHEAD, and GENROUT have been available for about a year also, and are often used. The GENVAR tool is currently under development. It is anticipated that GENVAR will become the most popular tool for large projects (the idea of a dictionary of variable names is very appealing) when it becomes available.

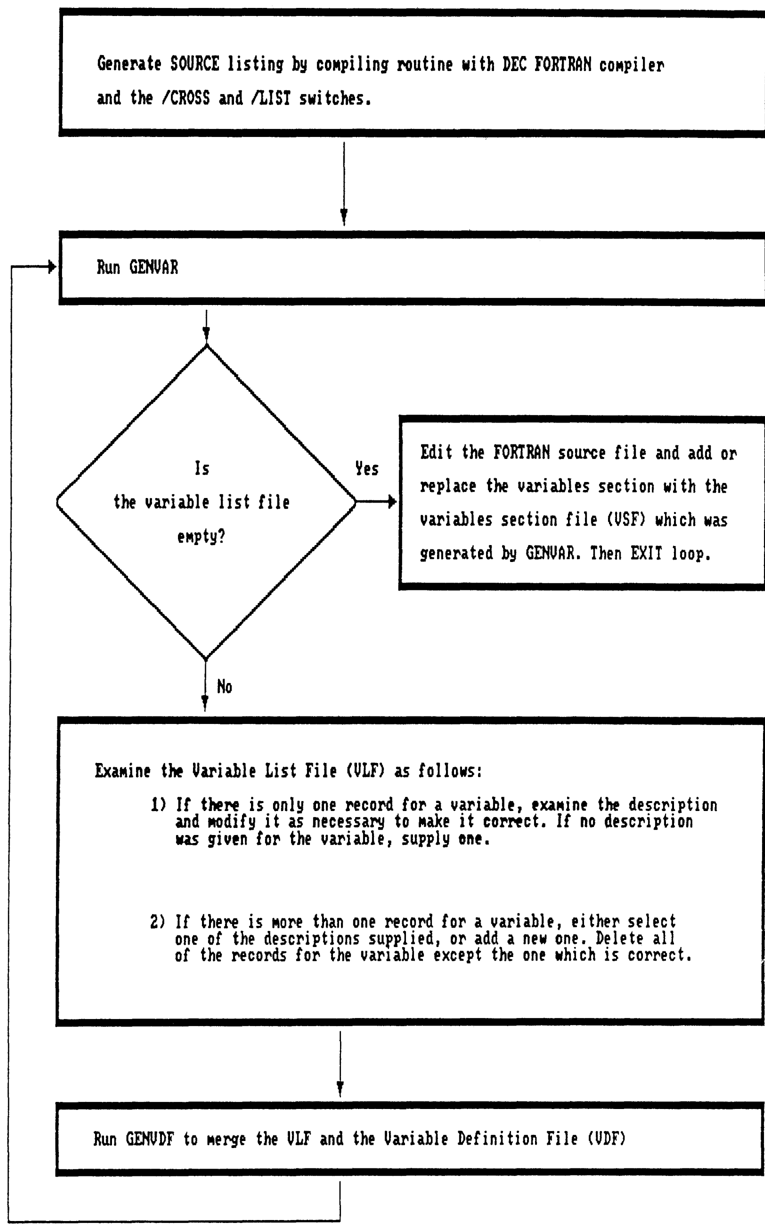
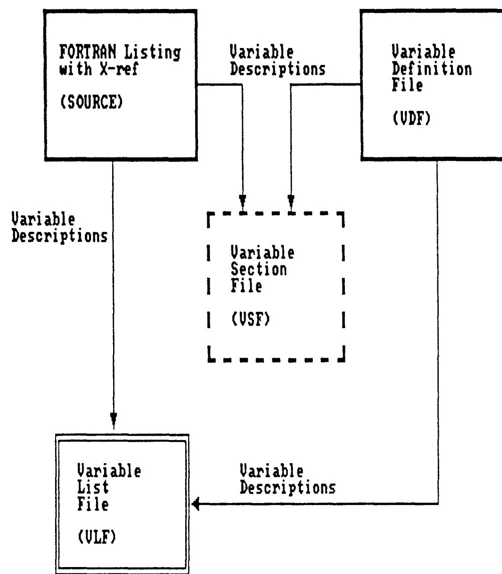



FIGURE 1. How to Use Genvar




| | |
|---|---|
| SOURCE | FORTRAN source listing. Must have been generated by DEC FORTRAN compiler with /CROSS and /LIST switches on. |
| VDF | Variable Definition File. This file contains descriptions for all variables used in the system. |
| VSF | Variables Section File. This file contains data type declarations for all variables to be declared in source. It also has variable descriptions for all variables which were described consistently in the VDF and the SOURCE file. |
| VLF | File with information about variables which are either not in the VDF, or are do not have descriptions in the source, or which have differing information in the VDF and source. |

FIGURE 2. Files Used by GENVAR

# "But That's Impossible in Pascal"

*or*

# Systems Programming in a High-Level Language

**E. W. (Wayne) Sewell**
Software Engineering Specialist
E-Systems, Garland Division
Mail Stop 53730, P. O. Box 660023
Dallas, Texas 75266–0023

### Abstract

Popular wisdom has always held that systems programming can be done only in MACRO or Bliss. While this is true for code running in kernel mode and/or directly accessing the system memory space, it does not automatically follow that it is not possible to use high level languages at all.

This session illustrates techniques for systems programming in Pascal, including utilization of user-written system services and accessing the system memory space. Examples include a simple program to display various system control blocks and portions of code from an Ancillary Control Process (ACP) written in Pascal. While the examples are in Pascal, the techniques illustrated can be used for any high-level language which supports a record structure, pointers, and the VAX standard calling mechanism.

### Special Requirements of Systems Programming

Before we can begin a discussion of how to do systems programming in a high-level language, we first must determine what systems programming is. There are many different definitions for a systems program. Virtually everyone agrees that the VMS Scheduler is a systems program and that a payroll system is an application program, but many other cases are not so clearly cut. For example, a compiler is logically a systems program, but the processing it does, although complex, makes no special demands upon the operating system and can easily be implemented in a high-level language. For the purposes of this discussion, we will ignore the function performed by a program, and instead define a system program to be any program which requires access to the internals of the VMS operating system not provided by the standard libraries (system services, run-time libraries, RMS, etc.). Using this definition, systems programs are those which must do the following:

- Access the system address space, including the nonpaged pool. The system space is simply not accessible from user mode. An attempt to reference a system address from a non-privileged mode has the same result as a reference to a non-existent virtual address—an access violation, followed by termination of the process.
- Access VMS routines in system space. Since they are not addressable, for the reasons mentioned above, they cannot be called by a user mode program.
- Raise IPL (Interrupt Priority Level, not to be confused with the priority of a process). Increasing IPL is a VMS mechanism for synchronization. A program wishing to lock the I/O data base for exclusive access raises its IPL to a predefined level, effectively locking out all other processes in the system attempting to access the data base. Programs in user mode run at IPL 0, and cannot issue the privileged instructions necessary to change IPL.
- Use MUTEXes (MUTual EXclusion—the VMS version of a semaphore). The MUTEX structures themselves and the system routines that manipulate them are in system space.

All of these functions require processing in kernel mode. Unfortunately, much as I hate to admit it, kernel mode processing in a high-level language really *is* impossible. There are too many constraints placed on code which is meant to run in kernel mode. For instance, all references to memory *must* be to valid addresses, because an access violation in kernel mode will crash the entire system instead of just the offending process (in MACRO, the `PROBE` instruction is utilized to test addresses for validity before actually referencing them). Many calls to system routines require the parameters to be passed in registers rather than pushed onto the

stack as specified in the VMS calling standard. In fact, these procedures require that the **JSB** instruction be used rather than the normal **CALLS** or **CALLG**. There are many other special instructions in the VAX architecture which are restricted to usage in kernel mode. The high-level language programmer has very little control over any code generated by the compiler, especially with optimization enabled. This is as it should be—one of the main advantages of using high-level languages is a lack of involvement with the actual machine code, but it does make it difficult to generate the special instructions needed by kernel mode such as **JSB** or **PROBE**.

So why not do systems programming in MACRO and be done with it? The primary reason is that it is tedious to write entire programs in MACRO. Textual output, so easy with the Pascal **WRITELN**, is especially tiresome. Without the inherent benefits of structured programming, MACRO programs tend to be harder to read without conscientious effort on the part of the programmer, and the control mechanisms supplied by a high-level language have to be performed manually. Another problem with MACRO is that some classes of system programs only need kernel mode at distinct points—the rest of the processing can be done in user mode. However, most system programs written in MACRO change to kernel mode when the program begins and remain in that state until termination. During development of such a program, the most trivial of logic errors can crash the entire system.

So what's the best way? My personal preference for all types of programming is to code the main program in a high-level language, supplemented by MACRO subroutines for whatever functions cannot be performed by that language. In the case of a systems program, the MACRO subroutines would exist in a change-mode-to-kernel dispatcher (described below). The subroutines can be made short and modular, and if coded generically they can be reused for other programs performing a similar function.

Systems programs which are likely candidates for high-level languages are programs which need to run in kernel mode only briefly, during distinct phases such as input or output, and most processing can be done in user mode. An example of such a program is an Ancillary Control Process (ACP). Systems programs which are poor candidates for high-level languages are exemplified by device drivers and other code loaded into system space, programs which cannot be implemented as a standard VMS process, or programs which must run in kernel mode most of the time.

### The change-mode-to-kernel dispatcher

A change-mode-to-kernel dispatcher is the VMS mechanism by which a normal user-mode process can execute privileged code. VMS system services are implemented this way. The dispatcher can be considered a special form of subroutine library. The subroutine definition in the calling program is exactly the same as for any other externally-defined subroutine (in Pascal, with the "**EXTERNAL**" designator). As with ordinary entry points in a shareable image, control is passed to a transfer vector rather than the actual entry point of the subroutine. For user mode routines, the transfer vector contains a simple jump instruction to the actual subroutine address. However, vectors for kernel mode routines instead issue a change-mode-to-kernel instruction, specifying the unique index number of the target subroutine. When control is transferred to the subroutine, the instructions are executed in kernel mode with all of the privileges thereof. When the subroutine finishes processing, it issues a normal return (RET) instruction. When control is returned to the calling program, *the process is once again executing in user mode.* The percentage of the time the process spends in kernel mode is minimized; during development, exception conditions crash only the process. After the various routines in the dispatcher are debugged, the system crashes may stop entirely, since these error-free procedures can be reused by the next system program developed and only the new user-mode code must be debugged.

This paper does not discuss the implementation of a change-mode-to-kernel dispatcher; this has been covered in other presentations at various Symposia. For examples of implementation, see the sample dispatcher in **SYS$EXAMPLES:USSDISP.MAR** and the one supplied with this paper, **EXAMPSUBS.MAR**.

The most important advantage to using home-grown system services is that it minimizes the percentage of the code executing in kernel mode. In a given program, some operations can be performed only in kernel mode, but many others can be performed equally well in user mode. For instance, the sample program described below, which does nothing more than dump the PCB of the current process, must access the non-paged pool to get its input data. This can only be done in kernel mode. However, there is no reason for the actual formatting of the PCB fields, which is done with ordinary Pascal write lines, to execute in kernel mode. The kernel mode processing is restricted to the few instructions necessary to get the address of the PCB and copy its contents into normal process memory space.

One drawback to systems programming in Pascal is that the overhead of the change-mode-to-kernel dispatcher is a factor in time-critical code. There is more overhead for a user-written change-mode-to-kernel dispatcher than for a simple procedure call (the amount of overhead is exactly the same as that caused by system service calls, since system services also utilize a dispatcher). If a process is continuously switching between user mode and kernel mode, this technique is not the optimum approach and the all-MACRO version should probably be used.

## The System Control Blocks

The system control blocks are defined in Pascal as standard record types and are referenced with standard Pascal pointer variables. There are two basic types of system control blocks—structured and unstructured. In a structured block, the definition of the first 11 bytes is the same for all block types: the first two longwords are forward and backward pointers, used to place the block into a queue; the word following the queue pointers (at offset 8) contains the actual length of the block in bytes; the byte following the length (at offset 10) contains a binary code uniquely identifying the block type.

As an example of a structured block, the I/O Request Packet (IRP) is defined in Pascal as:

```
irp_ptr= ^irp  ;

irp = packed record
irp$l_ioqfl :  irp_ptr ;
irp$l_ioqbl :  irp_ptr ;
irp$w_size :  word ;
irp$b_type :  byte ;
irp$b_rmod :  byte ;
.
.
.
end ;
```

The unstructured system control blocks have no header because they are never queued (they are referenced by other means) and the length of block is fixed or stored outside of the block. The best known example of an unstructured block is the Process Header (PHD)—the length is fixed, and the PHD address is contained in the PCB.

The major disadvantage of systems programming in Pascal is that code does not track changes to VMS data structures automatically; creating the record layouts for the system control blocks is a manual operation. In MACRO, the definition of the system blocks is provided with VMS, contained in the system macro library `SYS$LIBRARY:LIB.MLB`. When a new version of VMS is received, a new version of the library comes with it, and recompiling and relinking is the only action required to pick up any changes to the definitions of the control blocks (provided there are no major functional changes such as deletion of fields currently in use; in this case, MACRO and high-level language programmers are in the same boat). Pascal, since it is not expected to need this type of information (this is impossible in Pascal, remember?), has no corresponding definitions supplied with the system. The record layout must be edited by hand if a new version of VMS changes the system control blocks. (To minimize this activity, it is wise to place the record layouts in an environment or include file rather than duplicating them in every source file.)

When creating the record layout, it is possible to accelerate the process by dumping the macro definitions contained in the macro library to a temporary file and editing this file rather than starting from scratch. This file can be created by entering:

```
$ library/macro/extract=$xyzdef -
        /out=tempfile.dat       -
        sys$library:lib
```

where `XYZ` is the name of the system definition desired, such as PCB. The result of the above command is a file containing lines such as:

```
$equ pcb$w_size 8
$equ pcb$b_type 10
$equ pcb$b_pri  11
$equ pcb$l_pid  96
```

`$EQU` is the name of the macro which defines a symbol—it can be edited out. The middle column is the symbol itself and the number on the right is the offset (in bytes) of the field from the beginning of the record. The `$L`, `$W`, and `$B` within the name is the size of the field (longword, word, and byte, respectively). After editing, the above text should look like this:

```
pcb$w_size :  word;
pcb$b_type :  byte;
pcb$b_pri :  byte;
pcb$l_pid :  integer;
```

A great deal of care must be taken ensuring that the Pascal records match the system control blocks exactly. For this reason, the PACKED designator must be used on the record statements if fields identified as words or bytes are present. If left unpacked, these fields, which are defined as subranges (`word = 0..65535 ; byte = 0..255`), will be allocated space according to their base type (integer, 32 bits). This will cause all following fields to have incorrect offsets.

There are several places in the various VMS control blocks where privilege masks are used, notably in the PCB. A VMS privilege mask is an array of 64 bits, where each bit set to one represents the ownership of a particular privilege. The array is packed, so the mask is exactly a quadword (8 bytes) in size. In Pascal, privilege masks can be implemented either as a packed array of type boolean or as a set. Either approach can be used as long as the resultant variable is exactly a quadword in size.

A privilege mask defined using the boolean method would be coded like this:

```
type  privmask = packed array [0..63]
                 of boolean ;
```

### SCANPCB—a trivial sample program

The following program provides an example of accessing the non-paged pool. Its sole function is to dump the contents of the process control block (PCB), process header (PHD), and Job Information Block (JIB) of the current process to the standard output file. The fact that most of this information can be more easily obtained via `$GETJPI` is irrelevant, since this is only an example of the technique of accessing system control blocks; most of the other information in the system space *cannot* be acquired by using system services provided by VMS. One important concept illustrated in this example is block chaining. Once we have the address of the PCB, we can easily acquire the address of the other blocks from the associated pointers in the PCB.

#### Declarations

There are three routines contained in the example change-mode-to-kernel dispatcher:

- `GET_MY_PCB`, which simply returns the address of the process control block (PCB) of the current process
- `COPY_SYSTEM_CONTROL_BLOCK`, copies one of the system control blocks from the nonpaged-pool to process memory space or vice versa using the lengths contained within the control blocks themselves—used for structured blocks
- `COPY_NONPAGED_BUFF`, which performs the same copy operation as `COPY_SYSTEM_CONTROL_BLOCK`, except that the lengths are explicitly specified—used for unstructured blocks, which have no embedded length field

To access the fields in the control blocks, we define two pointer variables, a local pointer and a non-paged-pool pointer, for each of the three control blocks used by the program (PCB, JIB, and PHD). The local pointer contains the address of the control block created in the process virtual memory space, a standard Pascal record allocated on the heap and accessed by normal Pascal references. The nonpaged pointer cannot be used as a pointer in the standard way; the block it points to is in the non-paged pool and therefore inaccessible to a user mode program. If it is used as a pointer, an access violation will result and the program will abort. This variable is instead used to simply store the non-paged address until it can be used as one of the parameters to the `COPY_SYSTEM_CONTROL_BLOCK` procedure.

```
var

nonpaged_pcb_pointer,
          local_pcb_pointer
          :  pcb_ptr ;
```

The three procedures in the change-mode-to-kernel dispatcher are defined as Pascal `EXTERNAL` procedures. It is transparent to Pascal whether the procedures called reside in an object library, a shareable image, or a change-mode-to-kernel dispatcher. The calling mechanism is the same—the only concern is ensuring that the parameter definitions agree. Note that the copy-from and copy-to block addresses for the two copy procedures (formal variables `f` and `t`) are defined as integers, although they are actually pointers. These parameters would normally be defined as the pointer type for the block being copied. However, these procedures are called using pointers to three different block types (pcb_ptr, jib_ptr, and phd_ptr). In fact, since these procedures are intended for become the nucleus of a general purpose dispatcher to support a multitude of systems programs, the blocks to be copied could be of *any* type. The problem is that the procedure cannot specify multiple types on a formal parameter definition (actually, there is a method to accomplish this using variant records, but it can become very messy). The solution is to define the parameter as a neutral type of the same size as a pointer, such as an integer, and then using the type cast operator (::), which allows a temporary override of the rigid type checking of Pascal, when the procedure is actually called.

```
function copy_system_control_block
          (%immed f , t :  integer)     :
          integer ; extern ;


function copy_nonpaged_buff
          (%immed flen, tlen:  integer;
          %immed f , t :  integer)
          :  integer ; extern ;


function get_my_pcb
          (var p :  pcb_ptr)
          :  integer ; extern ;
```

#### SCANPCB processing

The first action performed by the SCANPCB program is to get the pointer to the current PCB, which is located in the nonpaged pool. At the same time, a local PCB is allocated in the normal process address space using the Pascal `NEW` function. The contents of the local PCB are initially undefined.

```
return_code := get_my_pcb
       (nonpaged_pcb_pointer) ;
new(local_pcb_pointer) ;
```

After the two pointer variables are loaded the system control block copy routine (the one used for structured blocks) is used to copy the inaccessible real PCB to the local duplicate. The type cast operator (::) is used to temporarily redefine `nonpaged_pcb_pointer` and `local_pcb_pointer` from type `pcb_ptr` to integer, which is the type specified on the formal parameters in the

procedure definition, without affecting these variables elsewhere in the program.

```
return_code := copy_system_control_block
        (nonpaged_pcb_pointer :: integer,
        local_pcb_pointer :: integer) ;
```

Now that the local PCB is a mirror-image of the one in the nonpaged pool, the fields within it can be accessed via normal Pascal mechanisms. In this sample program, we simply print the fields in decimal or hex on the standard output device.

```
with local_pcb_pointer^ do
   writeln ('pcb$l_pid ' ,
            '(process id) = ' ,
         hex (pcb$l_pid)) ;
```

The individual privileges in a privilege mask can easily be accessed as boolean variables in the packed array of bits.

```
with local_pcb_pointer^ do
   if pcb$q_priv[prv$v_oper] then
      writeln('this process has oper priv');
```

Since the field PCB$L_JIB contains the address of the JIB in the non-paged pool, we can initialize the non-paged pointer to it with a simple Pascal assignment statement.

```
nonpaged_jib_pointer :=
        local_pcb_pointer^.pcb$l_jib ;
```

Now that we have the address of the JIB, we can use COPY_SYSTEM_CONTROL_BLOCK again to create a local JIB and print its contents as we did for the PCB. The procedure to dump the PHD is the same except that COPY_NONPAGED_BUFF is used, since the PHD is an unstructured block.

### Procedures in the dispatcher

Within the change-mode-to-kernel dispatcher itself, the following MACRO code returns the address of the Process Control Block of the current process:

```
$ent     get_my_pcb
movl     r4,@4(ap)
movl     #ss$_normal,r0
ret
```

One of the key points illustrated by this subroutine is its brevity, which is enhanced by the fact that the address of the current PCB is preloaded into register 4 by the dispatcher; however, the active PCB can also be found via the absolute memory location **sch$gl_curpcb**.

The following code copies a system control block from the process virtual address space to system space or vice versa. The lengths of the blocks are contained within them.

```
$ent     copy_system_control_block
movl     4(ap),r2
movl     8(ap),r3
movc5    pcb$w_size(r2),(r2),#0, -
               pcb$w_size(r3),(r3)
movl     #ss$_normal,r0
ret
```

This subroutine is almost as short as the first one. It would be slightly longer in practice, since for clarity this sample routine does not include any validity checking of the input parameters, which would be *essential* in a real routine. The other block-copy routine, COPY_NONPAGED_BUFF, is not shown here because it is virtually identical to COPY_SYSTEM_CONTROL_BLOCK; the only difference is that the lengths of the two blocks are passed as parameters instead of being retrieved from the blocks themselves.

### A Pascal ACP

A VMS device driver operates in a limited context. The driver code does not execute as a normal process, and therefore has no user memory space, no event flags, no global sections, etc. It cannot issue system services, receive AST's, hibernate, or wait for event flags. Sometimes the functions being performed by a driver are complex enough to require one of these forbidden operations. Or, one I/O operation may cause several lower-level operations to occur, requiring the driver to issue I/O requests to itself or another driver via the QIO mechanism. An example of this type of operation would be a disk read of a virtual block, which would be transformed to multiple logical and physical operations (SEEK the physical block, READ the physical block, etc.).

An Ancillary Control Process (ACP) is a standard VMS mechanism to expand the capabilities of a device driver. It can be considered an extension of the driver to handle functions which cannot be performed easily by the driver. Since the ACP is a full VMS process, it can hibernate, issue system services, and do all of the other things a driver cannot.

The ACP described here is used with a simulated terminal driver, similar in function to the Decnet REMACP. It allows remote login over a proprietary local area network using non-VT100 terminals.

The functions performed by the ACP include:

- Creating and initializing the ACP Queue Block (AQB) and Volume Control Block (VCB) in the non-paged pool.
- Reading the I/O Request Packets (IRPs)
- Posting the I/O complete

### Declarations

In addition to the two block-copy routines listed above for the example program, the following additional routines are contained in the change-mode-to-kernel dispatcher for the ACP:

- GET_IRP_FROM_QUEUE, which gets the next IRP for processing
- POST_IO_COMPLETION, which places an IRP which has returned from the link into the I/O postprocessing queue
- ALLOCATE_ACP_QUEUE_BLOCK, which is used during process initialization
- CREATE_NEW_UCB and KILL_UCB, which are used to create and delete virtual terminals

### Initializing the AQB

This section of code allocates and initializes an ACP Queue Block (AQB). The ALLOCATE ACP QUEUE_BLOCK procedure is similar to the GET IRP FROM QUEUE procedure, in that it returns the address of a block in the non-paged pool, except that in this case, the block is actually created by the procedure.

```
return_code :=
     allocate_acp_queue_block
           (nonpaged_aqb_ptr);
```

All of the fields of the local block must be initialized, since this block will be copied to the pool to become the real AQB.

```
new(local_aqb_ptr);
with local_aqb_ptr^ do begin
     aqb$w_size := aqb$c_length;
     aqb$b_type := dyn$c_aqb;
     .
     .
     .
```

As before, the COPY_SYSTEM_CONTROL_BLOCK procedure is used. However, this time the local block is copied *to* the non-paged pool.

```
return_code :=
     copy_system_control_block
           (local_aqb_ptr :: integer;
           nonpaged_aqb_ptr :: integer);
```

### The main program

The following code is the main processing loop of the ACP; during each iteration, it either gets another IRP from the ACP queue or another Remote Data Packet (RDP) returning from the link. If either process_link_data or process_request_packet is executed during an iteration, the io_performed flag is set. If neither is executed, then the process will hibernate until new data is received from either side of the interface and a $WAKE is issued.

```
repeat
     io_performed := false;
     if link_data_present
          then process_link_data;
     if irp_present
          then process_request_packet;
     if not io_performed  then
          hibernate_until_new_input;
until dead;
```

### Reading the IRPs

I/O Request Packets (IRPs) are passed from the simulated terminal driver to the ACP via a linked list (queue) in the non-paged pool. Since the user-mode ACP cannot access the pool directly, a subroutine in the dispatcher is used to get the address of the next IRP in the queue.

```
get_irp_from_queue(nonpaged_aqb_ptr,
                   nonpaged_irp_ptr);
```

Using a technique similar to that of the PCB processing described earlier, the ACP creates a local IRP in the normal program space and uses the COPY_SYSTEM_CONTROL_BLOCK procedure to copy the inaccessible real IRP to the local duplicate.

```
new(local_irp_ptr);

return_code :=
     copy_system_control_block
          (nonpaged_irp_ptr ::  integer,
          local_irp_ptr ::  integer);
```

The I/O buffer associated with the IRP, which must be transmitted across the link and is in the form of a Remote Data Packet (RDP), must also be copied from the non-paged pool to the process address space. Since the RDP is not a structured block, the COPY_NONPAGED_BUFF procedure must be used and the actual length of the buffer must be specified (the address and length are contained in the IRP fields irp$l_svapte and irp$w_boff).

```
with local_irp_ptr^ do begin
     from_ptr := irp$l_svapte;
     to_ptr := local_io_buff_ptr;
     return_code := copy_nonpaged_buff
          (irp$w_boff,irp$w_boff,
          from_ptr :: integer,
          to_ptr :: integer);
```

Since multiple I/O operations may be in progress simultaneously and they may not complete in the same order they were initiated, we must save the IRP in a local table, identified by sequence number (this number is guaranteed by VMS to be unique). The RDP is then transmitted to the remote system where the I/O will actually be performed.

```
put_irp_on_local_queue(irp$l_seqnum);
send_rdp_across_link;
```

### Posting the I/O complete

When the response RDP returns from the link, now in a different format consisting of the completion code of the I/O and possibly containing data (in the case of a read operation), the original QIO must be posted complete. Using the sequence number, we locate the IRP associated with this RDP. COPY_NONPAGED_BUFF is again used, this time to overlay the original RDP in the system space with the response RDP just received (the block was originally allocated by the driver to be large enough for either form of an RDP, so we do not have to worry about overwriting system space).

Finally, we call a routine in the dispatcher to post the I/O operation complete.

```
return_code :=
    post_io_completion(nonpaged_irp_ptr,
    nonpaged_ucb,
    interrupt_routine_address);
```

### Procedures in the dispatcher

The following code in the change-mode-to-kernel dispatcher for the ACP returns the address of the next IRP in the ACP queue, if one is present; if the queue is empty, a warning-level return code is issued.

```
        $ent    get_irp_from_queue
        movl    4(ap),r5
        remque  @(r5),@8(ap)
        bvc     10$
        movl    #queue_is_empty,r0
        ret
10$:    movl    #ss$_normal,r0
        ret
```

It cannot be over-emphasized that the process is only in kernel mode for the amount of time needed to execute these tiny procedures in the dispatcher. The time window where the VAX is vulnerable to a full-system crash is very small, and procedures this size are comparatively easy to debug.

## Conclusions

- Some classes of systems programs *can* be implemented in a high-level language
- Systems programs can gain the benefits of structured programming and can be made more readable
- Development time can be shortened
- System crashes can be reduced during development
- Kernel mode routines (in a change-mode-to-kernel dispatcher) developed for one system program can be used again for another

```
program scanpcb (output);

const

    prv$v_cmkrnl = 0;
    prv$v_cmexec = 1;
    prv$v_sysnam = 2;
    prv$v_grpnam = 3;
    prv$v_allspool = 4;
    prv$v_detach = 5;
    prv$v_diagnose = 6;
    prv$v_log_io = 7;
    prv$v_group = 8;
    prv$v_acnt = 9;
    prv$v_prmceb = 10;
    prv$v_prmmbx = 11;
    prv$v_pswapm = 12;
    prv$v_altpri = 13;
    prv$v_setpri = 13;
    prv$v_setprv = 14;
    prv$v_tmpmbx = 15;
    prv$v_world = 16;
    prv$v_mount = 17;
    prv$v_oper = 18;
    prv$v_exquota = 19;
    prv$v_netmbx = 20;
    prv$v_volpro = 21;
    prv$v_phy_io = 22;
    prv$v_bugchk = 23;
    prv$v_prmgbl = 24;
    prv$v_sysgbl = 25;
    prv$v_pfnmap = 26;
    prv$v_shmem = 27;
    prv$v_sysprv = 28;
    prv$v_bypass = 29;
    prv$v_syslck = 30;
    prv$v_share = 31;
    prv$v_upgrade = 32;
    prv$v_downgrade = 33;
    prv$v_grpprv = 34;
    prv$v_readall = 35;
    prv$v_security = 38;


type

    privmask = packed array [0..63]
        of boolean;
    pack7 = packed array [1..7] of char;
    pack15 = packed array [1..15] of char;
    pack20 = packed array [1..20] of char;
    word = 0..65535;
    byte = 0..255;

pcb_ptr = ^pcb;
phd_ptr = ^phd;
jib_ptr = ^jib;

phd = packed record
phd$q_privmsk : privmask;
phd$w_wslist : word;
phd$w_wsauth : word;
phd$w_wslock : word;
phd$w_wsdyn : word;
phd$w_wsnext : word;
phd$w_wslast : word;
phd$w_wsauthext : word;
phd$w_wsextent : word;
phd$w_wsquota : word;
phd$w_dfwscnt : word;
phd$l_pagfil : integer;
phd$l_pstbasoff : integer;
phd$w_pstlast : word;
phd$w_pstfree : word;
phd$l_frep0va : integer;
phd$l_freptecnt : integer;
phd$l_frep1va : integer;
phd$b_dfpfc : byte;
phd$b_pgtbpfc : byte;
phd$w_flags : word;
phd$l_cputim : integer;
phd$w_quant : word;
phd$w_prclm : word;
phd$w_astlm : word;
phd$w_phvindex : word;
phd$l_bak : integer;
phd$l_pstbasmax : integer;
phd$l_pageflts : integer;
phd$w_wssize : word;
phd$w_swapsize : word;
phd$l_diocnt : integer;
phd$l_biocnt : integer;
phd$l_cpulim : integer;
phd$b_cpumode : byte;
phd$b_awsmode : byte;
phd$w_fill_30 : word;
phd$l_ptwslelck : integer;
phd$l_ptwsleval : integer;
phd$w_ptcntlck : word;
phd$w_ptcntval : word;
phd$w_ptcntact : word;
phd$w_ptcntmax : word;
phd$w_wsfluid : word;
phd$w_extdynws : word;
phd$l_ksp : integer;
phd$l_esp : integer;
phd$l_ssp : integer;
```

```
phd$l_usp : integer;
phd$l_r0 : integer;
phd$l_r1 : integer;
phd$l_r2 : integer;
phd$l_r3 : integer;
phd$l_r4 : integer;
phd$l_r5 : integer;
phd$l_r6 : integer;
phd$l_r7 : integer;
phd$l_r8 : integer;
phd$l_r9 : integer;
phd$l_r10 : integer;
phd$l_r11 : integer;
phd$l_r12 : integer;
phd$l_r13 : integer;
phd$l_pc : integer;
phd$l_psl : integer;
phd$l_p0br : integer;
phd$l_p0lrastl : integer;
phd$l_p1br : integer;
phd$l_p1lr : integer;
phd$w_emptpg : word;
phd$w_respgcnt : word;
phd$w_reqpgcnt : word;
phd$w_cwslx : word;
phd$q_authpriv : privmask;
phd$q_imagpriv : privmask;
phd$l_reslsth : integer;
phd$l_imgcnt : integer;
phd$l_pfltrate : integer;
phd$l_pflref : integer;
phd$l_timref : integer;
phd$l_mpinhibit : integer;
phd$l_pgfltio : integer;
phd$b_authpri : byte;
phd$b_fill1 : byte;
phd$w_fill2 : word;
phd$l_extracpu : integer;
end;

jib = packed record
jib$l_mtlfl : integer;
jib$l_mtlbl : integer;
jib$w_size : word;
jib$b_type : byte;
jib$b_daytypes : byte;
jib$t_username : pack20;
jib$l_bytcnt : integer;
jib$l_bytlm : integer;
jib$l_pbytcnt : integer;
jib$l_pbytlim : integer;
jib$w_filcnt : word;
jib$w_fillm : word;
jib$w_tqcnt : word;
jib$w_tqlm : word;
jib$l_pgflquota : integer;
jib$l_pgflcnt : integer;
jib$l_cpulim : integer;
jib$w_prccnt : word;
jib$w_prclim : word;
jib$w_shrfcnt : word;
jib$w_shrflim : word;
jib$w_enqcnt : word;
jib$w_enqlm : word;
jib$w_maxjobs : word;
jib$w_maxdetach : word;
jib$l_mpid : integer;
jib$l_jlnamfl : integer;
jib$l_jlnambl : integer;
jib$l_pdayhours : integer;
jib$l_odayhours : integer;
jib$b_jobtype : byte;
jib$b_fill1 : byte;
jib$w_fill2 : word;
jib$l_org_bytlm : integer;
jib$l_org_pbytlm : integer;
end;

pcb = packed record
pcb$l_sqfl : pcb_ptr;
pcb$l_sqbl : pcb_ptr;
pcb$w_size : word;
pcb$b_type : byte;
pcb$b_pri : byte;
pcb$b_astact : byte;
pcb$b_asten : byte;
pcb$w_mtxcnt : word;
pcb$l_astqfl : integer;
pcb$l_astqbl : integer;
pcb$l_phypcb : integer;
pcb$l_owner : integer;
pcb$l_wsswp : integer;
pcb$l_sts : integer;
pcb$b_prisav,pcb$b_pribsav,
    pcb$b_dpc,pcb$b_authpri :
    byte;
pcb$w_state : word;
pcb$b_wefc : byte;
pcb$b_prib : byte;
pcb$w_aptcnt : word;
pcb$w_tmbu : word;
pcb$w_gpgcnt : word;
pcb$w_ppgcnt : word;
pcb$w_astcnt : word;
pcb$w_biocnt : word;
pcb$w_biolm : word;
```

```
    pcb$w_diocnt : word;
    pcb$w_diolm : word;
    pcb$w_prccnt : word;
    pcb$t_termlen : byte;
    pcb$t_terminal : pack7;
    pcb$l_efwm : integer;
    pcb$l_efcs : integer;
    pcb$l_efcu : integer;
    pcb$l_efc2p : integer;
    pcb$l_efc3p : integer;
    pcb$l_pid : integer;
    pcb$l_epid : integer;
    pcb$l_eowner : integer;
    pcb$l_phd : phd_ptr;
    pcb$b_lnamelen : byte;
    pcb$t_lname : pack15;
    pcb$l_jib : jib_ptr;
    pcb$q_priv : privmask;
    pcb$l_arb : integer;
    pcb$l_unknown1 : packed array [1..11]
        of integer;
    pcb$w_mem,pcb$w_grp : word;
    pcb$l_unknown2 : packed array [1..15]
        of integer;
    pcb$l_aclfl : integer;
    pcb$l_aclbl : integer;
    pcb$l_lockqfl : integer;
    pcb$l_lockqbl : integer;
    pcb$l_dlckpri : integer;
    pcb$l_ipast : integer;
    pcb$l_defprot : integer;
    pcb$l_waitime : integer;
    pcb$l_pmb : integer;
    end;

var


return_code : integer;
pcb_size : integer;
nonpaged_pcb_pointer,
    local_pcb_pointer : pcb_ptr;
jib_size : integer;
nonpaged_jib_pointer,
    local_jib_pointer : jib_ptr;
phd_size : integer;
nonpaged_phd_pointer,
    local_phd_pointer : phd_ptr;
privnames :
    array [prv$v_cmkrnl..prv$v_security]
    of varying [30] of char;

procedure sys$exit (%immed rc : integer);
           extern;

function copy_nonpaged_buff
           (%immed fl,tl : integer;
           %immed f,t : integer)
           : integer; extern;

function copy_system_control_block
           (%immed f,t : integer) :
           integer; extern;

function get_my_pcb (var p : pcb_ptr)
           : integer; extern;

procedure print_privs (privmask : privmask);

var

 n : integer;

begin
for n := 0 to prv$v_security do
    if privmask[n] then
        writeln(privnames[n]);
end;

procedure print_pcb;

begin
with local_pcb_pointer^ do begin
    writeln('pcb$l_sqfl = ',
        hex(pcb$l_sqfl));
    writeln('pcb$l_sqbl = ',
        hex(pcb$l_sqbl));
    writeln('pcb$w_size = ',pcb$w_size);
    writeln('pcb$b_type = ',pcb$b_type);
    writeln('pcb$b_pri = ',pcb$b_pri);
    writeln('pcb$b_astact = ',pcb$b_astact);
    writeln('pcb$b_asten = ',pcb$b_asten);
    writeln('pcb$w_mtxcnt = ',pcb$w_mtxcnt);
    writeln('pcb$l_astqfl = ',
        hex(pcb$l_astqfl));
    writeln('pcb$l_astqbl = ',
        hex(pcb$l_astqbl));
    writeln('pcb$l_phypcb = ',
        hex(pcb$l_phypcb));
    writeln('pcb$l_owner = ',pcb$l_owner);
    writeln('pcb$l_wsswp = ',
        hex(pcb$l_wsswp));
    writeln('pcb$l_sts = ',hex(pcb$l_sts));
    writeln('pcb$b_prisav = ',pcb$b_prisav);
```

```
writeln('pcb$b_pribsav = ',
    pcb$b_pribsav);
writeln('pcb$b_dpc = ',pcb$b_dpc);
writeln('pcb$b_authpri = ',
    pcb$b_authpri);
writeln('pcb$w_state = ',pcb$w_state);
writeln('pcb$b_wefc = ',
    hex(pcb$b_wefc));
writeln('pcb$b_prib = ',pcb$b_prib);
writeln('pcb$w_aptcnt = ',pcb$w_aptcnt);
writeln('pcb$w_tmbu = ',pcb$w_tmbu);
writeln('pcb$w_gpgcnt = ',pcb$w_gpgcnt);
writeln('pcb$w_ppgcnt = ',pcb$w_ppgcnt);
writeln('pcb$w_astcnt = ',pcb$w_astcnt);
writeln('pcb$w_biocnt = ',pcb$w_biocnt);
writeln('pcb$w_biolm = ',pcb$w_biolm);
writeln('pcb$w_diocnt = ',pcb$w_diocnt);
writeln('pcb$w_diolm = ',pcb$w_diolm);
writeln('pcb$w_prccnt = ',pcb$w_prccnt);
writeln('pcb$t_terminal (length) = ',
    pcb$t_termlen);
writeln('pcb$t_terminal = ',
    pcb$t_terminal);
writeln('pcb$l_efwm = ',
    hex(pcb$l_efwm));
writeln('pcb$l_efcs = ',
    hex(pcb$l_efcs));
writeln('pcb$l_efcu = ',
    hex(pcb$l_efcu));
writeln('pcb$l_efc2p = ',
    hex(pcb$l_efc2p));
writeln('pcb$l_efc3p = ',
    hex(pcb$l_efc3p));
writeln('pcb$l_pid = ',hex(pcb$l_pid));
writeln('pcb$l_epid = ',
    hex(pcb$l_epid));
writeln('pcb$l_phd = ',hex(pcb$l_phd));
writeln('pcb$l_eowner = ',
    hex(pcb$l_eowner));
writeln('pcb$t_lname = ',pcb$t_lname);
writeln('pcb$l_jib = ',hex(pcb$l_jib));
writeln('pcb$q_priv = ');
print_privs (pcb$q_priv);
writeln('pcb$l_arb = ',hex(pcb$l_arb));
writeln('pcb$w_mem = ',oct(pcb$w_mem));
writeln('pcb$w_grp = ',oct(pcb$w_grp));
writeln('pcb$l_aclfl = ',
    hex(pcb$l_aclfl));
writeln('pcb$l_aclbl = ',
    hex(pcb$l_aclbl));
writeln('pcb$l_lockqfl = ',
    hex(pcb$l_lockqfl));
writeln('pcb$l_lockqbl = ',
        hex(pcb$l_lockqbl));
    writeln('pcb$l_dlckpri = ',
        pcb$l_dlckpri);
    writeln('pcb$l_ipast = ',pcb$l_ipast);
    writeln('pcb$l_defprot = ',
        hex(pcb$l_defprot));
    writeln('pcb$l_waitime = ',
        hex(pcb$l_waitime));
    writeln('pcb$l_pmb = ',hex(pcb$l_pmb));
    writeln;
    writeln;
    end;

end;



procedure print_jib;

begin
with local_jib_pointer^ do begin
    writeln('jib$l_mtlfl = ',
        hex(jib$l_mtlfl));
    writeln('jib$l_mtlbl = ',
        hex(jib$l_mtlbl));
    writeln('jib$w_size = ',jib$w_size);
    writeln('jib$b_type = ',jib$b_type);
    writeln('jib$b_daytypes = ',
        hex(jib$b_daytypes));
    writeln('jib$T_username = ',
        jib$t_username);
    writeln('jib$l_bytcnt = ',jib$l_bytcnt);
    writeln('jib$l_bytlm = ',jib$l_bytlm);
    writeln('jib$l_pbytcnt = ',
        jib$l_pbytcnt);
    writeln('jib$l_pbytlim = ',
        jib$l_pbytlim);
    writeln('jib$w_filcnt = ',jib$w_filcnt);
    writeln('jib$w_fillm = ',jib$w_fillm);
    writeln('jib$w_tqcnt = ',jib$w_tqcnt);
    writeln('jib$w_tqlm = ',jib$w_tqlm);
    writeln('jib$l_pgflquota = ',
        jib$l_pgflquota);
    writeln('jib$l_pgflcnt = ',
        jib$l_pgflcnt);
    writeln('jib$l_cpulim = ',jib$l_cpulim);
    writeln('jib$w_prccnt = ',jib$w_prccnt);
    writeln('jib$w_prclim = ',jib$w_prclim);
    writeln('jib$w_shrfcnt = ',
        jib$w_shrfcnt);
    writeln('jib$w_shrflim = ',
        jib$w_shrflim);
    writeln('jib$w_enqcnt = ',jib$w_enqcnt);
```

```
    writeln('jib$w_enqlm = ',jib$w_enqlm);
    writeln('jib$w_maxjobs = ',
        jib$w_maxjobs);
    writeln('jib$w_maxdetach = ',
        jib$w_maxdetach);
    writeln('jib$l_mpid = ',
        hex(jib$l_mpid));
    writeln('jib$l_jlnamfl = ',
        hex(jib$l_jlnamfl));
    writeln('jib$l_jlnambl = ',
        hex(jib$l_jlnambl));
    writeln('jib$l_pdayhours = ',
        hex(jib$l_pdayhours));
    writeln('jib$l_odayhours = ',
        hex(jib$l_odayhours));
    writeln('jib$b_jobtype = ',
        hex(jib$b_jobtype));
    writeln('jib$l_org_bytlm = ',
        jib$l_org_bytlm);
    writeln('jib$l_org_pbytlm = ',
        jib$l_org_pbytlm);
    writeln;
    writeln;
    end;

end;

procedure print_phd;
begin
with local_phd_pointer^ do begin
    writeln('phd$q_privmsk = ');
    print_privs (phd$q_privmsk);
    writeln('phd$w_wslist = ',phd$w_wslist);
    writeln('phd$w_wsauth = ',phd$w_wsauth);
    writeln('phd$w_wslock = ',phd$w_wslock);
    writeln('phd$w_wsdyn = ',phd$w_wsdyn);
    writeln('phd$w_wsnext = ',phd$w_wsnext);
    writeln('phd$w_wslast = ',phd$w_wslast);
    writeln('phd$w_wsauthext = ',
        phd$w_wsauthext);
    writeln('phd$w_wsextent = ',
        phd$w_wsextent);
    writeln('phd$w_wsquota = ',
        phd$w_wsquota);
    writeln('phd$w_dfwscnt = ',
        phd$w_dfwscnt);
    writeln('phd$l_pagfil = ',
        hex(phd$l_pagfil));
    writeln('phd$l_pstbasoff = ',
        hex(phd$l_pstbasoff));
    writeln('phd$w_pstlast = ',
        hex(phd$w_pstlast));
    writeln('phd$w_pstfree = ',
        hex(phd$w_pstfree));
    writeln('phd$l_frep0va = ',
        hex(phd$l_frep0va));
    writeln('phd$l_freptecnt = ',
        hex(phd$l_freptecnt));
    writeln('phd$l_frep1va = ',
        hex(phd$l_frep1va));
    writeln('phd$b_dfpfc = ',
        hex(phd$b_dfpfc));
    writeln('phd$b_pgtbpfc = ',
        hex(phd$b_pgtbpfc));
    writeln('phd$w_flags = ',
        hex(phd$w_flags));
    writeln('phd$l_cputim = ',phd$l_cputim);
    writeln('phd$w_quant = ',
        hex(phd$w_quant));
    writeln('phd$w_prclm = ',phd$w_prclm);
    writeln('phd$w_astlm = ',phd$w_astlm);
    writeln('phd$w_phvindex = ',
        phd$w_phvindex);
    writeln('phd$l_bak = ',hex(phd$l_bak));
    writeln('phd$l_pstbasmax = ',
        hex(phd$l_pstbasmax));
    writeln('phd$l_pageflts = ',
        phd$l_pageflts);
    writeln('phd$w_wssize = ',phd$w_wssize);
    writeln('phd$w_swapsize = ',
        phd$w_swapsize);
    writeln('phd$l_diocnt = ',phd$l_diocnt);
    writeln('phd$l_biocnt = ',phd$l_biocnt);
    writeln('phd$l_cpulim = ',phd$l_cpulim);
    writeln('phd$b_cpumode = ',
        phd$b_cpumode);
    writeln('phd$b_awsmode = ',
        hex(phd$b_awsmode));
    writeln('phd$w_fill_30 = ',
        hex(phd$w_fill_30));
    writeln('phd$l_ptwslelck = ',
        phd$l_ptwslelck);
    writeln('phd$l_ptwsleval = ',
        phd$l_ptwsleval);
    writeln('phd$w_ptcntlck = ',
        phd$w_ptcntlck);
    writeln('phd$w_ptcntval = ',
        phd$w_ptcntval);
    writeln('phd$w_ptcntact = ',
        phd$w_ptcntact);
    writeln('phd$w_ptcntmax = ',
        phd$w_ptcntmax);
    writeln('phd$w_wsfluid = ',
        phd$w_wsfluid);
    writeln('phd$w_extdynws = ',
        hex(phd$w_extdynws));
```

```
writeln('phd$l_ksp = ',hex(phd$l_ksp));
writeln('phd$l_esp = ',hex(phd$l_esp));
writeln('phd$l_ssp = ',hex(phd$l_ssp));
writeln('phd$l_usp = ',hex(phd$l_usp));
writeln('phd$l_r0 = ',hex(phd$l_r0));
writeln('phd$l_r1 = ',hex(phd$l_r1));
writeln('phd$l_r2 = ',hex(phd$l_r2));
writeln('phd$l_r3 = ',hex(phd$l_r3));
writeln('phd$l_r4 = ',hex(phd$l_r4));
writeln('phd$l_r5 = ',hex(phd$l_r5));
writeln('phd$l_r6 = ',hex(phd$l_r6));
writeln('phd$l_r7 = ',hex(phd$l_r7));
writeln('phd$l_r8 = ',hex(phd$l_r8));
writeln('phd$l_r9 = ',hex(phd$l_r9));
writeln('phd$l_r10 = ',hex(phd$l_r10));
writeln('phd$l_r11 = ',hex(phd$l_r11));
writeln('phd$l_r12 = ',hex(phd$l_r12));
writeln('phd$l_r13 = ',hex(phd$l_r13));
writeln('phd$l_pc = ',hex(phd$l_pc));
writeln('phd$l_psl = ',hex(phd$l_psl));
writeln('phd$l_p0br = ',
    hex(phd$l_p0br));
writeln('phd$l_p0lrastl = ',
    hex(phd$l_p0lrastl));
writeln('phd$l_p1br = ',
    hex(phd$l_p1br));
writeln('phd$l_p1lr = ',
    hex(phd$l_p1lr));
writeln('phd$w_emptpg = ',
    hex(phd$w_emptpg));
writeln('phd$w_respgcnt = ',
    phd$w_respgcnt);
writeln('phd$w_reqpgcnt = ',
    phd$w_reqpgcnt);
writeln('phd$w_cwslx = ',
    hex(phd$w_cwslx));
writeln('phd$q_authpriv = ');
print_privs (phd$q_authpriv);
writeln('phd$q_imagpriv = ');
print_privs (phd$q_imagpriv);
writeln('phd$l_reslsth = ',
    phd$l_reslsth);
writeln('phd$l_imgcnt = ',phd$l_imgcnt);
writeln('phd$l_pfltrate = ',
    phd$l_pfltrate);
writeln('phd$l_pflref = ',phd$l_pflref);
writeln('phd$l_timref = ',phd$l_timref);
writeln('phd$l_mpinhibit = ',
    hex(phd$l_mpinhibit));
writeln('phd$l_pgfltio = ',
    phd$l_pgfltio);
writeln('phd$b_authpri = ',
    phd$b_authpri);
    writeln('phd$l_extracpu = ',
        phd$l_extracpu);
    writeln;
    writeln;
    end;

end;



procedure init_priv_names;

var

 n : integer;

begin
privnames[prv$v_cmkrnl] := 'cmkrnl';
privnames[prv$v_cmexec] := 'cmexec';
privnames[prv$v_sysnam] := 'sysnam';
privnames[prv$v_grpnam] := 'grpnam';
privnames[prv$v_allspool] := 'allspool';
privnames[prv$v_detach] := 'detach';
privnames[prv$v_diagnose] := 'diagnose';
privnames[prv$v_log_io] := 'log_io';
privnames[prv$v_group] := 'group';
privnames[prv$v_acnt] := 'acnt';
privnames[prv$v_prmceb] := 'prmceb';
privnames[prv$v_prmmbx] := 'prmmbx';
privnames[prv$v_pswapm] := 'pswapm';
privnames[prv$v_altpri] := 'altpri';
privnames[prv$v_setpri] := 'setpri';
privnames[prv$v_setprv] := 'setprv';
privnames[prv$v_tmpmbx] := 'tmpmbx';
privnames[prv$v_world] := 'world';
privnames[prv$v_mount] := 'mount';
privnames[prv$v_oper] := 'oper';
privnames[prv$v_exquota] := 'exquota';
privnames[prv$v_netmbx] := 'netmbx';
privnames[prv$v_volpro] := 'volpro';
privnames[prv$v_phy_io] := 'phy_io';
privnames[prv$v_bugchk] := 'bugchk';
privnames[prv$v_prmgbl] := 'prmgbl';
privnames[prv$v_sysgbl] := 'sysgbl';
privnames[prv$v_pfnmap] := 'pfnmap';
privnames[prv$v_shmem] := 'shmem';
privnames[prv$v_sysprv] := 'sysprv';
privnames[prv$v_bypass] := 'bypass';
privnames[prv$v_syslck] := 'syslck';
privnames[prv$v_share] := 'share';
privnames[prv$v_upgrade] := 'upgrade';
privnames[prv$v_downgrade] := 'downgrade';
privnames[prv$v_grpprv] := 'grpprv';
```

```
privnames[prv$v_readall] := 'readall';
privnames[prv$v_security] := 'security';

end;



begin
init_priv_names;
return_code := get_my_pcb
     (nonpaged_pcb_pointer);
if not odd (return_code) then
    sys$exit (return_code);

new (local_pcb_pointer);
pcb_size := size (pcb);
local_pcb_pointer^.pcb$w_size := pcb_size;
writeln('local_pcb size = ',pcb_size);
writeln;
return_code := copy_system_control_block
          (nonpaged_pcb_pointer :: integer,
          local_pcb_pointer :: integer);
if not odd (return_code) then
    sys$exit (return_code);

pcb_size := local_pcb_pointer^.pcb$w_size;
writeln('pcb size = ',pcb_size);
writeln('pcb at ',
        hex(nonpaged_pcb_pointer));
writeln;
print_pcb;
writeln;
nonpaged_jib_pointer :=
      local_pcb_pointer^.pcb$l_jib;
new (local_jib_pointer);
jib_size := size (jib);
local_jib_pointer^.jib$w_size := jib_size;
writeln('local_jib size = ',jib_size);
writeln;
return_code := copy_system_control_block
     (nonpaged_jib_pointer :: integer,
     local_jib_pointer :: integer);
if not odd (return_code) then
    sys$exit (return_code);

jib_size := local_jib_pointer^.jib$w_size;
writeln('jib size = ',jib_size);
writeln('jib at ',
        hex(nonpaged_jib_pointer));
writeln;
print_jib;
writeln;

nonpaged_phd_pointer :=
        local_pcb_pointer^.pcb$l_phd;
new (local_phd_pointer);
phd_size := size (phd);
writeln('local_phd size = ',phd_size);
writeln;
return_code := copy_nonpaged_buff
    (phd_size,phd_size,
    nonpaged_phd_pointer :: integer,
    local_phd_pointer :: integer);
if not odd (return_code) then
    sys$exit (return_code);

writeln('phd at ',
        hex(nonpaged_phd_pointer));
writeln;
print_phd;
writeln;

end .
```

```
        .title        exampsubs - test system service dispatcher

;
;        macro definitions
;
;        define_service - a macro to make the appropriate entries in several
;                        different psects required to define an exec or kernel
;                        mode service.  these include the transfer vector,
;                        the case table for dispatching, and a table containing
;                        the number of required arguments.
;
;        define_service name,number_of_arguments,mode
;
        .macro       define_service,name,narg=0,mode=kernel
        .psect       $$$transfer_vector,page,nowrt,exe,pic
        .align       quad                 ; align entry points for speed and style
        .transfer    name                 ; define name as universal symbol for entry
        .mask        name                 ; use entry mask defined in main routine
        chm%extract(0,1,mode) #<%extract(0,1,mode)code_base+'mode'_counter>
                                          ; change to mode and execute
        ret                               ; return
        'mode'_counter='mode'_counter+1   ; advance counter

        .psect       'mode'_narg,byte,nowrt,exe,pic
        .byte        narg                 ; define number of required arguments

        .psect       sssacp_'mode'_disp1,byte,nowrt,exe,pic
        .word        2+name-%extract(0,1,mode)case_base       ; make entry in mode case table

        .endm        define_service

;        $ent - a macro to define an entry point
;
;        $ent entry_point regs
;
        .macro        $ent ep r=<^m<r2,r3,r4,r5,r6,r7,r8,r9>>
        .entry        ep,r
        .endm         $ent
;
;        equated symbols
;

        $pcbdef                         ; define process control block offsets
        $plvdef                         ; define dispatcher vector symbols
;
;        initialize counters for change mode dispatching codes
;
kernel_counter=0                        ; kernel code counter
exec_counter=0                          ; exec code counter

        .psect        kernel_narg,byte,nowrt,exe,pic
kernel_narg:                            ; base of byte table containing the
                                        ;  number of required arguments.
```

```
        .psect          exec_narg,byte,nowrt,exe,pic
exec_narg:                              ; base of byte table containing the
                                        ;  number of required arguments.

        define_service          get_my_pcb,1,kernel

        define_service          copy_nonpaged_buff,4,kernel

        define_service          copy_system_control_block,2,kernel



;
; the base values used to generate the dispatching codes should be negative for
; user services and must be chosen to avoid overlap with any other privileged
; shareable images that will be used concurrently.  their definition is
; deferred to this point in the assembly to cause their use in the preceding
; macro calls to be forward references that guarantee the size of the change
; mode instructions to be four bytes.  this satisfies an assumption that is
; made by for services that have to wait and be retried.  the pc for retrying
; the change mode instruction that invokes the service is assumed to be 4 bytes
; less than that saved in the change mode exception frame.  of course, the particular
; service routine determines whether this is possible.
;
kcode_base=-20202                       ; base chmk code value for these services
ecode_base=-20202                       ; base chme code value for these services
        .page
        .sbttl        change mode dispatcher vector block
;++
; this vector is used by the image activator to connect the privileged shareable
; image to the vms change mode dispatcher.  the offsets in the vector are self-
; relative to enable the construction of position independent images.  the system
; version number will be used by the image activator to verify that this shareable
; image was linked with the symbol table for the current system.
;
;                       change mode vector format
;
;           +-------------------------------------------+
;           !             vector type code              !   plv$l_type
;           !             (plv$c_typ_cmod)              !
;           +-------------------------------------------+
;           !           system version number           !   plv$l_version
;           !             (sys$k_version)               !
;           +-------------------------------------------+
;           !       kernel mode dispatcher offset       !   plv$l_kernel
;           !                                           !
;           +-------------------------------------------+
;           !          exec mode entry offset           !   plv$l_exec
;           !                                           !
;           +-------------------------------------------+
;           !                 reserved                  !
;           !                                           !
;           +-------------------------------------------+
```

```
;           !                 reserved                  !
;           !                                           !
;           +-------------------------------------------+
;           !          rms dispatcher offset            !   plv$l_rms
;           !                                           !
;           +-------------------------------------------+
;           !              address check                !   plv$l_check
;           !                                           !
;           +-------------------------------------------+
;
;
        .psect        sssacp_services,page,vec,pic,nowrt,exe

        .long       plv$c_typ_cmod          ; set type of vector to change mode dispatcher
        .long       sys$k_version           ; identify system version
        .long       kernel_dispatch-.       ; offset to kernel mode dispatcher
        .long       exec_dispatch-.         ; offset to executive mode dispatcher
        .long       0                       ; reserved.
        .long       0                       ; reserved.
        .long       0                       ; no rms dispatcher
        .long       0                       ; address check - pic image
        .page
        .sbttl       kernel mode dispatcher
;++
; input parameters:
;
;        (sp) - return address if bad change mode value
;
;         r0  - change mode argument value.
;
;         r4  - current pcb address. (therefore r4 must be specified in all
;                 register save masks for kernel routines.)
;
;         ap  - argument pointer existing when the change
;               mode instruction was executed.
;
;         fp  - address of minimal call frame to exit
;               the change mode dispatcher and return to
;               the original mode.
;--
        .psect  sssacp_kernel_disp0,byte,nowrt,exe,pic
kaccvio:                            ; kernel access violation
        movl    #ss$_accvio,r0      ; set access violation status code
        ret                         ;  and return
kinsfarg:                           ; kernel insufficient arguments.
        movl    #ss$_insfarg,r0     ; set status code and
        ret                         ;  return
knotme: rsb                         ; rsb to forward request

kernel_dispatch::                    ; entry to dispatcher
        movab   w^-kcode_base(r0),r1 ; normalize dispatch code value
        blss    knotme               ; branch if code value too low
        cmpw    r1,#kernel_counter   ; check high limit
```

```
        bgequ   knotme                  ; branch if out of range
;
; the dispatch code has now been verified as being handled by this dispatcher,
; now the argument list will be probed and the required number of arguments
; verified.
;
        movzbl  w^kernel_narg[r1],r1    ; get required argument count
        moval   @#4[r1],r1              ; compute byte count including arg count
        ifnord  r1,(ap),kaccvio         ; branch if arglist not readable
        cmpb    (ap),w^<kernel_narg-kcode_base>[r0] ; check for required number
        blssu   kinsfarg                ;  of arguments
        casew   r0,-                    ; case on change mode
                 -                      ; argument value
                #kcode_base,-           ; base value
                #<kernel_counter-1>     ; limit value (number of entries)
kcase_base:                             ; case table base address for define_service
;
;        case table entries are made in the psect sssacp_kernel_disp1 by
;        invocations of the define_service macro.  the three psects,
;        sssacp_kernel_disp0,1,2 will be abutted in lexical order at link-time.
;
        .psect  sssacp_kernel_disp2,byte,nowrt,exe,pic
        rsb                             ; return to reject out of
                                        ; range value
        .page
        .sbttl  executive mode dispatcher
;++
; input parameters:
;
;        (sp) - return address if bad change mode value
;
;         r0  - change mode argument value.
;
;         ap  - argument pointer existing when the change
;               mode instruction was executed.
;
;         fp  - address of minimal call frame to exit
;               the change mode dispatcher and return to
;               the original mode.
;--
        .psect        sssacp_exec_disp0,byte,nowrt,exe,pic

exec_dispatch::                         ; entry to dispatcher
;
;     Note: since there are no executive routines defined in this sample
;           program, the code for the executive dispatcher has been eliminated.
;           If a executive dispatcher were needed, the logic would be the same
;           as for the kernel dispatcher above.
;
enotme:         rsb                     ; rsb to forward request
ecase_base:                             ; case table base address for define_service
;
;        case table entries are made in the psect sssacp_exec_disp1 by
```

```
;        invocations of the define_service macro.  the three psects,
;        sssacp_exec_disp0,1,2 will be abutted in lexical order at link-time.
;
        .psect        sssacp_exec_disp2,byte,nowrt,exe,pic
        rsb                           ; return to reject out of
                                      ; range value
        .page
        .sbttl get_my_pcb - get address of current pcb


;
; equated symbols:
;
pcbaddr        = 4                                  ; address of pcb

        $ent     get_my_pcb                 ; save registers
        ifnowrt #4,@pcbaddr(ap),55$         ; br if can't write
        movl     r4,@pcbaddr(ap)            ; get own pcb address
                                            ;        and return as value
;
; Note: the PCB address is provided by the kernel dispatcher preloaded in r4;
;       if needed, the current PCB address can also be found in @#sch$gl_curpcb
;
        movl     #ss$_normal,r0             ; set normal completion status
        ret
55$:    brw      kaccvio                    ; indicate access violation

        .page
        .sbttl  routine to copy buffer to or from nonpaged

;
;
; calling sequence: rc := copy_nonpaged_buff(%immed fromlen,tolen,from_ptr,to_ptr)
;


        $ent     copy_nonpaged_buff
        movl     12(ap),r2                  ; get address of from buffer
        movl     16(ap),r3                  ; get address of to buffer
        bbs      #31,r2,47$                 ; br if "to" buffer is in nonpaged pool
        ifnowrt 8(ap),(r3),55$              ; check "to" buffer for write access
47$:
        bbs      #31,r2,57$                 ; br if from buffer is in nonpaged pool
        ifnord   4(ap),(r2),55$             ; check "from" buffer for read access
57$:
        movc5    4(ap),(r2),#0,8(ap),(r3) ; do the move
        movl     #ss$_normal,r0             ; set normal completion status
        ret
55$:    brw      kaccvio                    ; indicate access violation

        .page
        .sbttl  routine to system control block with internal length
```

```
;
;
; calling sequence: rc := copy_system_control_block(%immed from_ptr,to_ptr)
;


        $ent     copy_system_control_block
        movl     4(ap),r2                 ; get address of from buffer
        movl     8(ap),r3                 ; get address of to buffer
        bbs      #31,r2,47$               ; br if "to" buffer is in nonpaged pool
        ifnord   #pcb$w_size+2,(r3),55$   ; make sure we can read length of "to" buf
        ifnowrt  pcb$w_size(r3),(r3),55$  ; check "to" buffer for write access
47$:
        bbs      #31,r2,57$               ; br if from buffer is in nonpaged pool
        ifnord   #pcb$w_size+2,(r2),55$   ; make sure we can read length of "from" buf
        ifnord   pcb$w_size(r2),(r2),55$  ; check "from" buffer for read access
57$:
        movc5    pcb$w_size(r2),(r2),#0,pcb$w_size(r3),(r3); do the move
        movl     #ss$_normal,r0           ; set normal completion status
        ret
55$:    brw      kaccvio                  ; indicate access violation

        .end
```

# CHOOSING A DOCUMENT-FORMATTING SYSTEM

Richard K. Wallace
Los Alamos National Laboratory
Los Alamos, NM

ABSTRACT

After surveying available tools for formatting large computer code manuals, we chose the TeX system, to be initially implemented on VAX 11/780 and 8600 computers. We also recognized that a "What You See Is What You Get" word processor offers sufficient capabilities for small (5 - 10 page) reports and manuals, and recommended that WordMARC be considered for formatting in those situations.

## BACKGROUND

Los Alamos National Laboratory is a federally funded applied research laboratory managed by the University of California for the U.S. Department of Energy under contract W-7405-ENG- 36. The Laboratory engages primarily in energy, national defense, and accelerator/nuclear physics research. It employs about 7800 people and is divided organizationally into 43 Divisions. This paper discusses criteria used by the Applied Theoretical Physics Division (X Division) to select a document formatting system. X Division consists of about 260 employees, more than 200 of whom have doctorates in physics-related disciplines and all of whom have extensive interactive computing experience.

The major Laboratory computing center, managed by C Division, is the Central Computing Facility, which contains 7 Cray supercomputers, 8 large CDC computers, and 10 DEC VAXs, with a total computing capacity equivalent to 20 Cray-1 supercomputers. In addition, nearly 100 Distributed Processors, all VAX 11/780, 785, or 8600s, are scattered over 43 square miles, linked by DECNet and managed by the individual divisions. Owing to the defense work, the computing resources are divided into classification partitions, each completely separate (no communication channels) from all other partitions.

## PURPOSE

In August 1984, we formed a Committee to recommend a replacement for the then-current computerized documentation tools(TRIX/RED, REDPP), which would be unavailable after removal of the Laboratory's secure CDC 7600. Recent turnover in the code user groups emphasized the lack of current, comprehensive documentation (user and physics manuals) for the major X-Division production codes. This lack of documentation increases the training time required for new users and code developers and hinders efficient code use by them and by experienced users. The existing code manuals must be continually revised and expanded as the codes rapidly evolve.

We have therefore surveyed the field of document production in search of a modern, efficient, long-term document-formatting system that will satisfy our need for producing thorough, clear, current documentation as simply as possible. The system development was coordinated with C Division to reduce duplication of effort and prevent future compatability problems.

## SUMMARY

We recommended that TeX be used for formatting X-Division code manuals. Although the Division should not require the use of TeX, that tool should be seriously considered for any major documentation effort. We recognize that WordMARC may offer sufficient formatting capabilities for small (5-10 page) reports and manuals and should be considered for those applicat ons.

To obtain the full benefit of the TeX documentation system, the following hardware was recommended:

- A high speed (at least 24 pages/min) laser printer.
- An upgrade for one of our two VAX 11/780s to a DEC 8600 to provide greater responsiveness, larger CPU capacity, and improved availability of full screen text editors. Even if TeX became available on CTSS (the Cray operating system), the local VAXs could be heavily used for text entry and WordMARC applications.
- A low-cost (under $3000) laser printer that can produce local (in office) output; possible candidates include the DEC LN03 and the HP LaserJet.
- Workstations with a preview capability for frequent TeX users.

C Division was strongly encouraged to provide the following software support:

- A CTSS (Cray) implementation of TeX; this is in progress.

- Simple lineprinter/ASCII output from standard TeX DVI files; rudimentary package is now in use.
- Central Computing Facility output capable of producing 5000 formatted pages/day.
- A method to merge TeX text with graphics files that are in the unique Los Alamos Common Graphics System metafile format.
- Conversion programs for TROFF, TRIX/RED, and VMS WordMARC.
- Classified consulting services on TeX.
- "Writer's Workbench"-type software (such as a spelling checker) for TeX files.

Justification - Requirements

The selection of TeX for the X-Division formatting system was based on its satisfaction of the following unique X-Division requirements. The system should

1. be easily portable to new operating systems, minimizing future translations such as must now be done for the large number of LTSS (CDC 7600 operating system) TRIX/RED files. The system should also be widely used outside of DOE to increase the support for and knowledge about it,
2. be declarative (using predefined structures for headers/footers, sections, paragraph indentations, examples, etc.) rather than procedural (requiring the author to define page layout during text-, or content-entry). This requirement allows a few experienced people to maintain the detailed page layout macros, whereas casual users simply enter text,
3. easily accept mathematical equations and format them with as little user assistance as possible,
4. be capable of merging text with computer-generated graphics,
5. have automatic Table of Contents generation,
6. have automatic Index generation,
7. provide for nested tables,
8. have a source file format that facilitates macro construction to support detailed page layout macros, translation macros (from previous systems and into future systems), and text unformatting macros (to easily allow incorporation of arbitrary machine-readable text),
9. allow text input from any ASCII terminal (including Tektronix 4000 and 4100 series),
10. be accessible transparently from CTSS to eliminate user investment in learning a different operating system or accessing special hardware (most users work exclusively on the Cray CTSS systems rather than on VAXs),
11. produce simple ASCII text output for online help files from the same source file that produces fully formatted documents,
12. allow comments in the source file,
13. facilitate page layout changes or even allow determination of the layout after text entry,
14. symbolically reference equation, figure, section,and page numbers, and
15. allow "interactive" execution to provide error diagnostics and allow recovery from minor source file errors.

COMPARISONS

The major software for code documentation that begins to address the requirements listed above is the following:

Interleaf

Advantages:

1. Interactive "What you see is what you get" (WYSIWYG) system. This can be much easier and faster to use than a batch formatter for small files.
2. Instant feedback (screen shows all page layouts, fonts, text sizes, pagination, etc.).

Disadvantages:

1. No symbolic equation entry. Equations must be entered with a graphics package that draws each individual symbol or character on the page.
2. No symbolic referencing of equation numbers, sections, etc.
3. Operates only on SUN, APOLLO, and VAXStation II workstations.
4. Cost is $12,000 per workstation node, which is prohibitively expensive.

Interleaf was the most capable WYSIWYG formatting system on the market. It would unquestionably be the most productive system to have for a single user. However, the lack of symbolic mathematical entry and the unavailability for a timesharing system are fatal flaws for our purposes. The $12,000 per node price, coupled with the price of providing SUN-class workstations to everyone contributing text, is prohibitive. In addition, no SUN-class workstation has been approved for classified processing.

WordMARC, Version 5 ("Composer")

Advantages:

1. WYSIWYG system that is much easier and faster to use than a batch formatter for nonequation typing of small files.
2. Instant feedback of text and general page layout.
3. Preserves author's meaning (equations displayed on first typing).

Disadvantages:

1. Procedural; no declarative format.

2. Cannot easily change existing document format.

3. No comments allowed in source file.

4. VT100 emulation terminal required (for example, no Tektronix 4014).

5. Response slows to unacceptable times with large documents and many simultaneous users. Response time is more critical for completely interactive systems. The continuous formatting increases the CPU load compared with that of a batch formatter.

6. Less involvement allowed to professional editors/designers.

7. Limited (and in some cases insufficient) mathematical capabilities.

8. No proportionally spaced laser printer output.

The disadvantages indicate that WordMARC may be ideal for formatting memos and short reports but would be inadequate for very large manuals. Although WordMARC (from Marc Software) was specifically compared here, the disadvantages are similar for other WYSIWYG systems, such as MASS-11. They all generally require VT-100 emulation capability, are generally procedural (requiring some author involvement in page layout), are difficult to use for changing page layout retroactively, and require interactive computer response time. However, screen editors in such WYSIWYG systems could be used to prepare the ascii input files for a batch editor, such as TeX or TROFF.

We found no WYSIWYG systems with all the capabilities listed under "Requirements" above. However, two batch formatting systems in common use (TROFF and TeX) could satisfy nearly all of our requirements, and their respective advantages are listed below. C Division has decided to support both TROFF and TeX as Laboratory document production systems.

TROFF with EQN and TBL

May be easier to learn than standard TeX (but not significantly easier than LaTeX).

Better table generation capability than standard TeX.

Writer's Workbench editorial software available.

TeX

1. Arbitrary length command names (TROFF restricts commands to less than 2 characters).

2. More portable than TROFF (TeX is available in generic Pascal and C, whereas TROFF is tied intimately to the UNIX operating system).

3. Los Alamos Common Graphic System TeX interface exists for QMS laser printers, so merging text and graphics is a reality.

4. Slightly more control over output appearance.

5. More widely available screen preview systems (including SUN, APOLLO, IBM AT, Apple Macintosh, and Tektronix 4014).

6. TROFF requires the UNIX operating system, which is currently unacceptable for classified computing.

Points 2 and 6 above are sufficiently serious that we consider TROFF an unacceptable solution. TeX is therefore the most appropriate choice for an X-Division formatter.

## CONCLUSIONS

We chose TeX as our standard document formatting system, largely because of its great portability compared to TROFF. For small memos and reports, many secretaries still use WordMARC. Since we reached our decision, several other divisions at the Laboratory have begun using TeX, and the the official publication division (which uses an APS-5 phototypesetter for high-quality output) is committed to switching completely to TeX. The Laboratory is moving to standardize on Postscript (from Adobe Systems) as a common text/graphics device independent file structure, and we are now obtaining hardware and software to allow TeX output through Postscript devices. In addition, the Laboratory has just moved to support LaTeX (a TeX macro package) as the standard version of TeX. We currently use LaTeX on SUN, APOLLO, VAXStation II workstations, IBM XT, AT, Apple Macintosh, VAX/VMS, and VAX/UNIX, and have contracted for an implementation on CTSS.

## FURTHER INFORMATION

- TeX: TeX Users Group, P.O. Box 594, Providence, RI 02901.

- LaTeX: TeX macro package developed by Leslie Lamport (now at DEC). For information, contact the reference under "TeX".

- TeX on workstations, and output to Postscript devices: Textset Inc., 4116 4th. St., P.O. Box 7993, Ann Arbor, MI 48107. (313) 996-3566.

- TeX on IBM XT/AT: PC TeX Inc., 20 Sunnyside, Suite H, Mill Valley, CA 94941, (415) 388-8853, or Micro\TeX, Addison-Wesley Publishing Co., Educational Media Systems Division, Reading, MA 01867. (617) 944-3700, ext. 2677.

- WordMARC: Marc Software International, 260 Sheridan Ave, Suite 200, Palo Alto, CA 94306. (415) 326-1971.

- Interleaf: Interleaf Inc., 1100 Massachusetts Ave., Cambridge, MA 02138. (617) 497-5570.

- MASS-11: Microsystems Engineering Corp., 2040 Hassal Road, Hoffman Estates, IL 60195.

- TROFF: UNIX System manual, Bell Laboratories or Computer Science Division, University of California, Berkeley, CA 94720.

# Use of the DEC Test Manager in an ANSI Standard Maintenance Test Strategy

Jim Tibbetts
Hughes Aircraft Company - Ground Systems Group
Fullerton, California

ABSTRACT: The ANSI committee has published standards for test strategy development and documentation. Although primarily designed for use in new products, many of the precepts can be retro-fitted into existing software with the adoption of an aggressive testing policy. One of the necessary functions of such a testing policy is that of configuration control of the tests used; that is, a tracking system for version-to-version modifications and the specific tests used to validate their updated functionality. The DEC Test Manager provides a strong basis for this type of configuration control, as well as providing an ongoing baseline for product improvement. It is also adaptable to test tracking for products which require secondary analysis to validate their functions; for example, a tool which produces terminal-specific output and requires the substitution of datafiles into the Test Manager library. This paper describes some practical testing applications which have been developed using the DEC Test Manager.

## 1 INTRODUCTION

Both the Institute of Electrical and Electronic Engineers (IEEE) and the Department of Defense (DoD) have recently published standards for software testing [1-4]. Although these standards are primarily aimed at the new software development market, many of the precepts used can provide the basis for an aggressive maintenance testing program for existing software which has a significant portion of its expected life cycle still to come. In particular, the definitions of test design, procedure, and test case can serve as models for the development of a strong maintenance test strategy, and provide a clearly documented rationale for verifying both test coverage and test justification. Additionally, the methodologies described by both organizations are destined to become further integrated into current and upcoming software projects. The incorporation of these models into maintenance testing will insure a smooth and cohesive transition from development to continuous system use and upgrade.

## 2 HOW IS MAINTENANCE TESTING DIFFERENT?

Fundamentally, of course, testing is testing, whether designed towards individual program units, integration of large systems, or post-development maintenance. However, maintenance testing can often provide the test writer with some unique and sometimes perilous design situations. These come about in part due to the fact that maintenance testing is primarily a specialized type of regression testing; that is, the tests should be designed to remain essentially the same throughout the testable life of the software system. This is consistent with the intuitive idea that the correct functionality of a particular software system should also remain unchanged throughout its life cycle. Errors, naturally, are to be corrected as they are identified, and new tests should be added to the test base to verify these corrections, rather than continually modifying the original tests, as is more frequently the case during development testing. Extensive regression testing usually results in large test bases.

One of the most frequent problems which faces maintenance testing is the case where no previous tests exist at all. This state of void can be caused by one or more of a number of factors. A few of the more common excuses given are:

1. personnel changes: for example, the five year veteran test lead was transferred and no one else really knew what he did;

2. company politics; as in the development team which NEVER talks to the maintenance team;

3. old age. This is perhaps the most common; the system simply is so old that no-one has any idea where the previous tests would be located, if they existed at all.

The usual result of the 'no previous test' situation is that each user has developed a personal test suite, which verifies enough of the system capability to satisfy his own needs, but probably does not test the system completely or effectively. Often a good starting point for developing a maintenance test strategy is to collect as many of these personal test suites as possible, evaluate them, and blend them together to come up with a 'first cut' test base. However, it must always be remembered that this approach will probably result in an incomplete set of tests, and that more test design work will be necessary before a high level of confidence can be attained.

Almost as many problems can arise when a previous test base DOES exist. With the normal pressures of managerial deadlines and cost cutting, it can be a serious temptation to use something which already exists without any further investigation. This is especially true with a group of somewhat cryptic tests, perhaps left over from the original development team guru, if they are purported to thoroughly exercise a particular software system. They are often passed on, like a mystical religious secret, and heaven forbid that anyone should ever question the accuracy or validity of those tests! Aggressive maintenance testing is, therefore, by definition, a potentially heretical position, and very possibly will not make many friends. Nonetheless, test sets must be carefully updated and expanded to keep pace with the software system changes.

Somewhat related to this latter case is the situation where a severely patched system, probably near the end of its useful life, is turned over for maintenance testing. Here the testing problems are further compounded by the lack of anyone with significant knowledge about the system, including its original intent or condition. Any documentation that can be located is probably so far obsolete as to be essentially useless. There are no general rules to be followed in cases such as this, but a good starting point may be to re-evaluate the need to start such an aggressive testing policy so late in the useful life cycle of the particular system.

It may be a better use of resources to concentrate on newer, less obsolete systems and let whatever testing strategies exist continue until the older software can be phased out of circulation entirely. Experience and company policy will have to dictate the path to follow.

A final general point about maintenance testing which should be made is of a somewhat philosophical nature. By the time a software system has reached the point of maintenance in its life cycle, it will generally be assumed that it 'works' and that any problems to be located will either be virtually insignificant in nature, or due to such an extrordinary set of circumstances that no normal user would ever be apt to encounter them. While this is certainly the goal of good software engineering, bitter experience has proven that such an ideal state is usually not attained for several maintenance iterations, if, indeed, it is ever reached. However, the myth continues, and it often influences the test strategies which are developed for maintenance testing. The subconscious pressure to prove that a system works must be constantly tempered with a conscious effort to test objectively and completely.

## 3 A STRATEGY FOR AGGRESSIVE TESTING

In order to effectively use any testing tool, there are a few basic concepts which need to be considered. These points will not be belabored, as they could easily become the basis for an entire presentation on software testing, but a quick review of some of the more important ones which relate directly to the use of the Test Manager is appropriate.

The first of these is that the test writer must have a thorough understanding of the system being tested: its capabilities and limitations, as well as the impact of any recent changes made during the maintenance phase. For example, it would be an obvious misdirection of resources to test a mathematical function, defined as being accurate to four decimal places, with a

series of inputs which would generate results differing only at six decimal places. Unfortunately, much of the time the test strategies are not as obvious as in this example.

Once the system is thoroughly understood, a hard line must be drawn at that understanding, and each test case must be compared back to this line. Does the software perform as expected? Perhaps even as documented? It is here that many test design errors are made, especially if the test writer is the same individual who has been responsible for program maintenance. It is far too easy to overlook a simple item, perhaps a required input order, that is not specifically documented anywhere. An effective test would involve a change in the order, but, since the test writer may KNOW that the routine will work if the order is reversed, no test is generated to specifically check this case. The result may be a tested piece of software that has a 'time bomb' bug in it. After all, it is a well known fact that if something has been foolproofed, some fool will come along and find a way to break it.

The third concept is to have a test strategy which allows for expansion. A software system will seldom remain static during its useful lifetime. Error correction and enhancement create a constantly changing mode of operation, requiring a constantly changing set of tests to verify the correctness of that operation. Concurrent with this requirement is the need to verify that correct functionality is not lost during the maintenance cycle. This implies that large portions of the test base should remain intact, at least until it has been made obsolete by software changes. Of course, before any changes or additions can be made to the test base, it is necessary to know what was specifically tested previously. A well thought out strategy for testing can simplify the process of knowing what is and is not being tested, and thus can simplify the expansion process.

The final concept is one which is almost universal in any software engineering field. To develop an effective test strategy will require creativity, experience, and a good basic methodology. Creativity will provide the insight to generate tests that are at once specific and unique to the system being tested. Experience will provide the background to know the limitations of the test system, and how they will impact the test effort. The methodology will provide the mechanism for designing the tests and for tracking and reporting the results of the test effort. It is here that the DEC Test Manager shows its greatest strengths.

Already the basic requirements for an effective and aggressive test strategy have been hinted at. A good test writer must know what is appropriate to measure, and must know how to measure it, which is where experience becomes an invaluable partner. The Test Manager allows very specific test cases to be written, each of which is responsible for testing a single, unique characteristic of the software system. Thus the test criteria described by the test documentation can be translated almost directly into a functioning test base.

Additionally, the test writer must plan ahead for further testing improvements. This forward thinking can be as simple as test case organization and indexing, or it can be as complex as embedded cross referencing of test results, requiring that tests be performed in a certain order, or at certain times. Use of the Test Manager will ensure that current testing requirements of this type are carefully documented and controlled, so that future test expansion will not inadvertently destroy large portions of the test design.

What other benefits can be expected from all of this test planning and writing effort?

Careful maintenance test management will assist in catching 'old' glitches which have been undone by current correction or enhancement work. Similarly, it will help identify any 'new' glitches that may have been introduced by the current software revision. As automated test management becomes more and more integrated into the testing strategy, rapid debugging will aid in producing high-reliability software with a minimum of maintenance turnaround time.

Review of the test strategy and test cases, especially by those not directly involved in the maintenance programming effort itself, will help to identify any testing insufficiencies. By utilizing a well-structured test management strategy, the insight gained through periodic review can often lead to the saving of many dollars and staff hours of work by ensuring that all current functionality has been verified, and that many error trapping pathways have also been exercised.

Often the major schism between end-user satisfaction and expectation is a misunderstanding of the accompanying documentation. If the test writer insists on designing tests based on that same documentation, rather than on assumptions made about the system, shortcomings in the clarity of the documentation will become obvious. With the Test Manager, notes regarding sources of test rationale can be included as comments directly in the test case, ensuring that references to documentation will not be lost.

One last benefit from having a well structured and aggressive test strategy should be mentioned, although it may well be obvious. The test design and test results may themselves suggest future software development or test improvements. Since these suggestions for improvement were developed from the testing process itself, specific test cases and expected results have probably already been prepared, and can be implemented quickly into the test base.

## 4 SOME TRICKS TO USE WITH DTM

As with any tool, investigating its limits can often provide valuable insights for use in future situations. Four of the more valuable techniques which have been used are described below, as well as an indication of what conditions predicated their discovery. The techniques are not restricted only to the Test Manager; in fact, several of them are based solely on characteristics of the VAX/VMS system. Nonetheless, they have all helped to solve some interesting problems of maintenance testing.

### 4.1 DUMP Vs TYPE

Consider the case of a system which produces a binary object file as its major output. Many compilers fall into this category. Although the Test Manager is quite capable of directly comparing object files, since it uses the VMS 'DIFFERENCE' operator, the resulting information will be, at best, difficult to interpret. Use of a VMS 'TYPE' command to look at the file will result in a rapid display of garbage, possibly resulting in terminal lockup, with all of the ringing bells and flashing lights that normally occur when attempting to look at a binary file. If this command were embedded in a batch procedure, the entire process would hang until stopped by the user, the system operator, or some type of CPU time policing routine (i.e., after 2 hours of CPU time, the process is killed regardless of completion status).

One of the methods which can be utilized to verify the output of a routine such as this is to use the VMS 'DUMP' command rather than 'TYPE'. This command produces a nicely formatted display at the terminal, with the object code translated into ASCII, decimal, hexadecimal, or octal, according to individual user preference. Although the characters and numeric sequences may mean very little to a human user, the resulting display can be saved exactly as sent to the terminal by redirecting the output to some arbitrary file, using either the VMS 'DEFINE' or 'ASSIGN' operator for SYS$OUTPUT. In effect, the object file is translated from binary to an intermediate text form, relying on the operating system's capability for interpretations of this type. When two files have been saved in this way, they can be used for comparison by the Test Manager just as with other file types, and the differences between them, if any, will be found and indicated by the 'UNSUCCESSFUL' comparison status. Since 'DUMP', 'DEFINE', and 'ASSIGN' are ordinary VMS commands, they can also be invoked from inside a command file running in a batch mode. If desired, the batch log files themselves can be used

as the basis for comparison by the Test Manager, although, due to the additional output included in a log file, this will generally result in comparisons which are less clear than those obtained by specifically redirecting the output for the DUMPed files.

This method is very effective for locating differences in an object file, although actual use has identified one minor inconvenience. The formatted display from the DUMP command includes some additional lines, listing the file identification, current date, block number being read, and similar information. This will be different for each file that is DUMPed; therefore, if two files which were saved as described above are compared, the Test Manager will always indicate that they are different at these locations. Fortunately, a quick visual scan of the resultant Test Manager difference file will usually show if any non-identification differences were found, since the display format is easily recognizable. As the Test Manager does require an interactive review of the test results, this additional scanning step does not usually represent a significant increase in result analysis time. The difference files must be reviewed anyway, and the change in display format makes any relevant differences that much more obvious. If desired, of course, an editor could be invoked to remove or mask these lines prior to the comparison phase.

## 4.2 Artificial Ingredients

One technique which is widely used during development testing is known as 'stubbing'. This technique basically allows for testing of an isolated code unit without requiring all of its normal driving environment. In fact, this is often the only way to test some units during development. Maintenance testing can use an analogous technique, but the stubs must feed the external environment, rather than bypassing it.

The Test Manager allows for this type of testing. Since it uses a VMS command file for control, any commands which could be entered from the terminal keyboard can also be embedded into the command file to be used as input during batch processing. This ensures that the desired input values remain constant during test repetition, and that comparisons between the current test results and the previously benchmarked results are valid. Additionally, the control file can be liberally commented to document the (usually) cryptic inputs being used. However, no comments should appear on the actual input lines themselves, because the comments will also be interpreted as input data, with the obvious unintentional and disconcerting results.

## 4.3 Extraction Routines

Occasionally, portions of an output file are going to be different no matter how they are generated, while other portions will remain essentially the same. An example of this could be a tool designed to generate a Job Control Language (JCL) input request for a remote host. The JCL that is generated is verified by submitting the job to the remote host, and by analyzing the returned output listing file. In this case, the returned file will contain large portions of machine specific output generated by the remote host, and having relevance only to that remote host. The output file is quite lengthy and, in contrast to the 'DUMP' file case mentioned above, a generated difference file can be almost as large as the original output file itself.

Further analysis of this case will help rectify what appears to be an insurmountable problem of data reduction. Recall that the software being tested was stated to be a tool which generates a JCL request. According to this statement, the actual domain of testing should be limited to the JCL which is generated. Hopefully this limitation was mentioned in the test plan or test design. Since the JCL is all that should be used for testing comparison, a way of clarifying the test results is to eliminate all but the actual JCL lines from the returned file before any comparison is done by the Test Manager.

File reduction of this type could be done using an editor, but this can become a seriously time and resource intensive activity. However, all of the JCL lines have a common characteristic: they all start with a special character in the first position of every line; in this case, a forward slash. It is a relatively trivial exercise to write a routine which will scan through a file, locate all lines with a slash in the first character position, and copy them to a second file. This reduced file can then be used as the result file to be compared from run to run by the Test Manager. Only the differences in the generated JCL will be identified, rather than all of the differences in the original output file. Although not all data extraction routines will be as simple as this one, in general, it should be possible to identify some common characteristic of the desired portions of a file. In this way, machine specific data dumping will not interfere with a clear understanding of the result comparison criteria.

One final reminder: A careful review of the test design may help suggest which portions of an unwieldy output file should be used, and which can be disregarded. If the test documents do not specify these limits, it is possible that they need to be rewritten to clearly state what is and is not being tested, or what is in the actual domain of the test system's control. In the example given above, the only portion of the output file which could be VALIDLY analyzed was the JCL, since the software being tested was a tool designed to generate a JCL submission file. The rest of the output was generated by a remote system, over which the tool itself had no control, and, therefore, could take no responsibility. As mentioned before, it is important to know what the appropriate testing limits are for the system in question!

### 4.4 Changing The Image

One other technique that can be used with the Test Manager relies on the fact that the Test Manager really doesn't know, nor does it care, what results it is comparing. It must assume that the test writer has set up the tests in the correct manner, and that anything it has stored as results will then be used in the comparison phase. If a 'dummy' result file is overwritten before comparison by another file which was generated during the testing, the Test Manager will make its analysis based on the new file. This provides the test writer with a large amount of flexibility, especially with respect to the use of prologue and epilogue files.

Consider a software routine which generates, as its target output, some code that must be executed to be verified, but whose only direct output is a log file describing the generation process. An example of this type of software might be a revision of an operating system. The execution is a separate step from the code generation; in fact, it is at least one step removed, because the system must be generated and loaded, and a series of system validation tests must then be run. One way of utilizing the Test Manager for maintenance testing of upgrades of this type is to completely substitute the result file that is generated with a file which has more analytical validity.

The Test Manager would receive, as its initial result file, the log file which verifies that the code was produced. Although perhaps of some value initially, especially during test writing and debugging, this file gives no useful information regarding the success or failure of the software upgrade. During the epilogue portion of the test execution, however, a routine could be invoked which would generate the new system, load it into the test portion of memory, and execute the validation tests. These validation tests can be written in such a way that they produce a result file of their own. This result file can now be copied into the Test Manager's test execution subdirectory for this specific collection. This will effectively overwrite the previous results with a new file, and the comparison can be performed in the usual manner. The final product of all this prestidigitation is that the Test Manager is actually analyzing a totally different result file than it had

originally received, but which has a real function towards meeting the test requirements.

Three specific maintenance testing situations have resulted in some unusual but necessary data acrobatics. In each case, the versatility of the Test Manager has proven to be an integral part of the solution, and has provided for a strong, ongoing support of the products involved.

## 5 TRAPPING CRT OUTPUT

System configuration can present unexpected difficulties for maintenance testing. An example of this was the use of an overlayed operating system which was required for some in-house utilities.

### 5.1 The Problem

A group of software tools, specifically written to interface with a Remote Job Entry (RJE) package, required the use of a concurrent UNIX-type operating system. Although this presented no problems for a normal user, the system had one idiosyncracy which presented itself to the maintenance test team. Some of the output which was being sent to the terminal was bypassing both of the normal VMS routes SYS$OUTPUT and SYS$ERROR. The output did show up at the terminal, but it was not possible to redirect the output into a file for later analysis. This also meant that no output would be directed to a batch log file. The only method which was found to be successful in trapping the terminal output was to use a VMS 'SET HOST /LOG' command. Unfortunately, this command cannot be used from inside a batch procedure, thus making it difficult to design tests for use with the Test Manager.

### 5.2 The Solution

The first obstacles to overcome were how to trap the desired output in a file, and determining how this file could be used by the Test Manager. Since a 'SET HOST /LOG' command copies everything which is sent to the terminal, the log file would have a large amount of extraneous data in it, such as login notices, as well as being a single, large file. However, it would also include the necessary test outputs. The concept was that the Test Manager could utilize this log file to simulate the test results which it would otherwise expect from the terminal. This design required that the log file be generated prior to the initiation of the Test Manager batch process.

A collection was specified for the Test Manager, using dummy command files, and the batch process was submitted with a '/NOCOMPARE' qualifier. This produced a series of result files in the subdirectory identified by the global variable DTM$COLLECTION_NAME, which is defined by the Test Manager at batch submission time. No comparisons would be performed at this point, since the result files were to be overwritten during the next phase of the Test Manager processing.

The actual extraction routine was invoked from an epilogue file. This routine used a control file containing the same test execution lines which generated the 'SET HOST' output file. These lines serve as markers to separate the log file into individual test result files. A new file is opened when a marker line is found, and the log file is then copied into the new file until the next marker line is located, The process continues until all of the control file has been processed. In this way, the single file generated by the 'SET HOST' command would be broken up into result files corresponding to individual test cases, as expected by the Test Manager.

Once the individual test result files were generated, they were copied to the appropriate subdirectory as the final step of epilogue file processing. The batch process completed, and the test analysis was continued by invoking the Test Manager 'DTM COMPARE' command prior to the interactive 'DTM REVIEW' phase. Results were then generated in the same manner as during normal Test Manager processing.

## 6 CONTROLLING EXTERNAL DEVICES

Many software systems rely on the use of external devices, such as tape drives, to obtain input from or to return output to offline storage. Although it is possible to request the use of system resources such as tape drives from a batch process, unless the job is monitored, a process can wait essentially forever for the tape request to be acknowledged, or for the proper disk to be mounted. This problem involves the necessary interface with an additional individual, usually an operator, who probably has little knowledge about the test process which is going on. Nonetheless, a truly automated test design must account for this type of interface requirement.

### 6.1 The Problem

One set of tools assigned for maintenance testing assisted in tape-to-disk and disk-to-tape transfers between a VAX and a non-VAX remote system. Specifically, the tools read in an extremely foreign tape into a VMS file format, and wrote VMS files onto a very foreign tape format. It was not feasible to leave a tape drive allocated and mounted overnight, when the Test Manager batch job would normally be processed, but it was equally difficult to have the test team stay on into the second shift during the testing, and it was undesirable to have the tests run during prime time. Therefore, the problem became isolated at the tape/batch job interface.

### 6.2 The Solution

One helpful factor was that an operator would be available during the second shift time span, and the Test Manager batch job could be initiated during this time. Although the operator would have no time to logon and start a process, mounting and dismounting tapes would be acceptable. This reduced the problem to determining a way to have the Test Manager alert the operator that it required to have a tape mounted to begin its processing. The apparent place for such a control routine would be in a prologue file, processed before any testing began.

The first step was to allocate the tape drive. It was a distinct possibility that the desired tape drive would already be in use, due to system load and cluster configuration. For this reason, a status testing loop was inserted into the prologue. This loop would attempt to allocate the tape drive, and, if not successful, wait a designated period of time before trying again. Until the drive could be allocated, no further processing would be done, resulting in a minimal use of system resources.

Next, a mail message was sent to the test account, so that when the test team came in the following day, there was also a record that the tape drive was allocated for testing, and when. This time stamp helped to determine when the actual testing process started.

With all of the other duties that a second shift operator is assigned, it sometimes happens that they miss or forget about a tape request. To help avoid this problem, a special operator request was sent, which required that they reply to it. Experience has shown that this method provides inexpensive insurance that the tape 'MOUNT' request which follows is observed and performed expeditiously.

After the completion of the prologue file execution, the actual testing of the tools proceeds. The tests are arranged so that files are first written to the tape, and then read back from it. In this way, the test procedures do not require that a particular tape is mounted; any scratch tape will serve the purpose.

An epilogue file essentially reverses the procedure described in the prologue file. First, the tape is dismounted and deallocated. A mail message is sent to the test account providing a time of completion of the testing, and an extra message is sent to the operator requesting that the tape be returned to the scratch library. Note that no status loop is required in the epilogue

file; if another user wishes to use the same tape drive, it has been fully released by the Test Manager at this point, even if the operator doesn't reply to the message immediately.

## 7 THE COMPILER CHALLENGE

It is a pleasant occurrence to develop a process to handle a specific function, and then find out that the same process can be adapted to handle a second function with only minor modification. Such versatility became especially important during the testing of a compiler which was undergoing some extensive code revision.

### 7.1 The Problem

The source for this particular compiler was kept on the VAX under configuration control. However, the source was compiled against a DoD supplied image running on a remote IBM host, and the resulting executable was targetted for yet a third machine, in this case a proprietary embedded system. The verification tests to be run were designed to be compiled and executed on the embedded system. This was to be done using several small program units on the target, and directing the output to temporary files, which could be accessed later. Transfer between the VAX and the IBM was done via an RJE telecommunications link, and the executable image was transferred to the target system via tape. The temporary files could also be dumped to tape.

### 7.2 The Solution

Since the VAX was by far the most friendly of the three systems concerned, and since the source was impounded there anyway, it appeared to be an obvious choice to base the maintenance testing effort, or at least the majority of it, on that system. Additionally, management had requested that as much testing as possible be put under the control of the Test Manager.

The solution was achieved by looking at the sequence of events, starting with the end product. Several files were going to be generated on the target system. These files could be dumped to tape and transferred to the VAX, since the tools required to read the tape had already been written, and included the necessary file conversion parameters. They were actually written at the same time as the tools to write the executable tape from the VAX to the target, since what can be done can usually also be undone. Therefore, the major obstacle was solved: the result files could eventually be accessed by the Test Manager.

The manipulations required for the Test Manager to be able to use these files would need to encompass several functions. First, it would need to have a collection defined which specified dummy command procedures to create the psuedo-results for comparison. These result files were to be overwritten with the target result files when they became available. No prologue file would be needed, as all of the pre-Test Manager functions would take place on different machines. However, an epilogue command file would be specified, because it would be responsible for a majority of the file manipulation that would need to occur.

Now the compiler source code would be processed, and the executable image transferred to the target machine, where the tests would be run. Upon completion, the temporary result files from these tests would be dumped to tape.

The Test Manager collection would be submitted to its job queue, specifying a '/NOCOMPARE' in the submission line to ensure that the necessary file manipulation took place before processing and analysis was complete. As usual, the collection's subdirectory would be created and filled with the pseudo-result files from the dummy command procedures.

In the epilogue file, a request to mount a tape would be issued, similar to the request in the last example, but this time, requiring a specific tape by name. This was

the signal to the test operator to load the tape that had been produced on the target system. The tape would be read, the files converted as necessary, and then copied to the Test Manager subdirectory specified by DTM$COLLECTION_NAME. A dismount request completed the epilogue processing.

The Test Manager would now complete the batch job, and the operator can continue with the comparison and analysis phase when ready by issuing a 'DTM COMPARE'. The Test Manager will proceed just as though the entire testing process had been executed on the VAX, rather than on three different systems.

## 8 SUMMARY

Maintenance testing can be a constant challenge to the test team. Individual and unexpected problems can arise at almost every opportunity, requiring all of the resources that creativity and experience can offer. Full and efficient utilization of these resources requires an aggressive and well-structured test strategy, ideally modelled after the strategies described by the IEEE and the DoD for use during software development. Concurrently, effective use of testing strategies of this nature need an efficient and easy-to-use test methodology to ensure that an appropriate and functionally complete test base is developed. Application experience has demonstrated that the versatility of the DEC Test Manager can help provide the basis for reliable, high-quality software throughout its maintained life.

## 9 REFERENCES


1. IEEE Standard for Software Test Documentation, ANSI/IEEE STD 829-1983, IEEE Computer Society Press, 1983.

2. IEEE Standard for Software Quality Assurance Plans, ANSI/IEEE STD 730-1984, IEEE Computer Society Press, 1984.

3. Military Standard, Defense System Software Development, DOD-STD-2167, Department of Defense, Washington DC, 1985.

4. IEEE Standard for Software Unit Testing, Draft of 5-March, Subcommittee on Software Engineering Standards, IEEE, 1986.

# Guided Tour of an Emacs Extension: dired

Pete Kaiser
Digital Equipment Corporation
Marlboro, Massachusetts

Dired is an extension to the popular WYSIWYG editor Emacs. It provides a good way of examining and pruning directories of files, rapidly and easily.

## 1 What's Emacs?

This is not a tutorial on Emacs. However, in brief: Emacs is an interactive, extensible, customizable, full-screen character-cell editor invented by Richard M. Stallman. It exists in several different versions, the oldest of which is the one for PDP-10s, written largely by Stallman; and the latest of which is GNU Emacs, also written largely by Stallman. Versions exist for computers ranging in size from mainframes down to microcomputers; and under a large variety of operating systems, from the generic to the proprietary. These versions aren't identical to one another—they have different sets of primitives, they relate somewhat differently to their environments, and their extension languages differ—but they all have in common a philosophy ("power to the user; 7 bazillion keybindings isn't too many; extension and customizing are essential") and a high degree of malleability, so you can use them all from a single cognitive standpoint.

## 2 What's an Emacs extension?

An "extension" (or "package") is software written in the extension language—here a language called "Mlisp", for "Mock Lisp", because it's nearly, but not quite, Lisp—that uses Emacs's primitives to perform some function. The function need not be editing; indeed, it may not even use the screen (notwithstanding my calling it a "full-screen editor"; the version of Emacs I use most can run in batch mode, and one of my extensions uses it that way). So in writing an extension, or in thinking one up, you think in terms of writing programs that use the services Emacs can provide. These are services like the ones below; for the full list, see the documentation of your favorite version of Emacs.

- full-screen WYSIWYG (What You See Is What You Get) editing with intelligent screen management
- multiple windows
- multiple buffers
- the ability to run a subprocess with input from, and output to, a buffer
- callability from other programs
- changeable keybindings
- file wildcarding

## 3 What does dired accomplish?

Dired is an ergonomically good way of pruning directories of unwanted files; it helps solve the problem of how to examine quickly many files from a list—without having to TYPE each one—and get rid of the unwanted ones without having to DELETE each one individually.

Dired presents the user with a screen showing a directory listing, which may be any list of files you can find with a single VAX/VMS DIRECTORY command. With the cursor positioned at the left of a line containing the name of a file, the user can, with a single keystroke, bring a file onto the screen for viewing, mark it for deletion, or unmark it for deletion. With the cursor positioned at the left of a line containing only the name of a directory (e.g., the first line of the selection buffer) the user can use a single keystroke to mark or unmark all the files within that directory for deletion.

When the user is done viewing and marking, another single keystroke begins the actual cleanup. If any files are marked for deletion, the user gets a new screen showing only those files, and is asked for confirmation to delete them. At that point the user can (once again, with a single keystroke) get back to the directory screen; quit without deleting anything; delete all the marked files and quit; or delete all the marked files, displaying their names along the way, and quit. Directories that are emptied of files along the way, and that are marked for deletion (as files) are properly deleted—i.e., deletions are done from the bottom up, not from the top down.

There are different versions of dired. Why did I write another? Because the version included with the Emacs I use under VAX/VMS doesn't have the capabilities I want, in particular, the ability to work on more than one directory's worth of files at a time. For me that nearly crippled its usefulness. I wanted to be able to scan whole trees of directories at a time. I also wanted to be able to watch deletions being made, and the distributed version couldn't do that. So I wrote my own version.

# 4 Invoking an extension

Here we plunge into details. From now on, you can assume that I'm saying "in the version of Emacs I use on VAX/VMS" in relation to all details.

One invokes a package by invoking Emacs with the qualifier /PACKAGE=*extension*. In my login procedure I have a line

```
$ DIRED == "EMACS /PACKAGE=DIRED"
```

to make it easy to invoke DIRED with the line

```
$ DIRED
```

When Emacs sees /PACKAGE=DIRED it coerces DIRED into lower case and invokes an Emacs function named dired.mlp, which it expects to find in a library or somewhere along a search path defined in the logical name EMACS$PATH which you may define for yourself (there is a system default, however). Then it invokes a function named dired-com which is usually defined by dired.mlp, and dired-com then does the real work. Here's dired.mlp from my system:

```
(defun (dired-com
   (declare-global ~DIRED-com)
   (setq ~DIRED-com 1)
   (setq checkpoint-frequency 0)
   (setq silently-kill-processes 1)
   (execute-mlisp-file "dired")
   (dired
      (if (> (argc) 1) (argv 1) "")
      )
   ))
```

To translate: define the function dired-com. Declare a global variable ~DIRED-com to show that dired is active, and set it to show activity. Don't automatically checkpoint buffers. On exit from Emacs, kill subprocesses without asking for confirmation. Load a file named simply dired, which must be found in a library or along the search path. Invoke function dired, which that file defines, with an argument; if the user provided one, use that, and otherwise use the null string. Close all parentheses.

# 5 Keybindings

The ability to bind a single keystroke to a function that may do something very complex is at the heart of the Emacs philosophy. How is it done? First the function has to exist, and every Emacs worthy of the name has a long list of functions built in, and lots more available in libraries—like dired; then the function has to be known to Emacs when the key is bound to it. Furthermore, a keystroke can consist of several characters—the VT200 function key F20, for instance, transmits five characters at a single stroke. (How Emacs recognizes such a sequence of characters as a single keystroke is magic, and I'm not going to explain it here; just take it from me that it works.) Finally, a key can be bound to a function globally for all buffers, or locally within a single buffer. Dired does everything with local bindings, and here's how one of them looks:

```
(local-bind-to-key "~dired-help" '?')
```

To translate: "In the buffer we're in when this binding is made, execute the function ~dired-help when the user presses the '?' key". In all other buffers, the "?" key will have the usual effect: to insert a "?" in the text in the buffer.

# 6 Dired's subfunctions

This version of dired (there are others) uses some seventeen subfunctions:

```
~dired                ~dired-next
~dired-d              ~dired-previous
~dired-e              ~dired-q
~dired-e-help         ~dired-quit-delete
~dired-filename       ~dired-resume
~dired-format         ~dired-u
~dired-go-away        ~dired-un-e
~dired-help           ~dired-unlink
~dired-log-delete
```

## 6.1 ~dired

Initialize global values for dired, give the user a polite initialization message, and bind keys for the selection buffer. Run a subprocess with the VAX/VMS DIRECTORY command to get into a buffer the names of the files wanted.

Dired uses several buffers. The selection buffer, named "DIRED selection", is the one that holds the directory listing the user sees, and can be regarded as the principal buffer. Another, named "DIRED view", holds the file being viewed, when the user requests that service. "DIRED confirmation" is the buffer where file names are displayed for the user to confirm that those files to be deleted, and a hidden buffer "DIRED deletion", which the user never sees, holds those same filenames in another form for dired to manipulate further. And the "Help" buffer is used to display help text.

## 6.2 ~dired-d

In the selection buffer, mark a file for deletion. Remember that this only marks the file, it doesn't delete it yet.

## 6.3 ~dired-e

In the selection buffer, choose a file to examine, and get that file up on the screen.

## 6.4 ~dired-e-help

In the viewing buffer, display help text.

## 6.5 ~dired-filename

In the selection buffer, build the file's full name (absolute pathname) for use by another function. This function is internal, and isn't bound to a keystroke.

### 6.6 ~dired-format

An internal function, not bound to a keystroke. It's invoked in the buffer set up by ~dired with the results of the VAX/VMS DIRECTORY command, and reformats the buffer for dired's use.

### 6.7 ~dired-go-away

Done with dired; delete unwanted buffers and leave Emacs.

### 6.8 ~dired-help

Invoked in the selection buffer, display help text.

### 6.9 ~dired-log-delete

Invoked from the confirmation buffer, set up to show deletions as they're done, then make that happen by invoking ~dired-quit-delete.

### 6.10 ~dired-next

In the selection buffer, position the cursor at the beginning of the next line with a filename.

### 6.11 ~dired-previous

In the selection buffer, position the cursor at the beginning of the previous line with a filename.

### 6.12 ~dired-q

Invoked from the selection buffer, set up to confirm and perform deletions.

### 6.13 ~dired-quit-delete

In the confirmation buffer, the user has confirmed deletions. Delete the files and quit Emacs. If a flag has been set by ~dired-log-delete, display the deletions as they're done.

### 6.14 ~dired-resume

In the confirmation buffer, the user wants to return to the selection buffer. Do so.

### 6.15 ~dired-u

In the selection buffer, unmark a file for deletion. It's not an error to unmark a file that was never marked.

### 6.16 ~dired-un-e

In the examination buffer, return to the selection buffer.

### 6.17 ~dired-unlink

An internal function that deletes a single file, optionally displaying its name as it does so.

## 7 ~dired-e in detail

Here we examine in some detail how one of dired's subfunctions works.

```
(defun (~dired-e fn
   (beginning-of-line)
   (if (! (looking-at "[ D]")) (error-message ""))
   (setq ~dired-saved-dot (dot))
   (setq fn (~dired-filename))
   (switch-to-buffer "DIRED view")
   (local-bind-to-key "~dired-un-e" "\eOS")
   (local-bind-to-key "~dired-e-help" '\037')
   (if (error-occurred (read-file fn))
      (progn (switch-to-buffer "DIRED selection")
         (goto-character ~dired-saved-dot)
         (error-message (concat "Can't get " fn)))
      (message "(Use PF4 to return to dired)"))
   (setq mode-line-format
      (concat " DIRED examining file: " fn " ")))
  )
```

### 7.1 (defun (~dired-e fn

Define the function ~dired-e, with a single local variable fn.

### 7.2 (beginning-of-line)

Bulletproofing: put the cursor at the beginning of the line. (Just in case the user has put it elsewhere on the line, which is possible, although not usual.)

### 7.3 (if (! (looking-at "[ D]")) (error-message ""))

More bulletproofing: if the line doesn't begin with a space character or "D", it doesn't contain a filename the way dired formats them. Quit the function with a beep to the user.

Note that, although it's not usually expected to happen, a knowledgeable Emacs user can force things to happen that are beyond dired's ability to recover, and that's why there's bulletproofing. This has occasionally saved my own hide.

### 7.4 (setq ~dired-saved-dot (dot))

This looks like a legitimate line, at least a little bit. Set a global variable to the value of the current position of the cursor, so we can return here later.

### 7.5 (setq fn (~dired-filename))

Set the function's local variable to the full filename of this file.

### 7.6 (switch-to-buffer "DIRED view")

Switch to the examination buffer.

### 7.7 (local-bind-to-key "~dired-un-e" "\eOS")

Locally bind to the PF4 key the function to return to the selection buffer.

### 7.8 (local-bind-to-key "~dired-e-help" '\037')

Locally bind to the key with octal value 37 (on a VT200 keyboard, control-/), the display of help text for the examination buffer.

### 7.9 (if (error-occurred (read-file fn))

Try to read the chosen file into the buffer, and if that's not possible (e.g., if you haven't permission to read it) ...

### 7.10 (progn (switch-to-buffer "DIRED selection")

... return to the selection buffer and ...

### 7.11 (goto-character ~dired-saved-dot)

... position the cursor where we started from, ...

### 7.12 (error-message (concat "Can't get " fn)))

... giving an error message to let the user know why we're back here.

The progn above is needed because the `if` function expects each clause:

```
(if (test) (success) (failure))
```

to be a single function, and the progn allows us to wrap several actions inside a single envelope.

### 7.13 (message "(Use PF4 to return to dired)"))

Give the user a message, in the message area (the line below the mode line) saying how to get back to the selection buffer.

### 7.14 (setq mode-line-format

Set up the mode line ...

### 7.15 (concat " DIRED examining file: " fn " ")))

... to show where we are and display the full filename of the file being examined.

## 8 Enhancements

It's not hard to imagine enhancements to the package. One possible enhancement would be to make a file in the selection buffer invisible to `dired`; in other words, to remove it from the selection display and from all further consideration. This could be considered bulletproofing or uncluttering. It would work this way: with the cursor positioned on a line containing a filename, a single keystroke would cause that line to disappear. If that line were the only one under its directory name, the line containing the name of the directory would also disappear.

Another enhancement: with a single keystroke, mark a file for printing.

And finally: let a single keystroke mean "NOW!". With the cursor positioned on a line with a file marked for printing or deletion, the NOW key would cause it to be printed (deleted) immediately.

## 9 Fixing bugs

The package has some unwanted features. For instance, if `dired` is invoked from a VAX/VMS command line, I'd like to return to the CLI level when it's done; but when it's invoked from within Emacs, I'd like to return to the state Emacs was in—still within Emacs—when `dired` is done. So far I just haven't bothered to do this.

There are one or two others, but why embarrass myself?

## 10 Acknowledgments

Richard Stallman deserves tremendous credit for coming up with Emacs in the first place; Emacs's tremendous spread and popularity show that his idea was an idea of genius, and his GNU Emacs is a worthy successor in that tradition.

James Gosling wrote the first version of Emacs that used Mlisp and showed the world that a real Emacs could exist elsewhere than on DECsystem-10s and 20s. Until recently, this version of Emacs was undoubtedly the most widespread one; the version I use under VAX/VMS is based on it.

Bruce Dawson, Barry Scott, and Nick Emery are largely responsible for the current incarnation I've written about here. Barry and Nick continue to enhance it for use under VAX/VMS.

I am, of course, fully responsible for any software I've written, including the version of `dired` I've written about here.

# USING THE CMS CALLABLE INTERFACE

Glen Del Merritt
Computer Sciences Corporation
Moorestown, New Jersey

ABSTRACT

A discussion on using the DEC/CMS callable interface routines.

## INTRODUCTION

The DEC Code Management System (DEC/CMS, or just CMS) allows programmers to maintain source code in a controlled environment. Use of the CMS library allows programmers to track the development of their software and to retrieve previous generations of source for reference or further work. It also provides a "safe" place to store source files that are not the focus of current development.

DEC provides two ways to create, manipulate, and access files in a CMS library. The first, with which most users are familiar, is the DCL interface. Via DCL, you may enter commands at the VMS prompt, e.g.:

```
$ CMS FETCH MYFILE.CLD ""
```

or you may enter a special submode, in which all commands are assumed to be for CMS:

```
$ CMS
CMS> FETCH MYFILE.CLD ""
```

The former is the "original" interface and is still particularly useful in DCL command procedures. The latter is useful for purely interactive use. The leading "CMS" is not required, and execution of multiple commands is faster.

The second interface is the CMS callable interface. This interface allows programs written in any of the

VAX languages to call upon special entry points into CMS's sharable image. The DCL interface and programs written using the callable interface may peacably exist together, since access to the library is under complete control of CMS. The callable interface is the focus of this article.

## USES

Some of the potential uses of the callable interface are,

- o special source auditing tools to generate reports
- o maintaining additional tracking information in library transactions
- o security features that the DCL interface lacks, like
  - a "UAF" for a library to allow users different accesses to CMS elements
  - notifying a cognizant library manager of changes to the library AS THEY HAPPEN
- o generating an interface (and chucking DEC's) more suitable to your needs
- o providing input to compilers that are smart enough to get information from the library.

Report generation can be useful whenever you are asked to account for what you've done. Additional tracking information can make bug fixing easier by providing a cross reference to a list of previous fixes.

## DATA STRUCTURES

There are two special structures that you must use when accessing your CMS library with the callable interface. They are the library data block and the fetch data block.

The library data block is fifty longwords in size and is initialized by a call to CMS$SET_LIBRARY. It must be passed to most of the CMS routines. In general, if the routine gets information about a group, class, or element in the library, you must pass a library data block. NEVER ALTER THE CONTENTS OF THE LIBRARY DATA BLOCK, OR YOU MAY HARM THE CONTENTS OF YOUR CMS LIBRARY!!! See Figure 1 for an example of creating a library data block in VAX FORTRAN.

```
STRUCTURE /LIBRARY_DATA_BLOCK/
 UNION
  MAP
   INTEGER*4        %FILL(1:50)
  END MAP
  MAP
   INTEGER*4        LENGTH, STATUS
   INTEGER*4        DESCRIPTOR(1:2)
   INTEGER*4        %FILL(5:50)
  END MAP
 END UNION
END STRUCTURE
```

Figure 1 - the library data block

The fetch data block is five longwords in size and must be passed to the special routines for fetching an element line by line. It is used by CMS$FETCH_OPEN, CMS$FETCH_GET, and CMS$FETCH_CLOSE. See Figure 2 for an example of creating a fetch data block in VAX FORTRAN.

```
STRUCTURE /FETCH_DATA_BLOCK/
   INTEGER*4      %FILL(1:5)
END STRUCTURE
```

Figure 2 - the fetch data block

THE ROUTINES

Since there is an entry point into the CMS sharable image for each DCL-level command, most routines are simply named after the DCL command (e.g., CMS$SHOW_GENERATION). The notable exception is the RESERVE command. With the callable interface, an element is RESERVE'd by setting a flag used by the CMS$FETCH routine.

There are additional routines to:
- FETCH an element a line at a time
- translate strings returned from or given to CMS.

Figure 3 lists the routines currently available.

```
CMS$ANNOTATE               CMS$REMOVE_ELEMENT
CMS$DIFFERENCES            CMS$REMOVE_GENERATION
CMS$FETCH                  CMS$REMOVE_GROUP
CMS$FETCH_CLOSE            CMS$REPLACE
CMS$FETCH_GET              CMS$SET_LIBRARY
CMS$FETCH_OPEN             CMS$SHOW_CLASS
CMS$GET_STRING             CMS$SHOW_ELEMENT
CMS$INSERT_ELEMENT         CMS$SHOW_GENERATION
CMS$INSERT_GENERATION      CMS$SHOW_GROUP
CMS$INSERT_GROUP           CMS$SHOW_HISTORY
CMS$MODIFY_CLASS           CMS$SHOW_LIBRARY
CMS$MODIFY_ELEMENT         CMS$SHOW_RESERVATIONS
CMS$MODIFY_GROUP           CMS$SHOW_VERSION
CMS$MODIFY_LIBRARY         CMS$UNRESERVE
CMS$PUT_STRING             CMS$VERIFY
CMS$REMARK
```

Figure 3 - callable interface routines

The routines return statuses in the same manner as the run time library routines and system services - in R0. The status follows the standard rule that having the low order bit set corresponds to successful completion.

There is no $CMSDEF in FORSYSDEF (and probably not in other languages' text libraries); you must refer to the status codes with the %LOC function in FORTRAN (or its equivalent). There is no single list of possible return status codes in the documentation. Instead, the codes are listed as appropriate after each routine's description. The Figure 4 is a reasonably complete list.

```
CMS$_ABSTIM                CMS$_NODELETE
CMS$_CREATED               CMS$_NOELE
CMS$_CREATES               CMS$_NOFETCH
CMS$_DELETED               CMS$_NOFILE
CMS$_DELETIONS             CMS$_NOGRP
CMS$_DIFFERENT             CMS$_NOHIS
CMS$_EOF                   CMS$_NOINSERT
CMS$_ERRCREATES            CMS$_NOMODIFY
CMS$_ERRDELETIONS          CMS$_NORECOVER
CMS$_ERREPLACEMENTS        CMS$_NOREF
CMS$_ERRESERVATIONS        CMS$_NOREMARK
CMS$_ERRFETCHES            CMS$_NOREMOVAL
CMS$_ERRINSERTIONS         CMS$_NOREPAIR
CMS$_ERRMODIFIES           CMS$_NOREPLACE
CMS$_ERRREMOVALS           CMS$_NORMAL
CMS$_ERRUNRESERVES         CMS$_NOSINCE
CMS$_EXCLUDE               CMS$_NOTFOUND
CMS$_FETCHED               CMS$_NOUNRESERVE
CMS$_FETCHES               CMS$_NOVERIFY
CMS$_GENCREATED            CMS$_OPENIN1
CMS$_GENINSERTED           CMS$_OPENIN2
CMS$_GENNOINSERT           CMS$_OPENOUT
CMS$_GENNOREMOVE           CMS$_QUALCONFLICT
CMS$_GENNOTFOUND           CMS$_READIN
CMS$_GENREMOVED            CMS$_RECOVERED
CMS$_HISTDEL               CMS$_REMARK
CMS$_IDENTICAL             CMS$_REMOVALS
CMS$_ILLCLSNAM             CMS$_REMOVED
CMS$_ILLELEXP              CMS$_REPAIRED
CMS$_ILLGEN                CMS$_REPLACEMENTS
CMS$_ILLGRPNAM             CMS$_RESERVED
CMS$_INSERTED              CMS$_SEQUENCED
CMS$_INSERTIONS            CMS$_STOPPED
CMS$_INVFETDB              CMS$_TIMEORDER
CMS$_LIBSET                CMS$_UNFOUT
CMS$_MODIFICATIONS         CMS$_UNRESERVED
CMS$_MODIFIED              CMS$_UNRESERVES
CMS$_NOCLS                 CMS$_UNSUPFRMY
CMS$_NOCREATE              CMS$_USERERR
                           CMS$_VERIFIED
```

Figure 4 - return status codes

In addition to the return status codes, bit masks are used when specifying options to certain routines. Again, there is no one list of them, and they must be referenced by %LOC or its equivalent. The Figure 5 is a reasonably complete list.

```
CMS$M_CMD_COPY
CMS$M_CMD_CREATE
CMS$M_CMD_DELETE
CMS$M_CMD_FETCH
CMS$M_CMD_INSERT
CMS$M_CMD_MODIFY
CMS$M_CMD_REMARK
CMS$M_CMD_REMOVE
CMS$M_CMD_REPLACE
CMS$M_CMD_RESERVE
CMS$M_CMD_UNRESERVE
CMS$M_CMD_VERIFY
CMS$M_IGNORE_CASE
CMS$M_IGNORE_FORM
CMS$M_IGNORE_LEAD
CMS$M_IGNORE_SPACE
CMS$M_IGNORE_TRAIL
```

Figure 5 - bit masks

EXAMPLE: LIBED

LIBED is an example of using the callable interface to provide functions that the DCL interface does not have. LIBED is available on the L&T SIG tape.

LIBED is a simple CMS LIBrary EDitor. Along with the CMS interface, it uses the SMG runtime routines to format output and control the user interface. LIBED currently provides two capabilities:
- o ZOOM - Displays interesting information about an element generation.
- o VIEW - Types the element generation to the screen.

LIBED will help to demonstrate:

CMS$SET_LIBRARY
CMS$SHOW_GENERATION
CMS$GET_STRING
CMS$FETCH_OPEN
CMS$FETCH_GET
CMS$FETCH_CLOSE.

CMS$SET_LIBRARY - Prior calling most callable interface routines, you must first call CMS$SET_LIBRARY to initialize the library data block. Figure 6 is a fragment from LIBED that loops through a list of CMS element specifications to set up the data blocks for each. The library data block is LIBDB(NUM_WINDOWS), and the library name provided from the command line is in DISPLAY(NUM_WINDOWS).LIBRARY. MAKE_STRING translates the address of the descriptor in the library data block to a character string that will be used to lable the boarder of the windows that get displayed.

A key point of this example is that you may SET LIBRARY to as many libraries as you have allocated data blocks. This is not possible with the DCL interface. In addition, your program must use CMS$SET_LIBRARY prior to accessing the library. Using the DCL command CMS SET LIBRARY is not sufficient, nor is a CMS$SET_LIBRARY used from a previous run of the program.

CMS$SHOW_GENERATION - Once the library data blocks are initialized, your program is free to access the CMS library. LIBED uses CMS$SHOW_GENERATION to obtain information about the

element(s) specified on the command line, and to set up the SMG window(s). Figure 7 shows the use of CMS$SHOW_GENERATION with a user specified callback routine.

The arguments supplied are the library data block, the user routine (ADD_ELEMENT_TO_DISPLAY), a parameter to pass to that routine (in this case, an array element DISPLAY(I)), the element specification as specified on the command line (DISPLAY(I).ELEMENT), and the generation expression from the command line (DISPLAY(I).GENERATION). Note that DISPLAY is an array of VAX FORTRAN RECORDs. The remaining arguments' placeholders need not have been supplied; they are inserted for clarity.

"CHKLEN" is an integer function that returns the length of the string without its trailing blanks. If an element's name were "FOO.BAR", then passing CMS the element name "FOO.BAR      " would result in CMS's failure to find the element. The same generally holds true for other strings passed to CMS.

Figure 8 shows the format of the user supplied callback routine ADD_ELEMENT_TO_DISPLAY, and the use of CMS$GET_STRING. When CMS invokes the callback routine on behalf of your program, it will supply a predefined set of parameters to your routine. This includes the user parameter if you have specified one. Rather than pass the string descriptor of text that is supplied, CMS provides a string "id". CMS$GET_STRING is used to "decode" this string id and makes it available to your program. Times are passed as quadwords, and other flags are passed as longwords.

Callback routines (and user supplied message handlers) may do just about anything, as long as they do NOT:

- unwind the stack past their point of invocation
- call other CMS routines (except CMS$GET_STRING and CMS$PUT_STRING)

As a general rule, however, it is advised that callback routines should "live" for their side affects (like updating a screen, or modifying data in a table) and should do as little else as possible. There is no point in asking for trouble when working with your CMS library.

Given the DCL command line

```
$ libed *.for/lib=[uxdsybd02.cms]
```

the display on the terminal might look like Figure 9. The cursor would be positioned at the element name in the top left corner (A8TODB.FOR). Pressing "z" for ZOOM would result in a display like that in Figure 10, which displays "interesting" information about the particular generation of that CMS element.

Line-by-line FETCH'ing - The VIEW feature of LIBED is supported by the interface routines for line-by-line FETCH'ing: CMS$FETCH_OPEN, CMS$FETCH_GET, and CMS$FETCH_CLOSE. Figure 11 is a fragment that shows the setup of the fetch data block with CMS$FETCH_OPEN, accessing of the element with CMS$FETCH_GET, and the clean-up done by CMS$FETCH_CLOSE. As with the library data block, you may be working with as many elements at a time as you have allocated fetch data blocks.

Note that the library data block is not provided to these routines. The CMS$FETCH_OPEN call contains the library name, the name of the element, and its generation specification. As such, these three routines may be used independently of the other callable interface routines. Note also that CMS$FETCH_GET does not return the value CMS$_EOF as documented. Rather, it returns RMS$_EOF after the last line of the element has been returned.

The result of a VIEW on an element would appear as in Figure 12.

## CONCLUSIONS

There are many benefits of the interface as outlined by some of the possibilities already mentioned:

- Accessing CMS is done in the high-level language of your choice versus having DCL as the sole interface
- Like the CMS> command submode, you can save on image activations if you have a lot to do in the library
- UNLIKE the DCL version, your program may access AS MANY LIBRARIES AS YOU WISH (and have virtual memory for...)
- You may supply your own output routine to most routines
- The interface can be integrated with all the other tools VMS has to offer: SMG, RTL, SYS, TPU, Scan, etc.

There are some weak spots, however:

- The interface is not item list oriented. Every major function has its own specific call and ways of working. Thus, for "FETCH" you have CMS$FETCH; for "SHOW HISTORY", you have CMS$SHOW_HISTORY; for "ANNOTATE", you have CMS$ANNOTATE, etc. This can make reusing code more difficult. The item list method may be tedious at times, but it can provide a stable hook into the interface
- Output/callback routine arguments differ for each CMS$ routine
- Output passed to the user-supplied output routine for CMS$ANNOTATE includes the page headers you would see in a CMS produced .ANN file
- Error handling can be difficult. While you may provide a message handling routine which CMS will call on your behalf, you still need to do some digging to provide meaningful error recovery. CMS should provide more information when it reports errors to the user supplied routine
- No support in FORSYSDEF
- You can't call CMS routines from callback routines (except for CMS$GET_STRING and CMS$PUT_STRING)
- The latest version of the Callable Interface Manual is dated November 1984. There are a number of inaccuracies sure to confuse the beginning user: as noted earlier, return status codes are not always as documented

And there are some bugs:

- The RESERVATIONS argument to the user supplied output routine is never set

o When using an output routine, CMS$ANNOTATE FAILS after reaching the end of the element with the error:
%CMS-F-BUG, there is something wrong with CMS or something it calls
-CMS-F-BADIOBLENGTH, The passed iob has an invalid length

REFERENCES


VAX DEC/CMS Callable Interface Manual, AA-Z340A-TE, November 1984.


```
      num_windows = 0
      do while (cli$get_value('ELEMENT', element))
         num_windows = num_windows + 1
         display(num_windows).element = element

cc    Get the name of the library to be looked at.  (if not specified, it
cc    will be CMS$LIB.)
         cli_status = cli$get_value('LIBRARY', display(num_windows).library)
         cli_status = cli$get_value('GENERATION',
     +                 display(num_windows).generation)

         CMS_STATUS = CMS$SET_LIBRARY(
     +          LIBDB(NUM_WINDOWS),
     +          DISPLAY(NUM_WINDOWS).LIBRARY,
     +                  )        ! place holder for user message routine.

         call make_string( libdb(num_windows).descriptor, library )
         display(num_windows).library = library
                 :
      enddo
```

Figure 6 - using CMS$SET_LIBRARY

```
      do i = 1, num_windows
                 :
cc    Call CMS with the element and generation specification.  (wildcards
cc    and group names may result in more than one invocation of the output
cc    routine ADD_ELEMENT_TO_DISPLAY)
         CMS_STATUS = CMS$SHOW_GENERATION(
     +          LIBDB(I),
     +          ADD_ELEMENT_TO_DISPLAY, DISPLAY(I),
     +          DISPLAY(I).ELEMENT(1:CHKLEN(DISPLAY(I).ELEMENT)),
     +          DISPLAY(I).GENERATION(1:CHKLEN(DISPLAY(I).GENERATION)),
     +          ,        ! place holder for from_generation expression.
     +          ,        !   "     "    "    "  ancestors flag.
     +          ,        !   "     "    "    "  decendants flag.
     +          ,        !   "     "    "    "  class member list flag.
     +             )     !   "     "    "    "  user message routine.
                 :
      enddo
```

Figure 7 - using CMS$SHOW_GENERATION

```
       INTEGER FUNCTION ADD_ELEMENT_TO_DISPLAY( NEW_ELEMENT, LDB,
     +          DISPLAY,        ! the user supplied parameter
     +          ELEMENT_ID, GENERATION_ID, USER_NAME_ID,
     +          TRANS_TIME, CREATE_TIME, REVISION_TIME, REMARK_ID,
     +          CLASS_LIST_ID, FORMAT, ATTRIBUTES, REVISION_NUMBER,
     +          RESERVATIONS )
                 :
                 :
       CMS_STATUS = CMS$GET_STRING( ELEMENT_ID, ELEMENT_NAME )
```

Figure 8 - callback routines and CMS$GET_STRING

```
+-----------------------DISK$PG22:[UXDSYBD02.CMS], 1+-----------------------+
|A8TODB.FOR         ACDATA.FOR          ACOK.FOR            ADDASS.FOR        |
|ASTOB.FOR          BASTOI.FOR          BILD.FOR            BILDT.FOR         |
|BITOA.FOR          BLDTCB.FOR          BTOD.FOR            BUG.FOR           |
|CABORT.FOR         CACHER.FOR          CADARI.FOR          CADD.FOR          |
|CBLDEX.FOR         CBLDHD.FOR          CBLDRC.FOR          CBUG.FOR          |
|CCDCAS.FOR         CCLDBS.FOR          CCOCAS.FOR          CCODE.FOR         |
|CCSRCU.FOR         CCSRCW.FOR          CCTRAN.FOR          CD2A.FOR          |
|CDDOUT.FOR         CDIREC.FOR          CDRIST.FOR          CDSTAK.FOR        |
|CEOB.FOR           CERROR.FOR          CFLUDT.FOR          CFNDAC.FOR        |
|CFNDCH.FOR         CGENFN.FOR          CGINFO.FOR          CGJLGN.FOR        |
|CGNL.FOR           CHECK_SUM.FOR       CHKDEL.FOR          CHKFLG.FOR        |
|CHSCLN.FOR         CI.FOR              CINC.FOR            CITRAN.FOR        |
|CKAUTO.FOR         CKSRCU.FOR          CKSRCW.FOR          CLEAR.FOR         |
|CLINK.FOR          CLINK1.FOR          CLO2UP.FOR          CLTOCF.FOR        |
|CLZASS.FOR         CLZINF.FOR          CMDLOD.FOR          CMDSAV.FOR        |
|CPROPT.FOR         CRADIR.FOR          CRDJCL.FOR          CREATE.FOR        |
|CNWGEN.FOR         COPASS.FOR          CPAGES.FOR          CPARSE.FOR        |
|CRECPR.FOR         CS.FOR              CSBPRS.FOR          CSECCK.FOR        |
|CSEDIT.FOR         CSETUP.FOR          CSPCED.FOR          CSPSEG.FOR        |
|CSREAD.FOR         CSRTRV.FOR          CSTGEN.FOR          CSWRIT.FOR        |
|CTDRIV.FOR         CTXIOC.FOR          CUPDAT.FOR          CUPDIR.FOR        |
|CUPPR.FOR          CURSOR.FOR          CUTD.FOR            CUTDSN.FOR        |
-----------------------------------------------------------------------------
                                    Press ?, PF2, or "HELP" for Help
```

Figure 9 - LIBED startup display

```
+-----------------------DISK$PG22:[UXDSYBD02.CMS], 1+-----------------------+
|A8TODB.FOR        ACDATA.FOR        ACOK.FOR          ADDASS.FOR            |
|ASTOB.FOR         BASTOI.FOR        BILD.FOR          BILDT.FOR             |
|BITOA.FOR         BLDTCB.FOR        BTOD.FOR          BUG.FOR               |
|CABORT.FOR        CACHER.FOR        CADARI.FOR        CADD.FOR              |
|CBLDEX.+-----------------------A8TODB.FOR, 1+----------------------+        |
|CCDCAS.|Generation: 1                                              |        |
|CCSRCU.|Created by: U_SMITH                                        |        |
|CDDOUT.|Placed in library:  1-OCT-1986 15:09:14.67                 |        |
|CEOB.FO|File created:      26-AUG-1986 16:30:47.96                 |        |
|CFNDCH.|Remark: CPPR 33923                                         |        |
|CGNL.FO+-----------------------------------------------------------+        |
|CHSCLN.FOR        CI.FOR            CINC.FOR          CITRAN.FOR            |
|CKAUTO.FOR        CKSRCU.FOR        CKSRCW.FOR        CLEAR.FOR             |
|CLINK.FOR         CLINK1.FOR        CLO2UP.FOR        CLTOCF.FOR            |
|CLZASS.FOR        CLZINF.FOR        CMDLOD.FOR        CMDSAV.FOR            |
|CMDTRA.FOR        CMSGP.FOR         CNUMWD.FOR        CNVERT.FOR            |
|CNWGEN.FOR        COPASS.FOR        CPAGES.FOR        CPARSE.FOR            |
|CPROPT.FOR        CRADIR.FOR        CRDJCL.FOR        CREATE.FOR            |
|CRECPR.FOR        CS.FOR            CSBPRS.FOR        CSECCK.FOR            |
|CSEDIT.FOR        CSETUP.FOR        CSPCED.FOR        CSPSEG.FOR            |
|CSREAD.FOR        CSRTRV.FOR        CSTGEN.FOR        CSWRIT.FOR            |
|CTDRIV.FOR        CTXIOC.FOR        CUPDAT.FOR        CUPDIR.FOR            |
|CUPPR.FOR         CURSOR.FOR        CUTD.FOR          CUTDSN.FOR            |
+----------------------------------------------------------------------------+
                                       Press ?, PF2, or "HELP" for Help
```

Figure 10 - ZOOM output

```
cc Open the element.
      CMS_STAT = CMS$FETCH_OPEN(
     +                  FETDB,
     +                  DISPLAY.LIBRARY(1:CHKLEN(DISPLAY.LIBRARY)),
     +                  ELEMENT(1:CHKLEN(ELEMENT)),
     +                  DISPLAY.GENERATION(1:CHKLEN(DISPLAY.GENERATION)),
     +                  1,      ! no history.
     +                  1,      ! no notes.
     +                  GENERATION,
     +                    )     ! place holder for user message routine.
                  :
      done = .not. cms_stat
      more = .true.
      count = 0

cc Loop until EOF or until the user wants no more.
      do while (.not. done)

cc    FETCH a line of the element
         CMS_STAT = CMS$FETCH_GET( FETDB, STRING )
         count = count + 1

cc    Check to see if we've reache EOF.
cc       NOTE: CMS returns RMS$_EOF, not CMS$_EOF as documented.
         DONE = CMS_STAT .EQ. RMS$_EOF
                  :
      enddo

cc Close the element.
      CMS_STAT = CMS$FETCH_CLOSE( FETDB )
```

Figure 11 - FETCH'ing line-by-line

```
-------------------------------A8TODB.FOR, 1--------------------------------
      SUBROUTINE A8TODB( NAME, TYPE )
C***************************************************************************
C
C               Author: Randall Smith
C
C  Date of Last Update: 07/09/86 (1)
C
C     Revision History: (1) Created for CPPR 33923
C
C           Parameters: NAME  (IN/ ) - 8 char name to be stored
C                       TYPE  (IN/ ) - DD for directory, DI for data page/word
C
C           Local Data: TEMP   - Storable version of NAME
C
C             Function: Takes an 8 char name and stores it into the next 2 words
C                       of either the directory or data section of the SDB.
C
C***************************************************************************
C
      IMPLICIT NONE

----------------------------------------------------------------------------
                                                                       More?
```

Figure 12 - VIEW output

# NETWORKS SIG

# Developing a Message Bus for Integrating VMS High Speed Task to Task Communications

**Glen Macko**
Digital Equipment Corporation
West Hartford, CT

**Abstract**

This paper reviews the requirements and design decisions considered during the development of a general purpose facility for VMS task-to-task communications, the PAMS Message BUS. High throughput, integration of ALL communications paths, and ease of use were primary goals.

## Introduction

The intent of a Message Bus is to provide application designers and programmers an umbrella facility from which they can standardize their interfaces for peer-to-peer communications between programs. The umbrella can be very small and only handle local messages within a CPU, or the umbrella can be very large and encompass a wide variety of local messages, remote messages, remote networks, and miscellaneous events. Figure 1 attempts to show how extensively a message bus could extend its integration of messages and events to provide application programs with a consistent interface.

In looking at the world of communications, one can quickly become engulfed with an immense array of choices. As part of trying to enhance efficiency it is desirable to standardize on some type of communications. But when choosing between DECnet, OSAK(OSI), MAP, TCP/IP, X.400, and many others, there can be a large penalty to pay if a wrong choice is made. A properly developed message bus will present an environment that can migrate to any particular message transport protocol WITHOUT requiring conversion of application code.

A large number of software projects involve the building of applications using components that cooperate but execute as separate processes or tasks. This is generally known as a multi-task application. A detailed analysis of the benefits of developing multi-task versus single-task applications is beyond the scope of this paper, but a brief list of some the reasons for multi-tasking include the following:

- Modular Design to Parallel the Problem Being Solved
- Breaking Job into Workable Units for Parallel Programming Development
- Prioritizing Execution
- Context isolation to prevent one program bug from stopping an entire application
- Geographically Distributed Users
- Distributing Computing Power to the User's Desk via Workstation
- Parallel Computing over a Cluster or a Network
- Parallel Computing over a multiprocessor CPU such as an VAX 8800

Once you have made the decision to build a multi-task application, you must provide a communication mechanism for these cooperating tasks. Choosing a communications mechanism can be time consuming and inherently risky if a mechanism is chosen from which the industry later moves away or the mechanism is limited in scope and is unable to grow as the application gets more successful. Even applications using DECnet task-to-task may choose to migrate to the OSI call interfaces as Digital Equipment Corporation moves toward its goal of merging the the DNA architecture with the OSI protocols.

The primary purpose of a message bus is to provide a communications mechanism that can integrate the USER INTERFACE with all important communications mechanisms required today and easily expand to accommodate future mechanisms without application code changes.

Multi-task applications typically require the management of individual logical links or sessions between tasks. As more and more point-to-point sessions are established, the management and control of the communications becomes difficult. Figure 2 demonstrates some sample components of a multi-task application without a message bus. By integrating with a message bus, the lines of control are handled through a single path to the message bus as can be seen in Figure 3. This style of integrating through a single control path is conceptually similar to the CI and Ethernet communications hardware.

The concept of a message bus is not new. Any medium to large sized project usually assigns personnel to be in charge of communications. In fact, this person(s) is

| USER CODE | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| PAMS MESSAGE BUS ... | | | | | | | | |
| Local Message Global Section | Message Simulator | DECnet<br>- - - -<br>VAX PAMS<br>RSX PAMS | Direct Ethernet I/O<br>- - - -<br>VAX PAMS<br>RSX PAMS | User Setable Timers | VMS Mail box | BASE-WAY | TDMS Asyn Read | Direct DMR I/O |
| (1) | (1) | (1) | (1) | (1) | (2) | (2) | (2) | (2) |

| USER CODE | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| PAMS MESSAGE BUS ... | | | | | | | | |
| Direct TT Line I/O | SMG$... Terminal Read | DECnet<br>- - - -<br>DOS PAMS<br>ELN PAMS | Direct Ethernet I/O<br>- - - -<br>DOS PAMS<br>ELN PAMS<br>RT PAMS<br>TOPS PAMS | TCP/IP | Direct SCS/CI I/O | NSI Hyper-Channel | SNA LU6.2 | SNA 3270 Data Stream |
| (2) | (2) | (3) | (3) | (3) | (3) | (3) | (3) | (3) |

| USER CODE | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| PAMS MESSAGE BUS | | | | | | | | |
| Message Router<br>- - - -<br>DECmail<br>All-in-1 mail<br>X.400 | Lock Manager Grants | OSAK OSI Level 5 | VOTS OSI Level 4 | MAP | TOP | User Device #1 | User Device #2 | |
| (3) | (3) | (3) | (3) | (3) | (3) | (3) | (3) | |

(1) Primary PAMS Data Paths
(2) Integrated with PAMS as of November 1986
(3) Potiential Paths for Future Integration

**Figure 1:** Message Bus Umbrella over Messages and Events

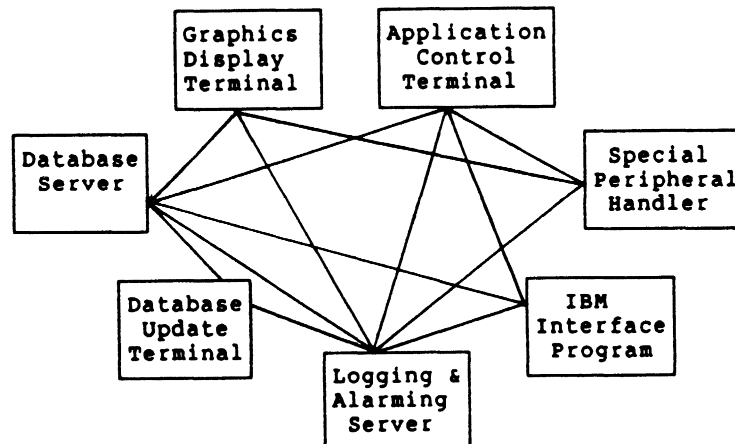



**Figure 2:** Multi-task Connectivity with Session Management

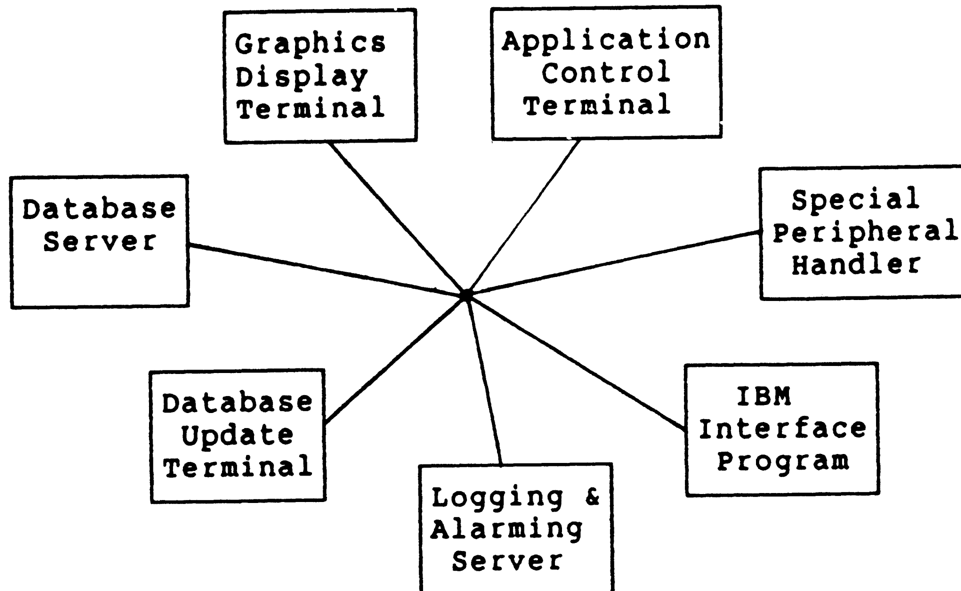



**Figure 3:** Multi-task Connectivity with a Message Bus

often one of the best systems programmers since communications is considered one of the most difficult elements of a project. In order to isolate the rest of the programming staff from the intricacies of the communications, a callable interface is often developed that will provide a friendly interface to the world of communications. Such a callable interface is referred to as a Message Bus in this paper.

The PAMS Message Bus was developed in such a manner. The key difference between the PAMS Message Bus and other project message buses is that the PAMS Message Bus was also designed to be a generic implementation that would easily operate in all or most application projects.

### Requirements of a Generic Message Bus

As part of choosing or developing a Message Bus, it is important to evaluate the entire range of requirements that might be important for messaging. During the design and development of the PAMS Message Bus we identified the requirements listed below. Unless otherwise noted, all the requirements listed are satisfied by the PAMS Message Bus.

- Easy to Use – The primary purpose of a message bus is to provide an easy to use data path between tasks. The application code wants the message bus to handle all the communications problems of connectivity, session management, and resource control. A message bus should provide the capability for a program to do a majority (or all) of its communications with the three Calls listed below.
  - CALL DECLARE_PROCESS( ... ) – This call should declare the process to the message bus and allow access to any other process that has declared itself to the message bus.
  - CALL RECEIVE_MESSAGE_WAIT( ... ) – This call is asking for the next message, unit-of-work, or transaction to be received so that the appropriate processing can occur. If a message is not available, the task will wait for the next message to arrive.
  - CALL SEND_MESSAGE( ... ) – This call allows the user to send a message to any target task known by the message system. The target task may be local, remote, a special hardware device, or even a different message facility such as IBM SNA LU6.2.
- Reliable / Bulletproof – Reliability is a goal for all software, but is particularly critical for a message bus since so many of the application components rely on its correct operation. Too often, all phases of a project are started at once and the message facility is not ready until many months into the programming effort. This presents a large impediment to the rest of the project staff as they are forced to develop code that cannot be tested while it is still fresh in the programmer's mind. Or, the code must be tested with custom "stub" software that must later be converted to the final facility before integration can start.
- Integrated Message/Event Reception – By developing programs that are driven primarily through a single integrated message interface, it becomes very easy to build systems that operate in a transaction driven fashion. This allows the program to request the next

"unit of work" by receiving a message and then dispatching to the appropriate action routines to complete the work item. Transaction driven system lend themselves to easily defined test cases and predictable behavior.

If an integrated reception facility is NOT available, more complex programming techniques are required such as 1) polling I/O paths for completions, and 2) multiple execution threads with Asynchronous System Traps (ASTs). ASTs can be particularly dangerous due to frequent race conditions that can be extremely difficult to reproduce and may only occur during heavily loaded production conditions.

- Fast – The importance of delivery speed will vary between applications according to the volume of messages compared to the amount of data processing. It is sufficient to say that it is important to application design that the communication mechanism be perceived as being fast. In that way, applications can be designed in a fashion that is most conducive to the natural flow of the data processing steps.

  Too often, task-to-task communications is viewed as having a high overhead and is avoided during system design phases. Resulting implementations often over-concentrate on processing speed which detracts from application elegance and results in a system that is hard to maintain and difficult to expand its function and capacity. By having high speed communications, the project team can build an elegant system that maintains quick response times.

- Network Transparency – A message bus offers network transparency by delivering messages both locally and remotely with exactly the same code interface. This allows program modules to be rearranged and redispersed to other nodes without coding changes.

- Priority Queuing – A message bus by definition is a queued transaction system. Some applications will routinely execute with very small queue depths. Other applications may use the queues to absorb various load peaks that might occur. By having priority queuing in the message bus, time-critical messages can be queued ahead of other less critical messages.

- Integrate with other message paths – The computing world is filled with approaches to data communications and message delivery. There will always be requirements to integrate an extra message capability that is not directly supportted by the "standard" you have chosen. A message bus that provides tools for custom integration of new message paths gives you the power to maintain a consistent, integrated approach to programming, testing, and maintaining applications.

- Integration with other events – Most systems that are designed to be message driven are in fact asynchronous event driven systems. In order to service the events in an orderly fashion, they are queued. Other events besides true messages can be queued as messages to provide an orderly transaction processing. Examples of non-message events that can be integrated with a message bus include: 1) terminal input completion, 2) timer expiration, and 3) special hardware I/O completion. See Figure 1 for some of the actual events that have been integrated with the PAMS Message Bus.

- Selective Reception for Request/Response dialogues – There are many cases where a program wishes to receive a message only from a specific program instead of the next queued message. This requirement is important for request/response sequences with a server process. It is also important when a program must wait for an acknowledgment from a specific program to ensure that a command has been completed and does not want to be disturbed by messages queued from other programs.

- Receive Timeout – Some programs may have special duties that preclude long delays while waiting for messages to be received. For those needs, it is important to have a timeout variable on the receive-with-wait Call. This requirement seems so basic that it is quite surprising to find that neither mailboxes nor DECnet task-to-task support a timeout argument within their QIO calls.

- Timer Event Delivery – If an application program needs to set special "reminder" timers, it is important to receive notification through the same mechanism as all other transactions are received, namely the message path. In this way, the message mechanism really becomes a mechanism for reception of all events. The PAMS Message Bus has special Calls to set timers and cancel timers. When a timer expires, it is converted to a message and delivered to the user. By definition, timers are time-critical and are always delivered with the highest priority.

- Support all major languages – A message bus should easily integrate with any programming language. This feature is relatively easy to implement due to the VMS calling convention. The PAMS message bus directly supports the following languages with INCLUDE files: Ada, Basic, C, Cobol, Fortran, Pascal, and PL/1.

- Message Simulator – A message simulator is a very powerful productivity tool. It allows a programmer to quickly and easily build test messages that can be used immediately for module testing long before other programs are ready to send the messages. The programmer is therefore able to thoroughly test his code the same day that it is written. With the PAMS Message Bus, the message simulator is linked into your program image so that message simulation occurs di-

rectly in you process WITHOUT having to run a detached or spawned process.

- Message Tracing – Message tracing is also a programmer productivity tool. It allows the programmer to view the messages as they are processed for both input and output and quickly identify program flow problems. The PAMS Message Bus translates the traced message into ASCII text that can be viewed at the executing terminal and/or written to a log file. In fact, the format is exactly the same as the Message Simulator format so that messages trapped in a log file can be later used for message input with the message simulator.

  In order to further enhance the readability of the traced messages, elements of the message header are translated into the symbolic names that have been previously chosen by the user for the source process name, the target process name, the message type and the message class. These symbolic names are the same ones used within the program and available through INCLUDE files.

- Handle large messages – If a message system does not support large messages, it can be a significant programming chore to break up a user data block and ship it as multiple messages that are reattached by the target process. The PAMS Message Bus relieves this chore by supporting messages of up to 32,000 bytes.

- Multiple Message Delivery Modes – A large number of rule sets can apply to how a message should be delivered and what to do if message delivery is blocked. Some environments such as real-time test data acquisition are tuned for high-speed, no-recovery environments where it is more cost effective to rerun a test in case of failure rather than build extensive recovery software and hardware. On the opposite extreme are applications such as electronic funds transfer where transaction completion must be absolutely guaranteed. Many high availability systems can use statistical analysis to project failure rates as small as one/year, one/decade, or even one/century. As part of supporting this wide variety of delivery needs, we have identified the following four delivery modes:

  - Datagram – A datagram delivery is the lowest cost delivery where the message is delivered on a "best try" basis. Loss of the target node, or intermediate links will result in loss of the message. In this mode, the sender immediately passes on the message and is never blocked from continuing execution.
  - Return to sender – Allows the sender to request that a message be returned to itself in case the message can not be delivered to the target process. As with a datagram, the sender's execution is never blocked. However, if the target node is not available or the target process is not available, the message will be returned to the sending process to allow the sender to take some recovery action.
  - Synchronized Queue to Target – This mode will synchronize a program's execution with the successful queuing of the message on the target process queue. In other words, the sender will be blocked until the message is on the target queue. With the PAMS Message Bus, queuing to a local process will happen immediately without context switches since the local message system is implemented with Global Sections. Remote message delivery will cause the sending process to be blocked until the message is successfully queued to the target and an internal acknowledgment is returned to the sender.
  - Journaled Guaranteed Delivery – In order to fully guarantee the delivery of the message, it must be 1) journaled to a non-volatile medium (e.g. magnetic disk), 2) delivered to the target process (at a later time if target not currently available), and 3) the user task must acknowledge the completion of the handling of the message via a "commit" operation similar to a Data Base Commit. When using journaled delivery, the user program must devise a scheme to handle the potiential for a duplicate message. This can occur if there is a CPU failure during the small window between the time that the user completes the servicing of the message and the time that the Commit is completed. This level of guaranteed delivery has not yet been implemented within the PAMS Message Bus but is now (11/01/86) under consideration for the next major release of the PAMS Message Bus.

- Monitoring and management tools – Additional tools should be available to monitor the state of the message bus and to determine the state of programs declared to the message bus. The PAMS Message Bus provides this capability in two ways. One method is by using an interactive program to display message counts and buffer usage in a fashion similar to the DECnet NCP program. A second capability within PAMS is to programmably request information from the message bus. The current programmable interface allows a program to: 1) request a list of all other programs known by the message bus and 2) be added to a "automatic notify list" to be notified whenever a program declares itself to the message bus or exits from the message bus. This automatic notify feature allows an application control program to immediately be notified via a message that a fellow process has exited and that a recovery action should be initiated.

## PAMS Design Decisions

In designing the PAMS Message Bus, many decisions were made on topics that must be considered for anyone building a communications facility or message bus. The following section reviews these design decisions.

### Local Message Transport Mechanism

There are various VMS tools available for implementing a local CPU message transport mechanism. The list of tools includes, but is not limited to: 1) Mailboxes, 2) DECnet task-to-task, 3) Global Sections, 4) Hibernate/Wake, 5) Event flags, 6) Logical names, 7) Lock manager, 8) Disk files (RMS/DBMS/RDB), and others.

Some of these mechanisms can quickly be dismissed as being far too slow or inflexible. For these reasons, it is easy to dismiss logical names and disk files.

Of the remaining facilities, only three have the ability to pass data while the remaining can be used for process synchronization. The three mechanisms of passing data are mailboxes, DECnet task-to-task, and global sections. Mailboxes and DECnet have built-in process synchronization mechanisms, while global sections provide only a data passing mechanism and must be implemented together with a process synchronization mechanism.

Since high throughput is a primary requirement of the PAMS Message Bus, a timing study was done to compare the raw throughput rates achievable from Mailboxes, DECnet, and Global Sections. Global sections were matched with Hibernate/Wake system services to provide process synchronization. The benchmark environment consisted of a VAX-11/780 with 8 Mbytes of main memory, a floating point accelerator, VAX/VMS V4.2, and DECnet-VAX. The test was run with message sizes of 20 bytes, 500 bytes and 10,000 bytes.

Two programs were built, a loop program and an echo program. The loop program would initiate the message transfers by sending one message and then executing a "receive and wait" operation until a response would come back. When the response arrived, the loop program would then continue the cycle of sending a message and waiting for the response.

The echo program would execute a receive-and-wait operation for a message to arrive and would then "echo" the message back to the loop program. Since the programs are run within the single CPU, they saturate the CPU with 0% NULL time and the message rates are calculated by dividing elapsed wall time by the number of messages deliverred. Each round trip loop is considerred 2 message deliveries.

The results of this timing study are summarized in the graph in Figure 4. Higher throughput rates could have been achieved by looping multiple messages simultaneously. This would have reduced the number of context switches and improved the cache hits. However, I do not believe that the typical application environment would be sending multiple messages in rapid order and therefore I chose to use a worse case scenario. As can be seen from the graph, Mailboxes had 1.8 times the throughput of DECnet for small messages and 3.6 times the throughput for large messages. When comparing Global Sections to Mailboxes, Global Sections had 3.8 times the throughput of Mailboxes for small messages and a whopping 19.5 times the throughput of Mailboxes for large messages.

Global Sections are able to maintain a constant throughput independent of the message size since the buffers are not copied from memory to memory–only buffer pointers are passed between the programs.

Global sections were a clear winner in the area of throughput. In addition, global sections provided some additional benefits for easily implementing some monitoring requirements such as tracking the number of messages sent, received, and pending. Therefore, global sections were chosen as the transport mechanism for local messages.

One drawback of dealing with global sections is the fact that all processes map themselves to the shared memory and an errant process could destroy data buffers of other users. To prevent this occurrence, the PAMS Message Bus optionally protects its buffers by allowing only EXEC mode access to the global sections. In this way, only the PAMS code within the program image is able to access the buffers and transfer the data from the global section buffer to a private user buffer. All user code will receive an "Access Violation" if it attempts to directly address the global sections.

### Remote Message Delivery Mechanisms

In evaluating the need for remote message delivery, it quickly became evident that no single interface could handle all environments. Therefore, rather than try to restrict the capabilities, it was decided to develop an interface that could accommodate any remote mechanism needed today or possibly in the future.

The PAMS Message Bus therefore built its system in a fashion that allowed the primary remote data paths to flow through DECnet links while data could optionally flow through any other message interface desired. Currently (November, 1986), PAMS has interface routines for the message paths listed below. Virtually any other message mechanism can also be integrated by using the tools provided with PAMS.

- DECnet task-to-task
- Direct Ethernet I/O
- Direct DMR I/O
- Direct Asynchronous Line I/O
- Baseway Application Bus

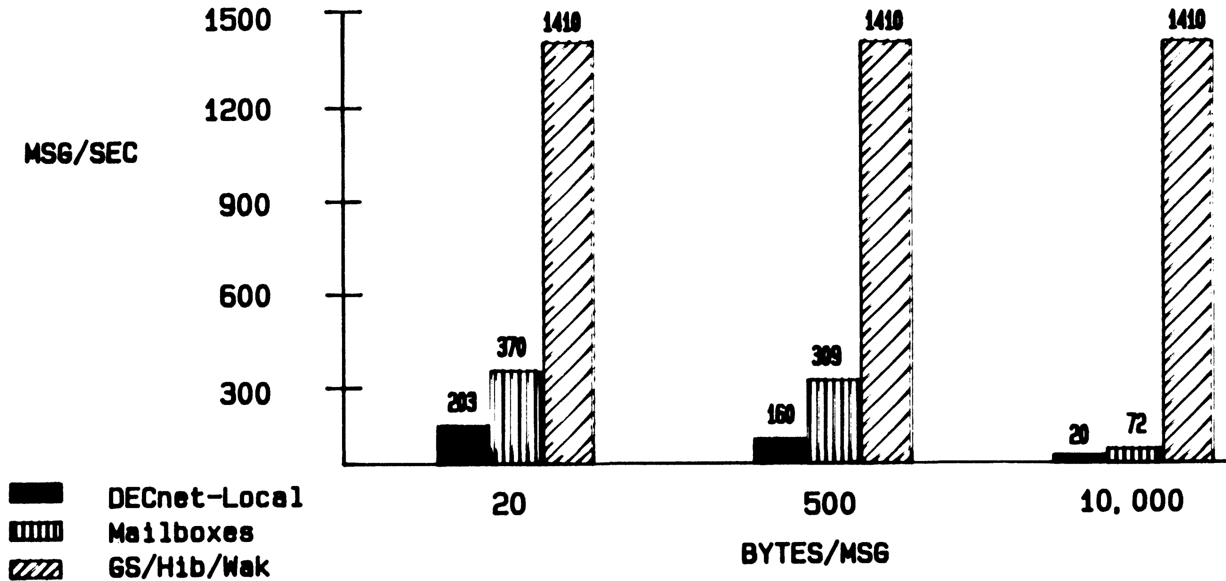



**Figure 4:** Throughput of Local Message Tools (VAX 780)

### Enhancing Ethernet Beyond Level 2 (Data Link Layer)

An interface for direct Ethernet I/O has been developed to provide higher throughput than what is available through the DECnet interface to Ethernet. This presented a number of problems since the Level 2 Ethernet protocol is rather limited in its ability to send large messages and to guarantee delivery is completed.

As part of developing a protocol to overcome these deficiencies, the following features were developed in support of direct Ethernet I/O:

- Chaining 1500 byte segments into 32,000 byte messages
- Multiplexing up to 100 Ethernet sessions from 1 set of subroutines
- Automatic Connection Mechanism – The code will continually attempt to establish a connection without intervention by the user code.
- Link Loss Detection via Heartbeat Messages – A user selectable rate of heartbeat messages can be sent so that the loss of a partner can be quickly detected.
- Optional Message Acknowledgment – If desired, the Ethernet interface will queue a message to the target process and wait for an internal acknowledgment before returning from a send message operation.
- Partner Locating Service – During the connection phase, the target Ethernet address does not have to be known. Instead, both partners can enable themselves for a multicast read until they find each other and then switch to using the physical Ethernet address.

### Comparison of PAMS to Other Message Busses

- VAXELN Message Services – The message services of the VAXELN operating system are a very powerful and elegant implementation of a message bus. One can speculate that VAXELN designers built a very nice messaging facility into VAXELN after recognizing the shortcomings of messaging with mailboxes in VAX/VMS.
- BASEWAY Application Bus – Baseway is targeted for the manufacturing market and not appropriate as a generic message bus. The messaging portion of Baseway provides no tools to integrate other message systems, does not support large messages, and is substantially slower than the PAMS Message Bus. Figure 5 displays a comparison of message rates between Baseway and PAMS. The benchmark environment con-

sisted of a VAX-11/780 with 8 Mbytes of main memory, a floating point accelerator, and VAX/VMS V4.2.

- VAX DEC/MAP – As of this writing, VAX DEC/MAP has just been announced by Digital Equipment Corporation. Although a very powerful message bus, VAX DEC/MAP is targeted for the factory automation market. VAX DEC/MAP V1.0 does not integrate with other Networks–not even DECnet. In addition, special hardware (KMS11, Concord Communications Inc Translator, and a broadband cable network) are required for its operation.

- Message Router for VMS – Message Router has a primary focus for electronic mail store-and-forward and is not oriented for the real-time messaging market. It would be more appropriately titled the "Mail Router".

- VAX ACMS Server Mechanism – ACMS has a powerful mechanism for developing distributed applications, but ACMS is only appropriate at sites that wish to adopt the entire ACMS environment.

- UNIVAC MCB-1100 (Message Control Bank) – UNIVAC has its own messaging bus known as the MCB-1100. This is an example of another industry implementation of a message bus, but unfortunately MCB-1100 will not run on VAX/VMS.

- ITT's VCF (VAX Communication Facility) – A previous DECUS Symposium paper was submitted by Howard Kilpatrick at ITT titled "A Fast Inter-Process Communication Facility for VMS". Unfortunately, it is not for sale to other companies.

- Hundreds or thousands of project-specific implementations – A large portion of the medium to large VMS projects that require task-to-task communications have implemented some sort of callable interface as a message bus to isolate the communications from the application programmer. Unfortunately these facilities are usually customized to the project and NOT suitable for the next project. In addition, few or none have the rich set of features identified in this paper and available with the PAMS Message Bus.

### Applications Using the PAMS Message Bus

- High speed data acquisition
- Defense industry real-time needs
- Network-wide transfer of Graphics/Pixel Images
- High volume access of database servers
- High volume Ethernet transfers
- Workstation access of VAXcluster databases
- Factory control and data collection

### Future Directions

Although the features of the PAMS Message Bus are comprehensive, there are still areas for enhancements. Listed here are the areas that are being evaluated as of this writing.

- Higher Throughput – We believe that the maximum throughput of the PAMS Message Bus can be roughly doubled from the current 300 msg/sec to 600 msg/sec on a VAX-11/780 by a careful review of the code and selective conversion of some code from PL/1 to Macro-32. As noted in the timing study graph, the theoretical throughput limit using global sections is 1410 msg/sec.

- Selective Guaranteed Delivery – Many applications require the capability to guarantee that messages will get delivered and processed even if the target process is not currently available or the target process aborts before completing the operation. Since all messages may not be critical, this feature would be selectable at the time the message is sent.

- Connectivity to other Message Mechanisms – A wide range of interfaces to remote message mechanisms may become packaged with PAMS. A sample of those areas being evaluated are listed below. Even without packaged software to service these functions, an integrating message bus such as PAMS allows the users to add their own features independent of message bus engineering.
  - Connectivity to MS/DOS via direct Ethernet I/O or DECnet-DOS,
  - Connectivity to VAXELN, TOPS-10, TOPS-20, and RT-11 via direct Ethernet I/O,
  - MAP,
  - Direct SCS/CI I/O,
  - NSI Hyperchannel connection to IBM systems,
  - SNA Gateway connection to IBM systems via LU6.2 or 3270 data stream,
  - TCP/IP,
  - Electronic Mail via Message Router.

- Interfacing with a higher performance Ethernet Driver – The current QIO interface to the Ethernet Controllers is designed to operate strictly according to the Ethernet Data Link Layer interface. As such, large user messages must be broken up into segments no larger than 1500 bytes and individually sent to the device driver with a QIO. The QIO overhead can become excessive in some applications. Rather than require that a larger CPU be purchased, substantial CPU cycles can be saved by sending a single QIO with a message up to 32,000 bytes in it. A specialized Ethernet driver could then cycle through the multiple 1500 byte segments WITHOUT running back and forth between the device driver and user code.

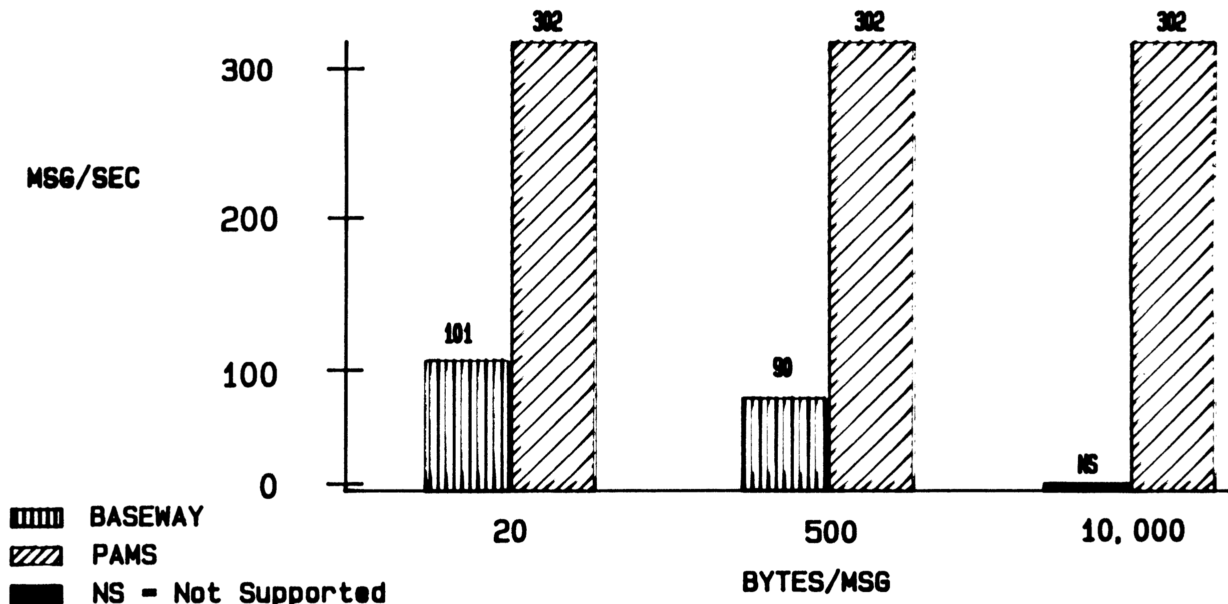



**Figure 5:** Throughput of PAMS versus BASEWAY (VAX 780)

## Conclusions

Peer-to-peer communications is becoming increasingly important as applications and databases become more complex and more distributed. Through proper planning, communications can be integrated to avoid a strangling patchwork that might otherwise occur.

Software development costs are exploding and it is important to both reduce these direct costs and to build applications that have long life cycles with minimal maintanance effort. By developing applications based on an integrating message bus, applications can be 1) completed sooner, 2) migrated easily, and 3) grow into new technologies without additional changes to user code.

## Acknowledgements

There have been a wide range of Digital Equipment Corporation employees and Digital Equipment customers that have contributed to the concepts of a message bus and the requirements of the PAMS Message Bus. The author thanks all of them for their contributions. Specifically, the author thanks Martin Michelsen, project leader of the PAMS Message Bus, for his help in preparing the paper and providing benchmark data. The author also thanks Randy Skelding as a longterm user of PAMS who has continually pushed for more features within PAMS to make the building of applications easier.

## References

"VAX/VMS System Services Reference Manual", Digital Equipment Corporation, April 1986

"VAX/VMS I/O User's Reference Manual", Digital Equipment Corporation, April 1986

Kilpatrick, Howard, "A Fast Inter-Process Communication Facility for VMS", *Proceedings of the Digital Equipment Users Society*, USA Fall 1984.

"VAX Realtime User's Guide", Digital Equipment Corporation, October 1986

"VAXELN User's Guide", Digital Equipment Corporation, March 1985

"VAX PAMS User's Guide", Digital Equipment Corporation, September 1986.

"VAX PAMS Operations Manual", Digital Equipment Corporation, September 1986.

# Network Print Servers

R.E. McGee
W.V. Dixon
General Electric Corporate Research and Development
Schenectady, New York

**Abstract**

The VMS print command has little support for network print jobs. Print features improve in a cluster environment, but clusters have their own limitations. At past symposia speakers have described DECnet applications to extend the basic VMS print command. This paper describes the design and implementation of the VAX portion of our network print server. The VAX portion of our print server is based upon non-transparent task-to-task DECnet communications and is implemented as multi-threaded code; it is the backbone of our print service. We discuss the overall architecture and some of the implementation difficulties. Security, resource utilization, and performance issues are considered. We also describe the natural evolution of the work into a generic network server.

## INTRODUCTION

This paper is not a general treatise on network print servers. Rather, it is a description of a specific approach taken at the General Electric Corporate Research and Development Center, ( hereafter abbreviated CRD ), to distribute printer access in a heterogeneous network computing environment. We refer to this distributed printer access as the network print service and the thing that provides this capability as the Print Server, or Server. The paper will be divided into four major sections. The first section will briefly describe the computing environment at CRD and the problem of printer access that this project attempts to solve. The second section discusses the general, technical approach taken in the development of the Print Server. The third section will discuss some specific technical issues that have been addressed thus far in its development. The final section will summarize the results and current status of the Print Server project.

## 1.0 BACKROUND

### 1.1 The Problem

Perhaps the best word to describe the computing environment at CRD is "diverse". A wide variety of machines populate our local area network ( LAN ) as depicted in Figure 1; the numbers in parentheses in this picture denote the approximate number of each type of machine on CRD's LAN. Some of these computers have associated printers, some do not. Printers are either attached to a computer that resides on the LAN ( i.e. the printer "belongs" to that computer ), or the printer is attached directly to the

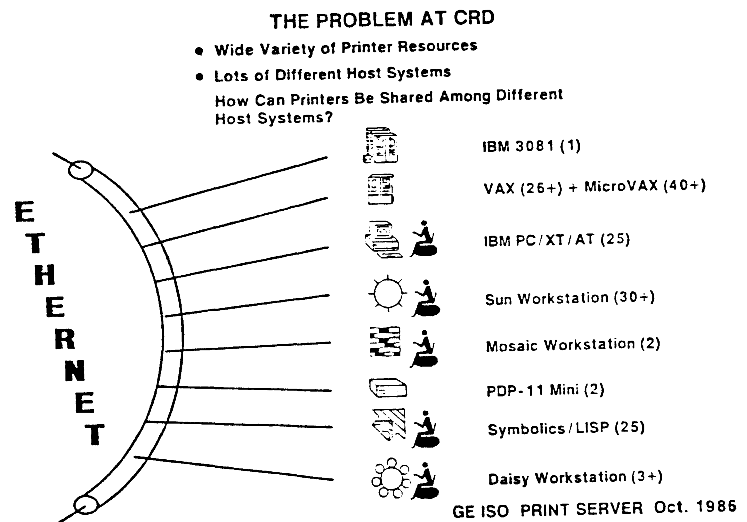



**Figure 1:** A survey of host systems on the CRD LAN.

LAN. With this configuration, printers aren't always readily accessible due to the restrictions imposed by physical connection. For example, users of a VAX/VMS system today may not be able to get output from a printer attached to another VMS system, because the user doesn't have an account on that machine, or the machine is not part of a common cluster. Thus, users may be constrained to use a small number of printers which, because of their location or print quality, may be inconvenient to use.

### 1.2 The Goal

The goal of the Print Server project is to make printers accessible to users on different host machines regardless of printer connectivity. Thus if a user is currently using machine "A" and the desired printer is attached to machine "B", the Print Server provides the means to use it.

Distributing printer access to users across the network has at least three advantages:

1. Money. If one or two printers can be shared among several groups of people (each of which perhaps uses a different computer on the network), then a network print service would preclude the need to buy a printer for each group.
2. Print jobs can be routed to printers best suited for the job type. For example, a programmer wanting to get a printout of a 100-page source listing could route the job to a high-speed line printer instead of a default slower printer that may be attached to the system.
3. People who use computers that are on a LAN tend to view the network as "the system". By providing transparent printing across the network this perception is made somewhat concrete.

## 2.0 GENERAL TECHNICAL APPROACH

### 2.1 Defining the Print Service

A Print Server can mean many things to many people. In defining *our* Print Server, we've tried to take various viewpoints.

From the developers' point of view, a Print Server is software that is distributed across several nodes on the local area network, and which provides users with the capability to access printers they normally could not use. Additionally, the Print Server can be defined in terms of the features it provides. Among these are that a user should be able to print a second-party file to a third-party printer. This means that if a user is using machine "A", he/she should be able to print a file residing on machine "B" to a printer attached to machine "C", where A, B, and C are distinct computers. Since the Print Server is distributed across a heterogeneous network ( i.e. lots of different *types* of host systems ), it must support several different machines such as PCs, Suns, VAXes, and others. And like any software project, the Print Server should be extensible in the sense that new features can be added as the need arises.

From the user's point of view, the Print Server manifests itself in the form of a new print command. This new command should be syntactically similar to the print command the user is already familiar with, and should provide the same capability. Response should be interactive, providing for wildcard confirmation and immediate notification of error conditions, when possible. If interactive response isn't possible, then the user should be notified by mail of the error condition. Once the print request is submitted, the user should be able to query the status of the print request locally to determine when the print job completes.

From the System Manager's point of view, the Print Server should provide its service without degrading the performance of the computer on which it is running. The Print Server should also consistently and gracefully recover from system shutdown or link failure. Print jobs should have a limited lifespan, so that if they are unable to be processed, ( i.e. printer is down ), they can be deleted. And, since printers are in effect made openly accessible to any user on the network, the System Manager should have the means to restrict access to printers.

Each of these three points of view have contributed to our understanding of what is expected of a network print service, and accordingly influenced our development approach.

### 2.2 Some Considerations

Before work even began on the Print Server, some observations were made that had a direct influence on the approach taken to tackle this project. One consideration is that the largest percentage of computer users at CRD are VAX users. Not surprisingly, the printers most-often used are those attached to the VAXes. Another consideration is the VAX itself. With a large virtual address space and disk capacity, it provides a good environment to do substantial software development. Another plus is that it supports other protocols (e.g. TCP), which is important given that the service being developed must accomodate users on systems that don't use DECnet. Apart from these considerations, the developers have more expertise with VAX/VMS than with any other environment. All of these factors contributed to our choice of the VAX as a development machine, and in fact as an integral part of the network print service architecture.

### 2.3 Development Approach

In order to provide network print service to users in a timely fashion, we've decided to implement the Print Server in a phased approach. Each phase will either increase the user-base for the network print service, or will increase the scope of printers made accessible by the print service. In all, there are four development phases. The

first phase establishes a "communications backbone" upon which the remaining three phases will be built. Completion of this phase will provide users with the ability to print files transparently from a given VAX running the Print Server to a printer attached to any VAX also running the Print Server software.

The second phase of the Print Server project establishes gateways to the IBM and Unix worlds in order that users from the first phase (i.e. VAX/VMS users) can access TCP/IP and IBM printers.

The third phase extends the user-base by providing Unix and IBM users with the capability to access printers from phases one and two. For example, at the completion of this phase, any Unix user would be able to send a print job to any printer attached to any VAX/VMS system running the Print Server.

The fourth and final phase brings Apple MacIntosh users in the user domain. Thus the completion of this phase will give a MacIntosh user the ability to print files on any printer made accessible in the first three phases of the project.

The development environment for the Print Server is pictured in Figure 2. Note that this environment does

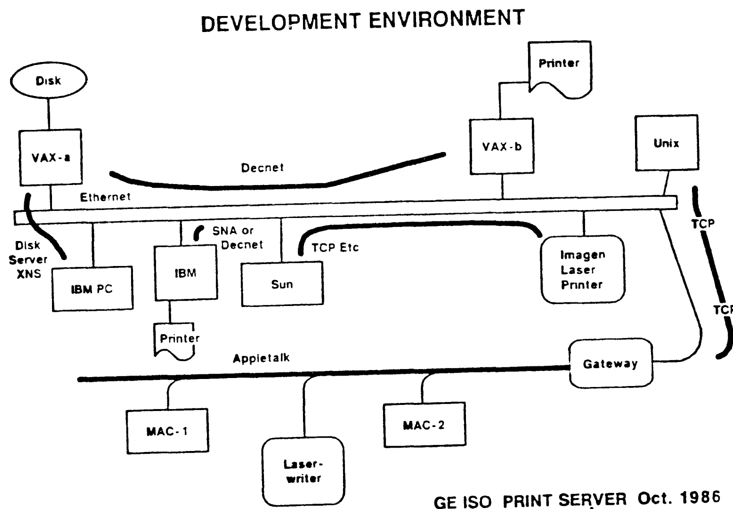



**Figure 2:** Development Environment for Print Server.

not include all the types of machines that are available on CRD's network from Figure 1 . However, this subset of users, though not exhaustive, does comprise the majority of users at CRD, and provides what we believe is a reasonable environment for which to aim.

### 2.4 Phase I Solution

There are several approaches one could take to provide transparent print capabilities across VAX/VMS systems on a local area network.

One approach is to do a DECnet file copy from the source node to the target node, (i.e. the VAX with the printer), and then print the file. Unfortunately, this approach won't work if the target node is heavily loaded. When a DECnet file copy is attempted, the target system attempts to create a subprocess on behalf of the user requesting the copy. This sub-process then communicates with the source node in order to complete the copy. If the target node is heavily loaded, the sub-process creation may not complete and the file copy will fail. This in fact does happen on a regular basis at CRD, depending on the source and target VAXes.

Another approach is to try to cluster several VAXes together. This allows sharing the resources among the VAXes in the cluster. This approach works quite well, except there is an upper limit to the number of VAXes that can belong to a cluster. Since this upper limit is much smaller than the number of VAXes on CRD's local area network, this approach is unsuitable.

The approach taken for this first phase is to provide a single Print Server process on every VAX/VMS system whose printers are to be made accessible by the network print service. Collectively, each Print Server process cooperates with other Print Servers in order to provide the network print service. This approach precludes the overhead of process creation (the Server process is already there), and there is no limit to the number of VAX/VMS systems on the network that run the Server. However, there is another reason for taking this approach. If the architecture of the Server process is made sufficiently general, then it is possible to implement several network services in addition to the print service. This is discussed more in section 3.1.

The overall configuration for the first phase is pictured in Figure 3. At the present time, this configuration repre-

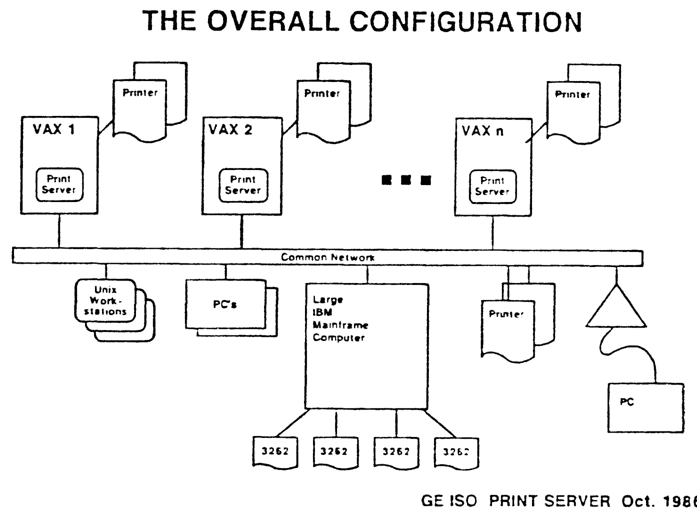



**Figure 3:** Phase I configuration.

sents the extent of the network Print Service developed at CRD. Phases two through four are future projects. Thus the specifics referenced throughout the remainder of this paper refer to issues addressed during the first phase of

development (transparent printing from VAX to VAX).

### 2.5 User-Interfaces

A user-interface is a program used as the bridge between the user making the request and the Server, which provides the service. The Phase I user-interfaces are manifested as three new commands, two of which have DCL ( DEC Command Language ) counterparts. The new commands are depicted in Figure 4. The ***net print*** command is analo-

**SOME COMMANDS**

- net print file-spec [, ... ]
  - Qualifers:
    - /copies
    - /queue
    - /exclude
    - /confirm
    - /delete
    - /header
    - /flag
- net status [job-id]
- new directory [file-spec [, ...]]
  - /exclude

GE ISO PRINT SERVER Oct. 1986

**Figure 4:** Print Server commands.

gous to the DCL *print* command. The syntax is identical, except that fewer command options are provided, due to development time constraints. Those options that are provided are what we feel to be a very useful subset of the DCL *print* options. Similarly, the ***net directory*** command is nearly identical to the DCL *directory* command. The third command, ***net status***, allows a user to monitor the print job once it has been submitted. Upon issuing a valid ***net print*** command, a job identification number (a small integer value) is returned to the user. The user may specify that "job-id" in the ***net status*** command to find out about the specific job. Alternatively, no job-id will give the user information about all jobs in the server's job queue.

## 3.0 SPECIFIC TECHNICAL ISSUES

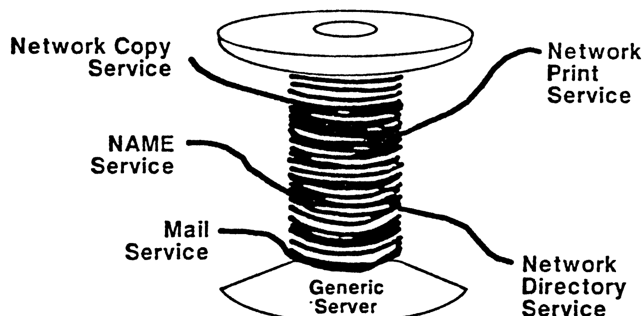
### 3.1 The Server Process

The overall notion for Phase I is to have several VAXes (each using version 4.0 VMS or higher) on the LAN running the Print Server. Each such VAX is referred to as a "Server-node". Each Server-node cooperates with other Server-nodes in order to provide transparent printing between the Server-nodes. Note that there is always just one Print Server process running on a Server-node. Thus several users on a Server-node are serviced by a single Print Server process.

Because the architecture of the Server process is sufficiently general, the Server process can be viewed as a "generic" server, within which several subservices may be defined; the Print Service is just one of the possible subservices, as depicted in Figure 5.

**THE SERVER PROCESS**

On a Given VAX, Multiple Services are Provided by One Server Process to Many User Processes




GE ISO PRINT SERVER Oct. 1986

**Figure 5:** The Server process.

Figure 5 illustrates the Server process as a spool of thread. This represents the fact that several services can be provided by the Server process, and that several users can be serviced by that one process at the same time. The ability to provide multiple services to multiple users via one process is accomplished by setting the Server's process quotas high enough ( e.g. like that of a Symbiont ), and by using multi-threaded code. The multi-threaded code is achieved via non-transparent communication and asynchronous disk and network I/O. From the server's point of view, this means that at any given instant, it may have several different types of "jobs" (i.e. print job, copy job, etc.) in its internal job queue for several different users.

The Server process resides on the Server-node as a network known object. At startup, the Server allocates data structures used for I/O, declares itself a network object, and posts an asynchronous read on its permanent mailbox. Incoming non-transparent connection requests are placed in this mailbox, and the Server awakens from its state of hibernation in order to process these incoming connection requests.

Figure 6 represents a typical service request, in this case, a print request.

In the first part of the picture, the user-interface makes a connection request to the Server process. Note that the user-interface runs in the context of a user's process. Thus the user-interface and the server always run as separate processes. In the second part of the picture, with the connection request accepted by the Server, the user-interface writes a service request to the Server. This service request is just an integer that specifies some sub-

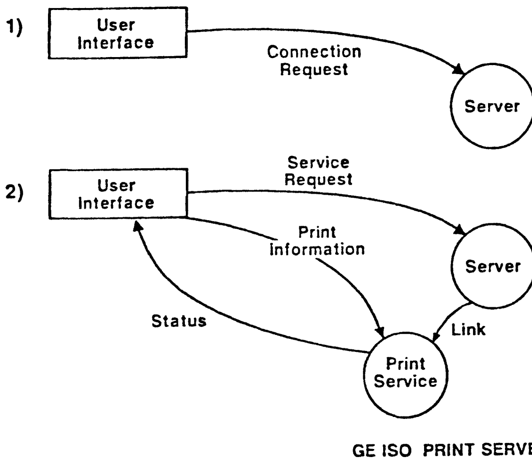



**Figure 6:** An example of a service request.

## OTHER SERVICES IN SUPPORT OF PRINT

- Network Directory Service - Performs Wildcard Expansion and Returns Information Useful for a File Copy.
- Network Copy Service - Performs File Transfer to the Node at Which the Target Queue Resides.
- The Result is a very Layered Design Incorporating Supporting Services and DECnet Routings.

| PRINT SERVICE | | |
|---|---|---|
| Directory Service | Copy Service | Other |
| DECnet Routines | | Other System Calls |
| SYS QIO, SYSASSIGN | | |

GE ISO PRINT SERVER Oct. 1986

**Figure 7:** Services are layered on top of each other.

service (i.e. print, copy, etc.). The Server then invokes the sub-service, passing along the link information from the original connection request. The user-interface and the sub-service then communicate the information necessary to complete the service request. Based on this information, an internal job representation is created, placed in the Server's job queue and is later processed.

### 3.2 Print Service Architecture

There are three logical operations that must be performed as part of a print request. First, a directory search has to be made to resolve all wildcard expansions. Second, the file(s) to be printed must be copied to the target node (i.e. the VAX with the printer). Files are "pulled" to the target node, not pushed (i.e. the target node does the file copying). Finally, the file(s) are printed. Both the directory search and the file copying are independent operations, and in fact are independent sub-services. This idea is depicted in Figure 7, where the Print Service is layered on top of the Directory Service and the Copy Service, as well as the DECnet routines.

The DECnet Routines, written to abstract QIOs, are used for network I/O and are pictured in Figure 8. There are five principal DECnet routines. *Netstartup* must be called once, before any of the other DECnet routines are invoked. This in a sense establishes the "bandwidth" for I/O. The "#Links" parameter specifies how many logical links can be opened for network communications. The parameter "I/O per Link" specifies how many outstanding I/O requests can be placed on a given link at any one time. "Object flag" specifies the object number used in specifying the Server as a network object. *Netopen* is analogous to a file open in C language, except this routine requests a logical link instead of a file pointer. *Netread* and *Netwrite* perform the I/O over the logical link, and *Netclose* closes the logical link.

## DECNET ROUTINES

- A Library of Routines That Makes Network I/O Seem Easy
- Routine Interfaces Not Specific to DECnet
  - Netstartup (#Links, I/O Per Link, Object Flag, AST Address)
  - Netopen (Link, Buffer, Access String, Ast Address, AST Parm, IOSB)
  - Netread (Link, Buffer, Length, AST Address, AST Parm, (OSB)
  - Netwrite (Link, Buffer, Length, AST Address, AST Parm, IOSB)
  - Netclose (Link)
- All Are Layered Over Calls to SYS$QIO, SYS$ASSIGN, SYS$DASSGN.

GE ISO PRINT SERVER Oct. 1986

**Figure 8:** The DECnet routines.

The overall Print Server architecture is depicted in Figure 9.

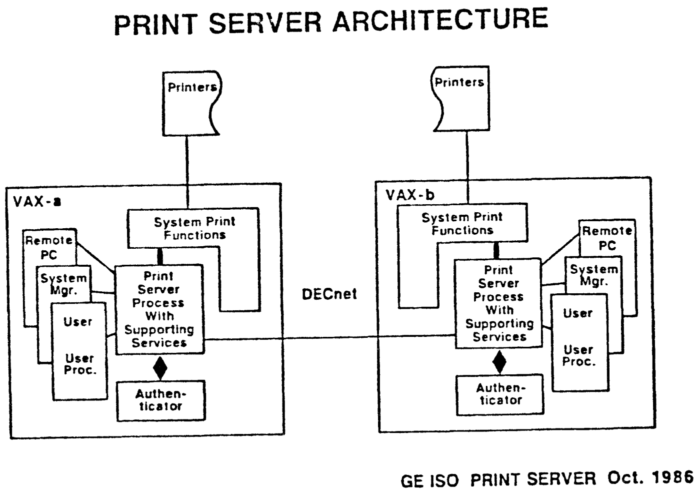



Figure 9: Symmetric Print Server architecture.

This picture is shows that the Print Server architecture is symmetric from one Server-node to the next. The Server-nodes are represented by the boxes such as the ones labeled VAX-a, VAX-b. Within a given Server-node, there are six boxes. The rectangles to the right in the box, labelled "User", "System Mgr", and "Remote PC", represent user-interfaces. The box labelled "System Print Functions" constitutes the job controller and any system routines used in queue manipulation. The "Authenticator" box is used to authenticate print requests to make sure the person requesting the print in fact has access to the files. The final box in the center represents the Server process and all sub-services used in support of the Print Service.

### 3.3 The Authenticator

When the Server starts up, one of the things it does is post an asynchronous read on a permanent mailbox. DECnet places connection request or termination messages in this mailbox. When the Server gets a message in this mailbox, it wakes up and basically checks to see if it is a connection request or some type of termination notification. If it is the latter, the Server deallocates some data structures and closes the logical link. If it is the former, then some service has been requested by someone on a Server-node. Early on in development we found that the identifying information in this mailbox connection request was scant, consisting of only a source node-name and process name of the user making the request. We felt that this wasn't enough to ensure the integrity of file access, and thus developed the Authenticator. A data flow diagram for the Authenticator is given in Figure 10.

To simplify things, assume that everything pictured in Figure 10 is occurring on one Server-node (i.e. the print

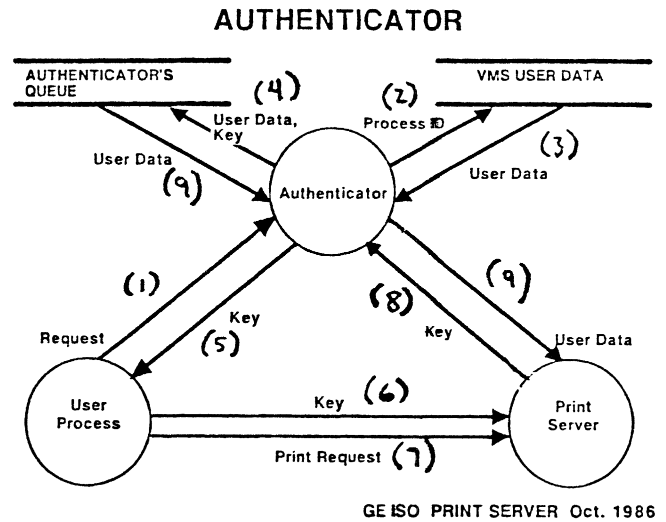



**Figure 10:** Flow diagram for Authenticator usage.

is a local print). Three bubbles are pictured. Two of the bubbles, the "Authenticator" and the "Print Server" are actually part of the same process (i.e. the Server process). The third bubble, "User Process", is a separate process and represents some user who has invoked the print service user-interface via a print command. An explanation of the interaction between these three bubbles follows. The numbers in parentheses indicate numbers next to the arrows in the picture.

When the Server first starts up, it creates a permanent mailbox (distinct from the one used for connection requests) that is used by the Authenticator. An asynchronous read is placed on this mailbox. When the print service user-interface is invoked by the user, one of the things it does is create a temporary mailbox. It then assigns a channel to the Authenticator's mailbox and writes its own temporary mailbox unit number to the Authenticator's mailbox (1). The mere act of writing this information to the Authenticator causes it to wakeup and use the completion I/O status block to get the user's process ID. This process ID is used by the Authenticator in a call to SYS$GETJPI to generate a record of information about the user (2,3). It then generates a random number or "key" and associates that number with the user information (4). A timer is set by the Authenticator so that the user information will become invalid in a very short period of time. Finally, the Authenticator writes this "key" back to the user process (5). The user process (interface) uses this key value in a print request (6,7). The Print Server receives the request and queries the Authenticator about the user by specifying the key (8). If user data is associated with the key, it is sent to the Print Server (9). The Print Server then uses this information to validate that the user who made the print request in fact has access to the files.

The authentication information that the Print Server makes most use of is the username and the uic ( user iden-

tification code ) of the user who submitted the request. In order to determine if the user has access to the file(s), the Print Server temporarily changes its uic to that of the user and tries to access the file(s). Success or failure determines whether or not the user can print the file(s).

This scenario is the one always followed for any print request. The user-interface communicates directly with the local Server only. If files need to be copied, it is handled cooperatively by the Servers. This scenario works well if the file resides locally, because the uic used by the local Server to determine file access is local to the node. If the file(s) to be accessed reside on some other VAX, things don't work as nicely. This is due to the fact that if a user has several accounts on different VAXes, it is likely that the accounts won't have the same username and uic on all of them. If the user is on VAX-A and the file(s) to be accessed are on VAX-B, then the VAX-A Server sends the authentication information it generated locally to VAX-B. The VAX-B server uses the username/uic to access the files. In the event that access fails, the user would have to specify a username-password as part of the file specification. In that case, the server would verify the username-password from the system user authorization file. If correct, the uic from that account is used to access the files.

## 4.0 RESULTS

### 4.1 Performance

When we started the Print Server project one goal we set was that this new network print service should perform no worse than twice as slow as the DCL copy-print (or direct copy to a device). We found that an unloaded server performs quite comparably to its DCL counterparts. Timings were completed by summing the CPU time for each process involved in a print. We believe the timings, though somewhat crude, give a pretty good indication of the Print Server performance. In all, two categories of printing were compared: completely local printing, and remote printing. There were three sizes of files printed: small (3 blocks), medium (92 blocks), and large (249 blocks). The results are described in the sections below. All timings were performed on unloaded systems.

#### *4.1.1 Local Printing*

The timings for local printing for each category of file ( small , medium , and large ) are specified below for both the *NET PRINT* and DCL *PRINT* commands. The timing for *NET PRINT* consisted of summing the CPU-seconds for the Job Controller, Symbiont, NETACP, the *NET PRINT* command, and the Print Server. Timing for *PRINT* consisted of summing the CPU-seconds for the Job Controller, Symbiont, and the *PRINT* command itself. The values listed are approximate CPU-seconds. Though *NETPRINT* was slower than the DCL *PRINT*, it was no worse than twice as slow, and thus satisfied our performance goal.

| PROCESS | SMALL | MEDIUM | LARGE |
|---|---|---|---|
| Job Controller | 0.36 | 1.12 | 2.20 |
| Symbiont | 0.37 | 3.23 | 8.27 |
| NETACP | 0.21 | 1.23 | 2.21 |
| NET PRINT | 0.80 | 0.78 | 0.82 |
| Print Server | 0.82 | 0.64 | 0.87 |
| TOTALS | 2.56 | 7.00 | 14.37 |

**Figure 11:** Timings for local *NET PRINT*, in CPU-seconds.

| PROCESS | SMALL | MEDIUM | LARGE |
|---|---|---|---|
| Job Controller | 0.32 | 0.57 | 0.77 |
| Symbiont | 0.41 | 3.36 | 8.42 |
| PRINT | 0.78 | 0.71 | 0.71 |
| TOTALS | 1.51 | 4.64 | 9.90 |

**Figure 12:** Timings for local DCL *PRINT*, in CPU-seconds.

#### *4.1.2 Remote Printing*

The timings for remote printing for each category of file ( small , medium , and large ) are specified below for both the *NET PRINT* and DCL *PRINT* commands. In each case, the timings consisted of summing the CPU-seconds for all processes involved in the print on both the source and target nodes. In the tables, source node processes are prefixed by "s-", whereas target node processes are prefixed by "t-". For *NET PRINT*, the contributing processes on the source node were the Print Server, NETACP ( Network Ancilliary Control Process ), and the *NET-PRINT* command. On the target node, the contributing processes were the Job Controller, Symbiont, NETACP, and the Print Server. For the DCL *PRINT* command, the contributing processes on the source node were the DCL *COPY* command, and NETACP. On the target node, the contributing processes were FAL ( File Access Listener ), NETACP, the DCL *PRINT* command, the Job Controller, and the Symbiont. The values listed are approximate CPU-seconds. As can be seen, *NETPRINT* was somewhat faster than the DCL *PRINT* for the medium and larger files.

| PROCESS | SMALL | MEDIUM | LARGE |
|---|---|---|---|
| s-Print Server | 0.88 | 0.72 | 0.83 |
| s-NET PRINT | 0.11 | 0.08 | 0.12 |
| s-NETACP | 0.37 | 0.34 | 0.41 |
| t-NETACP | 0.61 | 1.19 | 2.02 |
| t-Job Controller | 0.56 | 1.08 | 1.78 |
| t-Symbiont | 0.39 | 3.43 | 8.41 |
| t-Print Server | 0.69 | 0.69 | 0.83 |
| TOTALS | 3.61 | 7.53 | 14.40 |

**Figure 13:** Timings for remote *NET PRINT*, in CPU-seconds.

| PROCESS | SMALL | MEDIUM | LARGE |
|---|---|---|---|
| s-COPY | 0.02 | 1.57 | 2.85 |
| s-NETACP | 0.02 | 0.05 | 0.10 |
| t-FAL | 0.80 | 3.09 | 3.51 |
| t-NETACP | 0.16 | 0.15 | 1.60 |
| t-PRINT | 0.61 | 0.68 | 0.75 |
| t-Job Controller | 0.41 | 0.54 | 0.66 |
| t-Symbiont | 0.36 | 3.31 | 8.22 |
| TOTALS | 2.38 | 9.39 | 17.70 |

**Figure 14:** Timings for "remote" DCL *PRINT*, in CPU-seconds.

### 4.2 Current Status

Currently, Phase I of the Print Server (transparent printing from VAX to VAX) is in beta-release within CRD. It is our hope to begin Phase II of the Print Server project in the near future. The code for Phase I of the Print Server will be made available at the DECUS '87 Symposium, pending permission by the General Electric Company.

## ACKNOWLEDGEMENTS

The authors wish to thank the following people for their involvement with this project: Carl Chalek, for his work on the user-interfaces; Chris Jolly for helping to finish some modules, and for proofreading this paper; Howard Eskin for his comments throughout; Keith Decker for help in formatting this paper, and Mary Beth Renze, for her help in formatting and proofreading this paper.

# Using the KXT11-CA as an Intelligent Communications Controller

Arthur Hartwig
University of Queensland
St. Lucia, Queensland

**Abstract**

The KXT11-CA is a PDP11 based microcomputer which can act as a slave (or peripheral) processor to a QBUS system. It is useful for offloading processing from a QBUS master CPU such as a micro-VAX or PDP11/83. This paper will describe the development of a stand-alone run time environment for the KXT11, some useful debugging tools and debugging coding techniques. Examples of the techniques will be taken from an HDLC package for the KXT11 developed at the Prentice Computer Centre. Though the paper specifically references the KXT11-CA many of the principles discussed are equally applicable to the use of any general purpose microprocessor in communications.

### *Description of* KXT11-CA *board*

The KXT11-CA is a QBUS peripheral which includes a T-11 CPU (roughly the same processing power as the PDP-11/23), 32kbytes RAM, an asynchronous serial line (for a console terminal) two synchronous/asynchronous serial lines, some counter/timers and 24 bits of parallel i/o. The board includes sockets which can be used to add ROM (or EPROM) or extra RAM. Contained in ROM on the KXT11 board are various self-test routines, ODT and KXT11 load routines.

### *Development environment*

An environment is required for maintenance of source code, conversion from source code to executable code and the loading of the executable code into the KXT11-CA.

*Interactive system with* PDP-11 *assembler*

At the Prentice Computer Centre TOPS-10 provided the program development environment, but other environments such as RSX11, RT11, VMS and UNIX could be used equally effectively.

KXT11 *loader*

Code was developed to load the KXT11 by DMA transfer from both a PDP-11 running under RT-11 and a micro-VAX running under VAX/ELN.

*Interactive debugger*

The DECUS DDT package (e.g., DECUS 11-SP-6) can be configured to operate in a stand alone mode suitable for execution on the KXT11. This debugger is far superior to ODT in that it offers symbolic debugging, symbolic instruction type-out and type-in.

KXT11 *Dump Analyser*

The dump analyser (DDT11) which ran on the DEC-10 allowed a KXT11 memory dump to be analysed to determine the cause of failure of the code. This proved to be very useful in helping to track down a variety of bugs which caused internal consistency errors within the KXT11 code. Similar facilities would be available through a suitably modified version of the DECUS DDT package.

*Protocol analyser*

A protocol analyser is a very useful tool for observing what a protocol implementation does in response to a variety of events and stimuli. Often such devices can themselves engage in conversation with implementations and a log of such conversations is helpful in development. The symbolic decoding provided by many such devices is also useful in understanding protocol exchanges.

### *Replacing the operating system*

The KXT11 stand alone application runs without the support of an underlying standard operating system so a variety of services should be provided to replace standard operating system functions such as:

*Machine error handling (memory dumps)*

It was decided early on that in the case of an unexpected condition arising the KXT11 code should be dumped into a file for later analysis. This is the familiar "crash dump" action included in many operating systems. However for it to be possible to create a crash dump the machine must "fail" in a well defined way. Therefore interrupt vectors for machine errors (such as illegal instruction) must all perform a well defined action. A simple but effective approach is to execute an EMT or TRAP or IOT instruction on such a condition. Then code a handler for the EMT (or TRAP or IOT) interrupt to save all registers on the stack and save the stack pointer away.

However if the DDT package is being used certain vectors must be preserved to the values set by DDT. Conditional assembly can easily cope with this.

MACRO-11 code to initialise the interrupt vectors might be:

```
        CLR   R0              ;Start at vector at 0
        MOV   #CRSINT,R1      ;Load error interrupt
                              ;handler
        MOV   #340,R2         ;Disable further
                              ;interrupts
.IF DF FT.DDT
        MOV   NXMVEC,-(SP)    ;Save vectors set by
                              ;DDT
```

```
        MOV   NXMVEC+2,-(SP)     ;
        MOV   ILSVEC,-(SP)       ;
        MOV   ILSVEC+2,-(SP)     ;
        MOV   BPTVEC,-(SP)       ;
        MOV   BPTVEC+2,-(SP)     ;
.ENDC;.IF DF FT.DDT
14$:    MOV   R1,(R0)+           ;Set interrupt handler
        MOV   R2,(R0)+           ;and its priority
        INC   R2                 ;Adjust priority
        BIC   #20,R2             ;But don't let T bit get
                                 ;set
        CMP   R0,#400            ;Gone high enough?
        BLT   14$                ;Branch if no
.IF DF FT.DDT
        ;Restore DDT's interrupt vectors
.ENDC;.IF DF FT.DDT
        MOV   #EMTINT,EMTVEC     ;Initialise EMT vector
```

If FT.DDT is undefined the vectors at 0, 4, 10 etc will specify a processor status word for the interrupt handler of 340, 341, 342 etc. An interrupt through one of these vectors will execute at CRSINT where in the following code segment the EMT will force the newly loaded processor status word of 340 (or 341 or 342 etc) onto the stack followed by the PC (CRSINT+2). The EMT interrupt handler then corrects the PC on the top of the stack to point to the executed EMT and saves all registers on the stack and then saves the stack pointer so that examination of the dump will show the active stack frame at the time of the crash.

```
CRSINT: EMT                      ;Easy way to deal with
                                 ;unexpected interrupts
EMTINT: SUB   #2,@SP             ;Point to offending EMT
        MOV   R5,-(SP)           ;Save the registers
                                 ;on the stack
        MOV   R4,-(SP)           ;
        .     . .                ;
        .     . .
        MOV   SP,#0              ;Save stack pointer
                                 ;Dump KXT11 memory
                                 ;to QBUS host.
```

*Multiprocessing*

Considering a KXT11 application to be a number of independent processes is a useful structuring tool. A protocol implementation can often be described as a receive process, a transmit process and a timer process. The transmit process creates messages and starts the serial interface chip transmission. The transmit interrupt handler manages the transmit done interrupts from the serial interface chip, supplying a new character on each interrupt until the whole message has been transmitted then it disables interrupts and schedules the transmit process which will look to see if a new message can be transmitted. Similarly a receive process can enable receive interrupts and allow the receive interrupt handler to assemble a message. When a message has been assembled by the receive interrupt handler it schedules execution of the receive process which then examines the newly received message and decides what action to take based on the protocol specification. The transmit and receive process can communicate through shared variables and flags. For example a newly received message may need to be acknowledged, so its sequence number might be stored, a flag set for the transmit process saying an acknowledgment must be sent and the transmit process scheduled. The timer process is called periodically by the clock interrupt code (usually the clock interrupt will cause scheduling of the timer process rather than calling it directly during interrupt processing) to force retransmissions on time-out or force the transmission of a particular message to probe the other end of the line to determine its state.

The process scheduling can be very simple. Each process may be allowed to use all machine registers and coded simply as a subroutine. Then the scheduler needs only to decide what subroutine to call. A bit mask will easily allow up to 16 different processes to be scheduled. Scheduling can easily be either strictly priority based, or round-robin. (This scheme doesn't easily support pre-emptive scheduling, but is very low overhead.)

A very simple scheduler might be as follows:

```
LOOP:   MOV   REQMSK,R0          ;Anything to do?
        BNE   10$                ;Branch if yes
        INC   NULCNT             ;Else count times
                                 ;through idle loop
        BR    LOOP               ;And look again

10$:    MOV   #REQTAB,R1         ;Load addr of
                                 ;table of despatch
                                 ;addresses
        MOV   #1,R2              ;Mask for requests
20$:    ASR   R0                 ;Shift out a possible bit
        BCC   40$                ;Branch if not requested
        BIC   R2,REQMSK          ;Clear the request
        MOV   @R1,#0             ;Remember last
                                 ;despatched
        JSR   PC,@(R1)+          ;Go there - shorter than
                                 ;JSR PC,@(R1)
        CMP   SP,#STACK          ;Check he didn't smash
                                 ;stack
        BEQ   30$                ;Branch if OK
        EMT                      ;Else doomers

30$:    CMP   0,#CRSINT          ;Smashed 0?
        BEQ   LOOP               ;Branch if ok, start
                                 ;again
        EMT                      ;Else its doomers

40$:    ASL   R2                 ;Shift mask along one
                                 ;place
        TST   (R1)+              ;Advance to next word
                                 ;in despatch table
        BNE   10$                ;See if had request for
                                 ;next entry
        BR    LOOP               ;At end, see if another
                                 ;to schedule

.MACRO  DESPAT
        X     ARC                ;Channel A receiver
        X     BRC                ;Channel B receiver
        X     AXM                ;Channel A transmitter
        X     BXM                ;Channel B transmitter
        X     CLK                ;Clock
        X     HST                ;Host interface
.ENDM
        $$    = 1                ;Highest priority
.MACRO  X     A
        RQ$'A = $$               ;Set request flag
        $$    = $$*2             ;Step to next flag bit
        .WORD REQ'A              ;Set despatch address
                                 ;for process
```

```
.ENDM  X

        DESPAT
;Generate table
        .WORD 0                 ;Mark its end
```

Note the use of the EMT instruction to cause a KXT11 crash if location zero no longer contains CRSINT when the scheduled process suspends itself (returns). Location zero might be easily inadvertently modified if an index register or buffer pointer becomes zero unexpectedly, either due to a coding or logic error.

Deliberate corruption of location 0 is a convenient way to generate a crash dump for the purpose of examining a running system to try to determine why it is behaving the way it is.

*CPU use statistics*

The scheduler described above can very easily test for something to schedule. The mask is zero if there is nothing to schedule. In this case a null time counter can be incremented. On a clock interrupt the value of the null counter gives a measure of idle time during the last clock tick. An averaged null time value can be easily computed in two instructions:

```
ADD   NULCNT,NULAVG     ;Up average of null time
ASR   NULAVG            ;
MOV   NULCNT,NULTIK     ;Record null count over
                        ;last tick
CLR   NULCNT            ;Reset idle counter
```

In a steady state system NULAVG converges fairly rapidly to the long term null time value.

The statistics from the idle loop counter can be used to gain some idea of the performance of the system. When there are no processes wanting to run the idle loop is executed about 1854 times per clock tick. With a single line operating at full speed 19.2kbps full duplex merely transmitting and receiving characters this value is reduced by 30%, with two lines at 19.2kbps *and* HDLC frame interpretation and 120 byte information frames this value is reduced by 65%.

*Buffer management*

Communications protocols commonly allow for variable length messages to be exchanged between communicating systems. In a system with "sufficient" memory it would be possible to allocate a number of fixed length buffers each capable of holding the longest message. Such buffers could be very quickly added to or removed from a buffer free list. However the KXT11-CA board doesn't have "sufficient" memory to allow the adoption of this technique. Therefore variable length buffers are required to make optimum use of available memory. However this can require fairly complex buffer management routines (particularly to avoid excessive fragmentation) and cause problems if buffers need to be allocated during interrupt processing. (To maximise throughput interrupt routines must be as short as possible. Spending unpredictable lengths of time in buffer allocation during interrupt processing is a very undesirable practice.)

In protocols such as SDLC, HDLC and Bisync the end of a message is marked by a particular bit sequence and so cannot be known before hand by the receiver. In contrast, DDCMP includes a byte-count in a message header so there is some scope (but not really very much due to the timing constraints) for pre-allocating a buffer to hold all the message.

A useful compromise between the conflicting requirements of being able to process variable length messages, memory efficiency and speed of buffer allocation and deallocation is to provide a pool of fixed length buffer fragments and chain these together (where necessary) to provide a variable length buffer composed of a linked list of fragments. The chain can use the first word of each fragment for the linkage with a link of zero terminating the chain. The list of free fragments can be maintained by pointers to the first and last fragments on the list. Then the operations of removing a buffer fragment from the free list and adding a buffer fragment to the free list are both short and simple operations.

Code to add a fragment (whose address is in R0) to the free list might be:

```
CLR   @R0           ;Clear link - this will be
                    ;end of chain
MOV   R0,@LSTFRE    ;Append this fragment
                    ;to end of chain
MOV   R0,LSTFRE     ;Point to new end of
                    ;chain
```

Code to remove a fragment from the free list and place its address in R0 might be:

```
MOV   FIRFRE,R0     ;Load pointer to first
                    ;free fragment
MOV   @R0,FIRFRE    ;Make next on list the
                    ;first free
CLR   @R0           ;This is not yet linked
                    ;to another
```

Of course if these code fragments are to be called by interrupt handlers then the CPU must execute these code segments at level 7 and the CPU bus request level must be saved and restored around these code segments.

If the buffer fragment length is an integral power of two (e.g., 32 decimal) and the fragments are all allocated so that they begin on an address boundary which is an integral multiple of the same power of two then the low order *n* bits of a buffer fragment address are always 0 and this property may be used to easily follow the fragment chains. Code for following the chains might be:

```
      MOVB (R0)+,(R1)+        ;Copy a byte from
                              ;buffer
      BIT   #BUFSIZ-1,R0      ;Reached end of
                              ;fragment?
      BNE   10$               ;Branch if no
      MOV   -BUFSIZ(R0),R0    ;Yes, start at next
                              ;fragment
      TST   (R0)+             ;Skip over its link word
10$:  ...                     ; ...
```

It is also useful to maintain a count of fragments currently free and the maximum number of free fragments. Such counters enable checks to be made that demand for buffer fragments doesn't exceed supply — that is the free count doesn't drop to zero.

Maintaining a "low water mark" of the free buffer count enables evaluation of the buffering strategy. Buffer management could be a problem to a KXT11 application because the host can supply messages to send far more quickly than the KXT11 can send them. Therefore indiscriminate copying to KXT11 memory of messages for transmission could cause a lockup situation due to buffer exhaustion. If there are no buffers available there can be no messages received so no messages which have been transmitted can be acknowledged so none of the buffers used for messages which have been transmitted can be returned to the free pool so the no buffer condition continues.

***Coding to ease debugging***

It is very easy to succumb to the temptation of optimism and believe that an implementation will be correct on initial coding. This rarely happens and even if it did problems can arise due to different implementors making different interpretations of the protocol specifications. Therefore it is useful to build into the code mechanisms which will leave tracks to help assist the discovery of why the implementation behaves as it does.

*Asynchronous entry to debugger*

An asynchronous entry into the debugger is a very useful tool for suspending the KXT11 application and allowing variables to be examined or modified or the execution flow to be changed. Debugger breakpoints are of no use if execution does not reach the breakpoint. For example, there may be no messages sent in response to incoming frames. If the problem is that the transmit process is not scheduled then having a breakpoint in the transmit process will not assist the programmer to discover why no frames are being sent.

Since the debugger is present there must be a console terminal. A receive interrupt handler for the console terminal might include a check for a special character (eg control-C) and if it is seen execute a BPT (BreakPoint Trap) instruction which will cause execution to enter DDT.

The asynchronous entry to the debugger can also be used to deliberately corrupt location zero to generate a crash dump for debugging purposes; for example, to see why communication is not proceeding.

*Counters and statistics*

Especially in the early stages of development it has been found to be useful to include instructions to maintain counters of the number of times particular events have occurred or particular code sequences have been executed. Event counters may be incremented from a number of places within the code whereas "code sequence counters" are incremented only once each time the code sequence is executed and so can indicate whether a particular code sequence has been executed or not.

A example may help to make the distinction clear. A counter for receiver overruns may be incremented from a number of places in the code: on hardware receiver overrun, on receiving end of frame when there isn't a buffer available to allocate for reception of a new frame, or even partway through receiving a frame if a fragment is full and another one is not ready for storing the next character. The receiver overrun counter may indicate there is a problem, the other counters suggest the particular area that might require a new strategy.

Counters of protocol events can help to identify problems caused by the parties attempting to communicate using incompatible configurations (e.g., in HDLC one is a DTE and the other also a DTE or incompatible maximum frame lengths). There is a story (believed true) of two laboratories in different parts of the country implementing X.25 and agreeing to use a default maximum packet length of 255 bytes. However was this 255 bytes of data or did the 255 include the X.25 header? The two laboratories adopted different interpretations. The two networks based in the two laboratories mostly communicated but occasionally the machines in one of the laboratories crashed in turn. Eventually it was found the cause was the other laboratory sending a packet with 255 bytes of data. This packet overflowed allocated buffers and corrupted buffer pointers but the corruption didn't have any impact until after the packet had been forwarded; then the machine forwarding the 255 byte packet crashed. However the next machine to receive the 255 byte packet again managed to successfully forward the packet before crashing.

It is good practice to assume the party with whom you want to communicate will use a strange dialect of the protocol, and then to write code to guard against it. Of course in the case just mentioned the particular implementation would not have been very robust if two abutting two hundred byte frames were transmitted and the intervening flag was corrupted by line noise.

Code sequence counters are helpful also in producing correctly functioning interface drivers. Documentation accompanying interface chips often does not give much detail on how the chip should be driven, especially in coping with error situations. Documentation is often vague concerning even straightforward things. For example is the end of frame status returned before, during or after the interface chip returns FCS bytes (if indeed the chip returns the FCS bytes at all)?

It is likely that "permanent" counters will be required for protocol events such as CRC errors. Other counters may be required only during the debugging phase. Code to maintain such counters may be produced by a MACRO then the definition of the macro changed when the counters are no longer required. The following MACRO maintains a counter or maintains the value of a variable the last time the MACRO code was executed depending on whether or not a MACRO parameter was specified.

```
.MACRO      TRACE$      ARG
.IF B ARG
            INC         #0
.IFF
            MOV         ARG,#0
.ENDC
.ENDM
```

The code sequence generated by the MACRO maintains a counter in only 4 bytes or maintains a value in no more than 6 bytes; alternate means would be slower in execution and require more space. However this technique is of no use if code and data are mapped separately or code is stored in read-only memory. In this case the MACRO should be modified to cause updates of a block of read-write memory and a symbol defined to remember how much such memory should be allocated for the block.

*Clock interrupt check for looping*

The responsiveness of the KXT software depends on the individual processes not taking "too long" (half a second?). The clock interrupt handler can easily enforce this by incrementing a counter every tick and requesting the running of the timer process which clears the same counter. If the clock interrupt handler increments this counter to too high a value it executes an EMT forcing a dump. This is an effective but brutal way of identifying the "looping" process! The main scheduler keeps the start address of the last scheduled process and the stack section preserved in the dump will point to the erroneously looping code.

*Frame gobbler*

A useful check on the protocol implementation is to simulate line errors by randomly discarding received frames. By "mashing" together a few changing numbers using the XOR instruction and checking that an appropriate combination of bits is zero a "random" frame discard mechanism can be devised. Changing numbers can be taken from the count of clock ticks since loading, count of frames received, count of idle loops since last clock tick and count of frames discarded.

*Use of hardware features for protection*

Where they are available hardware features should be used to help identify coding errors. For example if the microprocessor supports memory management it might be possible to map the code to a read-only section so that a write into the code will cause a memory management interrupt. The initialisation code might choose to initialise all the vectors then map the low area of memory (including location 0) in a read-only segment; again resulting in a memory management interrupt if a write occurs.

The T-11 CPU chip used in the KXT11-CA does not have memory management facilities but other PDP-11 CPUs (e.g. the J-11) do.

*Buffer fragment checking*

If the address of buffer fragments is always an exact multiple of a power of 2 the fragment address passed to the "free a fragment" routine can be checked for "plausibility"; for example:

```
BIT     #BUFSIZ-1,RØ     ;Address of fragment
                         ;plausible?
BEQ     +4               ;Branch if yes
EMT                      ;No, crash
```

Similar code can check a fragment address on removal of a fragment from the free list and so act as a cautious guard against wild code scribbling over the free list.

It is also easy to return a fragment to the free list when the fragment is already on the free list — simply by forgetting to clear a stale pointer. Such problems usually result in the free list coming to an end while the count of free fragments is non-zero. A useful technique for tracking down such problems is to allocate an extension fragment immediately following the active part of the fragment. The extension contains a flag which is set when the fragment is added to the free list and cleared on removal from the free list. Registers and a stack section are stored in the extension fragment following the flag. When a fragment is added to the free list, if the flag is set an EMT is executed and the dump examined to determine the two code segments which attempted to add the same fragment to the free list. The active stack section when the EMT was executed will identify the code which erroneously freed the fragment and the stack section stored in the extension fragment will identify the code which first put the fragment on the free list.

This technique proved to be helpful in one case of double buffer de-allocation. The routine called to start the transmission of a new frame firstly returns to the free list the buffer containing frames that have been acknowledged. The receiver process cannot de-allocate these buffers because the buffers may be in use for re-transmission of frames which hadn't been acknowledged within the time-out interval. Returning the buffers to the free-list disturbs the links between buffer fragments since the fragment most recently added to the free chain always has a zero link. Therefore if a chain of fragments is returned to the free list while the transmitter is sending from that chain the situation may arise that the interrupt handler will come to the end of a fragment knowing there are more fragments to send from but be unable to find those fragments since the links are no longer intact.

When the transmitter is looking for a new frame to send it is known that frames are not being transmitted so it is safe to disturb the link pointers. If the transmitter underruns an abort is sent. The serial interface chip generates a transmit buffer empty interrupt when the abort sequence completes (if interrupts are still enabled).

Now if the following sequence of events occurs trouble is encountered:

1. A fragment being transmitted is exhausted so the CPU BR level is lowered to allow further serial line interrupts while the next fragment is prepared for transmission.
2. Other serial line interrupts are processed lengthening the time required to prepare the next fragment for transmission and a transmit underrun occurs.
3. An abort is transmitted and the transmitter process queued.
4. The transmitter process looks for something to do and frees frames from the sent and acknowledged list.
5. Before the stale pointers are updated a transmit buffer empty interrupt occurs on completion of transmission of the abort and since there is no buffer to send the interrupt handler tries to get a new one and calls the top level "get a new fragment to send" routine which then looks for sent and acknowledged frames and then proceeds to free frames which have already been freed.

This particular problem could have been solved in a number of ways but the way chosen was to disable transmit buffer empty interrupts just before sending the abort. There was a window of about three instructions during which this sequence of events could happen — yet another manifestation of Murphy's Law: "If it can go wrong it will go wrong."

*Frame trace*

In the absence of a suitable protocol analyser the receive and transmit processes can be coded to write to the console a textual description of frames and a decoding of protocol variables specified in frames (for example, the send and receive sequence numbers). This tends to place quite a load on the CPU but is useful for at least checking operation at lower speeds. If the implementation doesn't work at low speed it is unlikely to work at higher speed unless the problem was the specification of too short a timer for the actual speed of operation.

*Hand check interrupt routines often*

Interrupt handlers must be very careful in how they modify the environment. It is especially important to check that the interrupt handler saves and restores all registers used. Failure to obey this rule could well result in intermittent errors which are very difficult to find. All paths should be carefully checked. This is another reason why interrupt routines should be as short as possible.

***Coding for performance***

As well as having a robust protocol implementation with adequate debugging facilities included it is desirable to have an implementation with good performance.

*Short interrupt routines — check longest path!*

For high throughput in communications applications it is important to make interrupt routines run for as short a time as possible while interrupts are disabled. The HDLC software driving both lines at 19.2kbps full duplex takes just under 60% of the CPU in character interrupt handling while HDLC frame processing uses 4% to 5% of the CPU. It would appear initially that there is scope for increasing line speeds by about 50% yet this is not possible in practice without code modifications. This is because the length of the longest code sequence with interrupts disabled and the device characteristics together determine the maximum throughput.

For example the KXT11-CA uses the NEC 7201 dual line USART. This device has a three character receive buffer, a single character transmit buffer and receive interrupts have a higher priority than transmit interrupts. With both lines operating at 19.2 kbps full capacity the transmitters require a new character about every 400 micro-seconds and new characters arrive for the receivers about every 400 micro-seconds. The longest receive interrupt processing happens at end of frame and takes at least 42 instructions to calculate the length of the frame and add the frame to the receive list. In the worst case the transmit interrupt handler takes about 30 instructions to supply a new character. If all worst case conditions coincide then transmit underrun occurs. (In fact this was observed about three times an hour under test conditions of 120 byte frames — perhaps too infrequent to be concerned about. Nonetheless recoding the transmit interrupt handler to reduce the delay in supplying a character to the transmitter to no more than 6 instructions removed the occurrence of transmit underrun.)

It is strange the chip designers chose to give most buffering *and* highest interrupt priority to the receivers. Assigning the transmitters higher priority would have relaxed timing constraints for full duplex operation. It is also a pity the KXT11-CA designers did not provide hardware to vector the 8 different 7201 interrupt conditions directly to the appropriate handler. The shortest 7201 interrupt routines which are also the ones executed at least 9 out of 10 times take 13 instructions of which 6 are consumed in despatching to the appropriate condition handler.

Provided care is taken an interrupt routine can itself be made interruptible. This is particularly useful if the interrupt routine must perform some "long" operation such as allocating a new buffer fragment. Take care though that the interrupt condition no longer applies when the CPU priority is lowered (that is the character has been read from the receive buffer or a new character has been written to the transmit buffer).

For example a receive interrupt routine might perform the following actions:

1. Store character into buffer.
2. Update buffer pointer and counter.
3. If still room in buffer dismiss interrupt.
4. Else get another buffer fragment and chain it to end of current list.
5. Dismiss interrupt.

This handler has further interrupts disabled for as long as it takes to store the character and ensure there is buffer space for the next character. Hence the maximum throughput is limited not by how long it takes to execute the half dozen or so instructions it takes to put a character into the buffer but by the hundred or so instructions it takes to get a new buffer and update the lists.

An alternative scheme making use of a primary buffer into which characters are initially stored and a secondary buffer which is allocated before the receiver is started might look like this:

1. Store character in primary buffer.
2. Update primary buffer pointer and counter.
3. If still room in primary buffer dismiss interrupt.
4. Else set primary buffer pointer and counter to secondary buffer (which becomes the new primary buffer) and chain it into list.
5. Enable further interrupts (lower CPU bus request level etc).
6. Allocate a new secondary buffer.
7. Dismiss interrupt.

This scheme ensures that there is a buffer available to store a received character while there is a possibility of a receive interrupt.

*Use of* DMA

Injudicious use of the DMA controller can also significantly lengthen the elapsed time spent in interrupt routines as the processor and DMA controller compete for KXT11 bus cycles. The KXT-HDLC software sets up the DMA controller to transfer a whole frame to the QBUS host using the chain reload feature of the DMA controller, and tests for completion by polling. In the first version of this code which attempted to do the correct thing by specifying DMA interleave on the KXT11 bus with the CPU, operation of the serial lines at speeds greater than 9600 bps resulted in frequent transmit underruns. A logic analyzer revealed that the DMA controller was faithfully alternating KXT11 bus cycles with the CPU thus significantly lengthening the interrupt servicing time. It was necessary to inhibit interrupts during DMA and inhibit DMA during interrupts. The interrupt handler could turn off DMA on entry and turn on DMA again on exit but this was not an optimum solution since bus interleaving would still take place during interrupt despatch, it would be awkward to do it for every interrupt cause and it would not be necessary on all interrupts anyway.

However the DMA controller doesn't perform DMA transfers while bit 0 of the master mode register is clear. So the controller is initialised in KXT11 bus interleave mode without bit 0 set. In this state DMA transfers can be started or suspended by toggling bit 0. Then the following code is executed:

```
        MTPS  #340            ;Disable interrupts
        BIS   #1,DMA.MM       ;Allow DMA while
                              ;interrupts are suspended
        BIC   #1,DMA.MM       ;Suspend DMA
        MTPS  #0              ;Allow interrupts now
                              ;
                              ;Test for DMA
                              ;completion. If not
                              ;complete branch back
                              ;to MTPS #340.
```

This code sequence effectively allows DMA activity in 4 word bursts with no contention for the KXT11 bus between the DMA controller and interrupt service routines. The addition of a few NOP instructions between the bit set and bit clear instructions will allow additional DMA cycles in the burst.

Use of DMA on the serial line does not cause problems due to the relative infrequency of the DMA requests.

*Use of addressing modes*

Absolute memory addressing (turned on by the MACRO-11 directive .ENABL AMA) is faster than PC relative addressing. This can provide a small performance gain especially in frequently executed pieces of code (e.g., interrupt handlers).

# Development of a Packet Switch Exchange

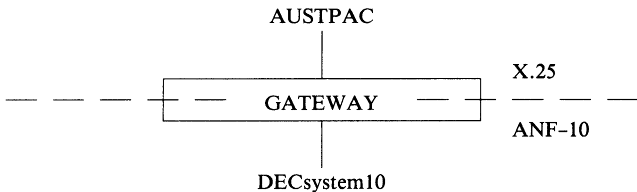
**Arthur Hartwig and Danny Smith**
**University of Queensland**
**St. Lucia, Queensland**

**Abstract**

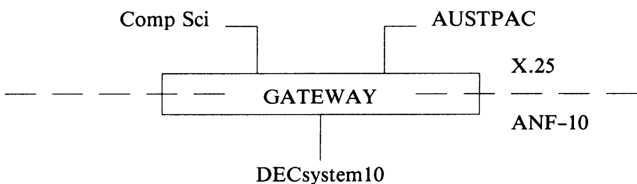
With increasing availability of X.25 implementations a need has arisen to interconnect campus systems using X.25 and also to provide access to public X.25 networks from a number of campus systems. This paper describes an X.25 packet switch which runs on micro-VAX CPUs under the VAX/ELN run-time system. The switch has extensive tailoring facilities and uses KXT11s to relieve the micro-VAX of HDLC processing overhead. The KXT11 HDLC software will be briefly described as well as the switch and its facilities.

### The Need for the switch — Some history

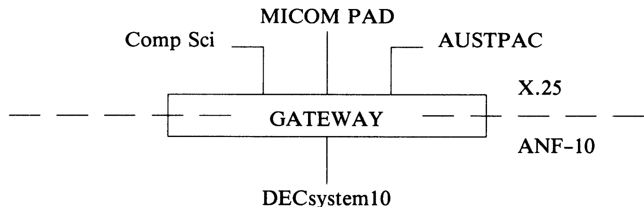
The Prentice Computer Centre's first X.25 connection was a gateway between AUSTPAC, the Australian public X.25 network and the University's DECsystem10 based ANF-10 network. The gateway was originally developed at the University of York (United Kingdom) and provided facilities for terminal access in both directions and process to process communication in both directions. File and electronic mail transfer facilities were built on the process to process communication facility and the so-called "Coloured Book" protocols (widely used in the academic and research community in the United Kingdom) were implemented on the DECsystem10 and the gateway. These protocols provide for process to process communication, file transfer, electronic mail and terminal access (both X.29 and TS29, an X.29 like protocol which copes much better with establishing connections through gateways than does X.29). The gateway also includes per user accounting facilities for both interactive and process to process calls.




In 1985 a UNIX based coloured book implementation was installed in the Computer Science department and a simple X.25 switching facility within the gateway was enabled. The switching facility allowed X.25 access from the Computer Science machine to the public network and vice versa (but bypassing the X.25 ↔ ANF-10 conversion) and DECsystem10 access to both the public network and the Computer Science machine (utilising the X.25 ↔ ANF-10 conversion function of the gateway).

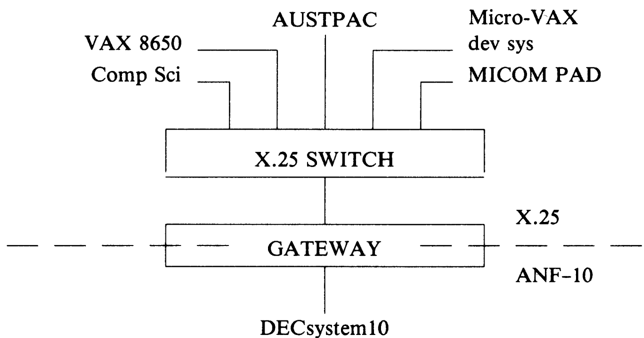



In late 1985 a MICOM X.25 PAD was installed in the campus MICOM 600 circuit switch to allow one of our large customers access to an IBM mainframe from the public data network. The gateway switching function was configured to allow X.25 access from both the public network and the Computer Science machine to the MICOM PAD. The MICOM PAD was configured to prevent it making X.25 calls since such calls could not be properly accounted for.




Now the single gateway installed to support protocol conversion is potentially also performing a significant switching function and handling considerably higher character interrupt loads than it was ever intended to do.

It was expected that increasing use would be made of the X.25 connection to the public network and the bandwidth required on that link would also increase. It was also expected that an increasing number of machines would require access to the public network so some means of offloading character interrupt processing from the gateway was required. At the same time the life of the DECsystem10 and the ANF-110 network was drawing to an end so it was decided to move the switching function to a dedicated machine and allow the gateway to return to providing just the ANF-10/X.25 protocol conversion function.




It was decided in the absence of alternative DEC products to use the KXT11-CA as an intelligent communication processor and write an HDLC implementation for it. (At the time the KMV11 HDLC software was not available and even if it was the KXT11 provided a much lower "per line" cost as well as the possibility of supporting many more physical lines on a fully configured switch than would be possible using KMV11s. This is because the KXT11 provides two lines on a single quad width board for only a slight increase in cost over the KMV11 which provides a single line on a quad width board.)

The micro-VAX running under VAX/ELN was chosen for the switch — mainly because of the good development tools available but also because the VAX CPU offered a much larger address space than the PDP11 (the machine used for the original ANF-10 gateway). The size of the address space was an important consideration because the PDP11 gateway was becoming harder and harder to maintain because adding new functionality increasingly required looking for ways of fitting new code into an address space that was already nearly full.

## Overview of X.25

X.25 is an international standard (CCITT) protocol for connecting a computer (or packet mode terminal) to a public packet switch network. In the literature a packet mode terminal is often called a DTE (for Data Terminal Equipment) and the network DCE (for Data Circuit terminating Equipment). The protocol can be considered to consist of two parts: a frame level protocol (HDLC for High-level Data Link Control) for ensuring the correct transfer of information between the network and the packet mode terminal and a packet level protocol for setting up, maintaining and closing down a number of circuits between packet mode terminals connected to the network. The term X.25 is sometimes (incorrectly) used to refer to the packet level protocol.

### Frame level protocol handling

KXT11-CA boards (with HDLC software written at the Prentice Computer Centre) are used to implement the frame level protocol of X.25. The KXT11 HDLC code transfers correctly received information frames to micro-VAX memory and transfers buffers from micro-VAX memory to KXT11 internal memory for transmission as information frames. All HDLC processing is done within the KXT11. Additionally the KXT11 HDLC software informs the QBUS host of HDLC events such as "HDLC started", "HDLC failed" and "HDLC reset".

The KXT11 HDLC code resides in a file in a named volume on the micro-VAX. The code is loaded into the KXT11 by a two stage process. The file is read and a KXT11 memory image created in micro-VAX memory, then the KXT11 memory image is DMA transferred into the KXT11 memory.

If the KXT11 HDLC code detects an unrecoverable or unexpected error condition it sets a status bit and interrupts the micro-VAX to request a memory dump. The micro-VAX writes a dump file which can be subsequently analysed to determine the cause of the crash.

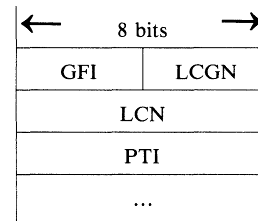
HDLC has a number of configurable parameters: window size (maximum number of sent but unacknowledged information frames), retransmission timer, maximum frame size and maximum number of times a frame may be retransmitted. In addition a HDLC implementation behaves slightly differently depending on whether it plays the role of DTE (Data Terminal Equipment that is a packet mode terminal) or DCE (Data Circuit terminating Equipment or network). All HDLC parameters and DTE/DCE behaviour can be individually configured for each KXT11 line by command from the micro-VAX. Though HDLC has two frame structures; one supporting a maximum window size of seven and the other a maximum window size of 127 only the variant supporting the window size up to seven is currently implemented.

The current version of the KXT11 HDLC software supports two lines operating at 19.2kbps with some processor capacity to spare.

### Packet level protocol

X.25 provides Permanent Virtual Circuits (PVCs) and Switched Virtual Circuits (SCVs or virtual calls). Both classes of virtual circuits provide bidirectional full duplex i/o channels. SVCs differ from PVCs in that SVCs are created as requested on the transmission of a CALL REQUEST packet from the packet mode terminal to the network. PVCs are created when the network is configured and are permanent in that they are always setup.

Each valid X.25 packet consists of a three byte header possibly followed by additional data. The form of the packet header is illustrated in the following diagram.




The four bit GFI (General Format Identifier) is a field which identifies whether the packet structure follows the rules for the modulo-seven sequencing form of the packet protocol or the modulo-127 sequencing form. The "Q" (Qualifier) and "D" (Delivery confirmation) bits also reside in the GFI field but they are of no significance to the switch.

The four bit LCGN (Logical Channel Group Number) field and the eight bit LCN (Logical Channel Number) field together form a 12 bit logical channel identifier which is used by both the packet mode terminal and the network to identify a particular virtual circuit. This number is purely local to the interface between the packet mode terminal and the network. (This is different from DECnet in which each logical link is identified by a separate link address for each end of the link. In DECnet the two link addresses will normally be different though they may be the same.)

The eight bit PTI (Packet Type Identifier) is encoded in a variety of ways but in all cases examination of this field is sufficient to determine the type of the packet.

PACKET LEVEL STARTUP

The packet level protocol is initialised by the network and the packet mode terminal exchanging RESTART REQUEST (or RESTART INDICATION) and RESTART CONFIRMATION packets. (The RESTART REQUEST and RESTART INDICATION packets have the same format. RESTART REQUEST is the name given to the packet sent from packet mode terminal to network and RESTART INDICATION is the name given to the packet sent from network to packet mode terminal.) In the event of a collision (e.g., the packet mode terminal receives a RESTART INDICATION packet in response to a RESTART REQUEST packet) the initialisation procedure is considered to have completed.

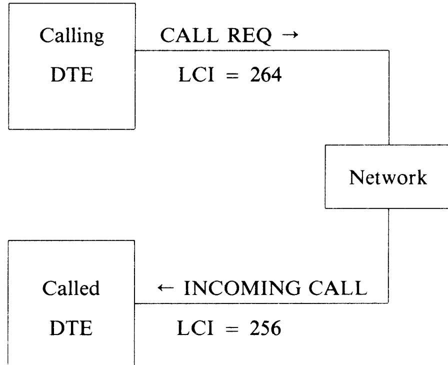
ESTABLISHING CALLS

A packet mode terminal will subscribe to a certain number of logical channels of each type and this will determine the

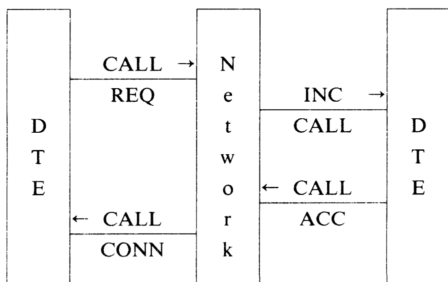
range of logical channel numbers that it may use. The packet mode terminal when establishing a SVC will use the highest allocated logical channel which is not in use. When the packet mode terminal sends a CALL REQUEST packet the network will pass to the called packet mode terminal an INCOMING CALL packet with the logical channel being the lowest allocated logical channel which is not currently in use. In case of a collision the packet mode terminal wins in that it ignores the INCOMING CALL packet and waits for a response to its CALL REQUEST packet.

For more specialised applications many networks allow a packet mode terminal to subscribe to "incoming-only" logical channels (the packet mode terminal is not allowed to make outgoing calls on these channels) and "outgoing-only" logical channels (the network will not send to the packet mode terminal any calls using those channels). Use of incoming-only or outgoing-only logical channels may reduce or eliminate the possibility of call collision.

The following diagram illustrates what might happen when a particular packet mode terminal sends a CALL REQUEST packet to the network. Suppose both packet mode terminals have subscribed to logical channels 256 to 264 for switched calls and no logical channels are in use. The packet mode terminal issuing the call chooses its highest subscribed logical channel which is not in use (264). The network in sending the corresponding INCOMING CALL packet chooses the lowest free subscribed logical channel (256).

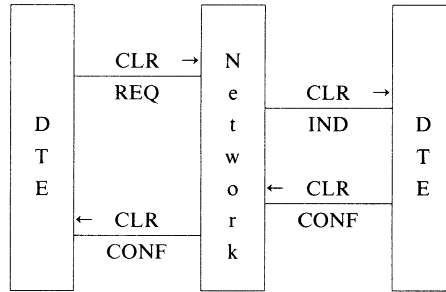



Note that the CALL REQUEST packet and the INCOMING CALL packet have the same format though the INCOMING CALL packet may not be an exact copy of the corresponding CALL REQUEST packet. CALL REQUEST is the name given to the packet sent from the packet mode terminal to the network while INCOMING CALL is the name given to the packet sent from the network to the packet mode terminal. A packet mode terminal can reply to an INCOMING CALL packet with a CALL ACCEPTED packet if it is willing to accept the call (in this case the calling packet mode terminal receives a corresponding CALL CONNECTED packet) or a CLEAR REQUEST packet if it is not willing to accept the call (in which case the calling packet mode terminal receives a CLEAR INDICATION packet).




CLEARING CALLS

Virtual calls are destroyed by either party (or by the network in an "emergency") sending a CLEAR REQUEST packet and the other party responding with a CLEAR CONFIRMATION packet.




The CALL and CLEAR packets may contain additional information such as a request for reverse charging (CALL packets) or call statistics (CLEAR CONFIRMATION packet) but the presence or absence of this information doesn't change the basic action of the switch.

FLOW CONTROL

Once a call is set up flow control is maintained by a "rotating window" mechanism with the receiver sending a RECEIVER READY packet to advance the window.

**Identifying packet mode terminals**

Packet mode terminals are allocated an address of up to 12 digits. The CCITT standard X.121 defines the address format. Additionally at least two digits are available for subaddressing within a particular packet mode terminal. Some networks may require the specification of an escape prefix digit (such as for calls to another network or another country) so an address may be up to 15 digits long.

## Action of the switch

**Startup**

The switch is down-line loaded over Ethernet from a DECnet host supporting this function. Alternatively in a non-DECnet environment the switch code may be loaded from floppy disk.

All KXT11s which are installed in the switch and for which the switch has been configured are loaded with the HDLC code and execution of that code commenced. The switch configuration file is read and HDLC parameters set for each physical line defined in the switch configuration file. Then when the KXT11 notifies the switch that HDLC has started on a particular line a RESTART REQUEST or RESTART INDICATION packet (as appropriate) is sent over that line. On completion of the packet protocol restart procedure the line is marked as being in a state in which virtual calls may be established.

**Post initialisation**

The switch must take significant action on a CALL REQUEST packet and again on a CLEAR CONFIRMATION packet. The address in the CALL REQUEST packet is compared with each entry in a switch table to determine the line along which the call is to be redirected. Then the logical channel table for the target line is examined to find a free logical channel for the next leg of the call. The logical channel tables for both lines are then updated so that each table specifies the logical channel identifier and line for the other call hop and both line tables point to a block used for keeping statistics on the call.

Thereafter packets arriving at the switch are examined to

find the logical channel number and then the line logical channel table consulted to determine the line to which the packet should be redirected and the new logical channel number it should be assigned. If the packet is a data or interrupt packet the call statistics block is updated to keep count of (possibly) chargeable segments sent and received.

On a CLEAR CONFIRMATION packet arriving at the switch the call statistics block is updated with the call clearing time, the statistics written to a log file and the cross line and logical channel linkages cleared.

The call statistics block contains the call start and call finish time, calling line number, called line number, called address, calling address and segments sent and received by the calling packet mode terminal. The call statistics blocks are written to a file which can be subsequently processed to generate bills for departmental computers.

**Switch configuration file**

The details of the action of the switch are defined by the configuration file which is stored on floppy disk. The configuration file is an ASCII text file which sets HDLC parameters for each each communications line to be used, defines parameters for the packet level operation of each line and defines the switch tables to be used for determining where call requests are to be routed. The configuration file also specifies where the KXT11 HDLC code is to be loaded from and where KXT11 crash dumps are to be written.

**Switch Tables**

The switch tables are defined in the switch configuration file and specify the call routing used. For incoming calls from public networks the switch table can be defined so that calls are routed according to the sub address specified in the INCOMING CALL packet so that calls specifying a particular sub address or even class of sub-addresses are routed to a particular line. For calls from campus X.25 nodes switched out to the public network the switch table can specify how the calling address in the CALL REQUEST packet is to be modified by the switch in order that the called packet mode terminal can return the call if it wishes to do so.

## Use of VAX/ELN in the switch

The VAX/ELN runtime kernel provides a very good environment for running the switch code. The extended PASCAL is certainly far easier to use for this sort of application than MACRO-11. The main problems encountered have really been learning ELN PASCAL and working out the most appropriate way to achieve our aim. One of us had written some simple ELN applications previously and read a few of the device drivers while it was the first exposure to ELN for the other.

The switch consists of a single program with a KXT11 load/ dump process which is common for all configured KXT11s and a KXT11 device driver process which is capable of handling up to four KXT11s. (The limit of four KXT11s is because VAX/ELN allowed a process to wait for at most four events.) Additional KXT11s can be handled by creating at run-time additional processes as needed to handle the additional KXT11s. The call switching is handled by subroutine calls from the driver process and providing driver entry points for the switch code to call to request transmission of packets. It was thought that this method would allow higher throughput than would be possible using a somewhat more conventional approach of a separate driver program for each device and message passing between driver and application. On a micro-VAX 1 each message passing operation takes about 3 milli-seconds for a single page and two such operations would be required for each packet if this "conventional" approach were used. Therefore the switch would have a throughput of less than 160 packets per second. The approach adopted should achieve significantly better figures than that but as yet no measurements have been done on actual throughput.

From past experience with this sort of stand alone application it seems the only serious lack of VAX/ELN is the ability to take a crash dump and subsequently analyse the dump to determine what the system was doing when the dump was taken. There are problems that arise due to careless use of dynamic memory, as well as subtle timing problems which are not easily solved with an interactive debugger.

## Planned future enhancements

The switch currently provides the basic set of functions which will meet the current needs of the Prentice Computer Centre. Nevertheless some enhancements are planned for implementation after a stabilisation period and after some experience has shown what additional features would be both useful and desirable.

**New features for KXT11 HDLC code**

The KXT11 performs all HDLC communication in character interrupt mode. There is a DMA controller on the board which could be used with one of the two communications lines. By adapting the software to work in this way it is hoped to be able to support operation of one line at speeds of 48kbps and more with the other line still operating in character interrupt mode at speeds of at least 19.2kbps.

Support for the modulo-127 frame sequencing variant of HDLC would not be a major change but due to the limited buffering available on the KXT11 board is not likely to be a very useful change, especially since the current software supports frame lengths up to 4100 bytes.

It is also proposed to adapt the software to allow use of a clock on the KXT11 board to provide an external clock and remove the need for a synchronous null modem on short haul lines. Use of this feature will be optional and due to the hardware design of the KXT11 will be able to be used on only one of the two lines.

**New features for switch**

Changes to provide support for permanent virtual circuits, incoming only switched virtual circuits and outgoing only switched virtual circuits would not be major, and these features could be useful in certain specialised applications.

Network management facilities currently provided are quite basic and this area will need considerable enhancement, particularly in the area of error reporting and statistics. Currently the switch assumes a DECnet environment for generation of the configuration file, and perhaps the network management facilities could include some method of doing this through X.25.

# DMI TUTORIAL AND DESIGN APPROACHES FOR A VAX-DMI FRONT END

Roger Russ
Advanced Computer Communications
720 Santa Barbara Street
Santa Barbara, California 93101

## INTRODUCTION

This paper discusses the Digital Multiplexed Interface (DMI) specification, its background and its relationship to other similar specifications. It covers all major topics of the specification and gives examples of DMI in a network. Finally, it discusses a design approach to a VAX-DMI front end.

## DMI BACKGROUND

DMI, Digital Multiplexed Interface, is an AT&T specification that defines an interface between a Private Branch Exchange (PBX) and a host computer. It is intended as an interim specification while a standard is developed for integrated voice and data services over telephone equipment. This standard, Integrated Services Data Network (ISDN), is being defined by two international standards bodies: CCITT, the International Consultative Committee for Telephone and Telegraph, and ECMA, the European Computer Manufacturers Association. A competing interim specification, the Computer to PBX Interface (CPI), is supported by Northern Telecom. Both AT&T and Northern Telecom are committed to making their specifications compatible with ISDN's evolving standards.

There is interest in DMI for several reasons. First, since it is sponsored by AT&T, DMI compatible equipment will be common. There is already some UNIBUS equipment that is DMI compatible, and more will be available soon. Second, DMI presents an interesting alternative to more commonly known network configurations. Finally, DMI answers that technical oddity: Why is the T1 rate 1.544 megabits per second?

## MAJOR DMI FEATURES

The DMI specification addresses three major topics. These are framing, how data is presented; signaling, how control information is passed; and data modes, four operational modes that support the migration from existing equipment to ISDN equipment. These topics are summarized in Figure 1 and discussed in the following sections.

**Major DMI Features**

Framing
- D4 Framing – Mandatory
- ESF Framing – Optional

Signaling
- Bit Oriented
- Message Oriented

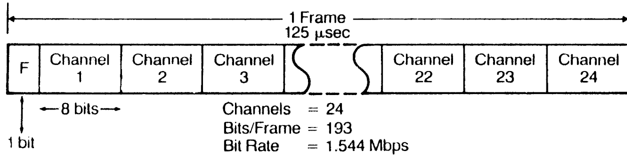
Data Modes
- Mode 0 64 Kbps transmission
- Mode 1 56 Kbps transmission
- Mode 2 Asynchronous/Synchronous terminal transmission
- Mode 3 Virtual circuit service

### Framing

Figure 2 shows the basic DMI frame. It is a serial stream consisting of 24 channels and one frame bit. Each channel contains one byte, so there are 193 bits in a frame. Frames are transmitted every 125 microseconds, an 8k frames/second rate. So, each channel has a 64k bits/second capability. This explains the T1 rate: With 193 bits/frame, 8k frames/second yields 1.544 bits/second. Channels 1 through 23 are data channels and are called "bearer" channels or B channels. Channel 24 is a special channel. It is the signaling channel, called the "dialing" or D channel. As a result, DMI is sometimes referred to as 23B + D.

The final framing parameter is the superframe. A superframe is simply a group of either 12 or 24 frames. DMI has two options for framing, D4 or ESF. D4 and ESF use the framing bit differently.

## DMI Framing




First, let's discuss D4 framing. In D4, there are twelve frames to a superframe. Figure 3 shows the 12 frames and the bit position of every framing bit within the superframe. The framing bit supplies either channel framing or signal framing. Channel framing identifies the start of the first channel and has a repetitive pattern of 101010. Signal framing identifies the "A" and "B" signaling bits. These bits are discussed in the section "DMI Signaling" below. The sequence of the framing bit in signal framing is 001110.

## D4 Framing

| Frame Number | Bit Number | F Bit | |
|---|---|---|---|
| | | $F_s$ | $F_c$ |
| 1 | 0 | - | 1 |
| 2 | 193 | 0 | - |
| 3 | 386 | - | 0 |
| 4 | 579 | 0 | - |
| 5 | 772 | - | 1 |
| 6 | 965 | 1 | - |
| 7 | 1158 | - | 0 |
| 8 | 1351 | 1 | - |
| 9 | 1544 | - | 1 |
| 10 | 1737 | 1 | - |
| 11 | 1930 | - | 0 |
| 12 | 2123 | 0 | - |

ESF is short for Extended Superframe. Figure 4 shows the framing bit in ESF. In contrast to D4 framing, ESF has 24 frames per superframe and has three functions. FPS is the Framing Pattern Sequence, 001011. It identifies the start of a superframe and is used for frame synchronization. FDL is the Facility Data Link. These bits provide a 4k bits/second message facility that is reserved for network use. The last column in Figure 4 is a 6 bit cyclic redundancy check (CRC) for each superframe.

## Extended Superframe (ESF)

| ESF Frame Number | ESF Bit Number | F-Bit Assignment | | |
|---|---|---|---|---|
| | | FPS | FDL | CRC |
| 1 | 0 | - | m | - |
| 2 | 193 | - | - | CB1 |
| 3 | 386 | - | m | - |
| 4 | 579 | 0 | - | - |
| 5 | 772 | - | m | - |
| 6 | 965 | - | - | CB2 |
| 7 | 1158 | - | m | - |
| 8 | 1351 | 0 | - | - |
| 9 | 1544 | - | m | - |
| 10 | 1737 | - | - | CB3 |
| 11 | 1930 | - | m | - |
| 12 | 2123 | 1 | - | - |
| 13 | 2316 | - | m | - |
| 14 | 2509 | - | - | CB4 |
| 15 | 2702 | - | m | - |
| 16 | 2895 | 0 | - | - |
| 17 | 3088 | - | m | - |
| 18 | 3281 | - | - | CB5 |
| 19 | 3474 | - | m | - |
| 20 | 3667 | 1 | - | - |
| 21 | 3860 | - | m | - |
| 22 | 4053 | - | - | CB6 |
| 23 | 4246 | - | m | - |
| 24 | 4439 | 1 | - | - |

### DMI SIGNALING

There are two types of signaling in DMI. The simplest is Bit Oriented Signaling (BOS). The other, Message Oriented Signaling (MOS), provides more functionality.

In BOS, ones and zeroes correspond to a telephone's on hook and off hook condition. These conditions perform all signaling in BOS. For example, the pulsing of on hook and off hook accomplishes dialing. As specified in the basic frame, this signaling information is sent in channel 24. The signaling bits for the 23 data channels are multiplexed into channel 24 as shown in Figure 5. Figure 5 shows the frame number in the left column and the channel 24 bit number across the top. It shows the contents of channel 24 through the entire superframe. Bits 1 and 2 are the "A" signal bits, ordered according to the channel number. Bit 3 is the "B" signal bit. In DMI the A bit is always equal to the B bit. And for BOS, these bits indicate either an on hook or off hook condition. At the DMI transmission rate, BOS provides a 1.5 millisecond update rate for each channel. Bit 7 is a one in every frame except frame 24. This bit is also used for frame alignment. All remaining bits are unused and are always zero. The bottom of Figure 5 shows that frame 24 is unique. It contains a specified pattern in bits 2 through 8 that is used for frame synchronization and frame alignment. Bit 6 in frame 24 is an alarm indicator.

## Bit Oriented Signaling

| Frame Number | Bit Use in Channel 24 | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 1 | A13 | A1 | B1 | X | X | X | 1 | X |
| 2 | A14 | A2 | B2 | X | X | X | 1 | X |
| 3 | A15 | A3 | B3 | X | X | X | 1 | X |
| 4 | A16 | A4 | B4 | X | X | X | 1 | X |
| 5 | A17 | A5 | B5 | X | X | X | 1 | X |
| 6 | A18 | A6 | B6 | X | X | X | 1 | X |
| 7 | A19 | A7 | B7 | X | X | X | 1 | X |
| 8 | A20 | A8 | B8 | X | X | X | 1 | X |
| 9 | A21 | A9 | B9 | X | X | X | 1 | X |
| 10 | A22 | A10 | B10 | X | X | X | 1 | X |
| 11 | A23 | A11 | B11 | X | X | X | 1 | X |
| 12 | 1 | A12 | B12 | X | X | X | 1 | X |
| 13 | A1 | A13 | B13 | X | X | X | 1 | X |
| 14 | A2 | A14 | B14 | X | X | X | 1 | X |
| 15 | A3 | A15 | B15 | X | X | X | 1 | X |
| 16 | A4 | A16 | B16 | X | X | X | 1 | X |
| 17 | A5 | A17 | B17 | X | X | X | 1 | X |
| 18 | A6 | A18 | B18 | X | X | X | 1 | X |
| 19 | A7 | A19 | B19 | X | X | X | 1 | X |
| 20 | A8 | A20 | B20 | X | X | X | 1 | X |
| 21 | A9 | A21 | B21 | X | X | X | 1 | X |
| 22 | A10 | A22 | B22 | X | X | X | 1 | X |
| 23 | A11 | A23 | B23 | X | X | X | 1 | X |
| 24 | A12 | 1 | 1 | 1 | 0 | Y | 0 | 1 |

This may seem complex, but, fortunately, frame detection and signal bit identification are implemented in available VLSI devices. These devices considerably simplify a BOS format implementation.

MOS provides an X.25 link on channel 24. The DMI specification defines a set of MOS messages that provide much more capability than BOS. MOS provides more sophisticated call setup and teardown procedures. It also specifies enhanced maintenance capabilities. The definition of these messages is beyond the scope of this paper. As with BOS, all signaling messages are sent in channel 24.

## DMI Data Modes

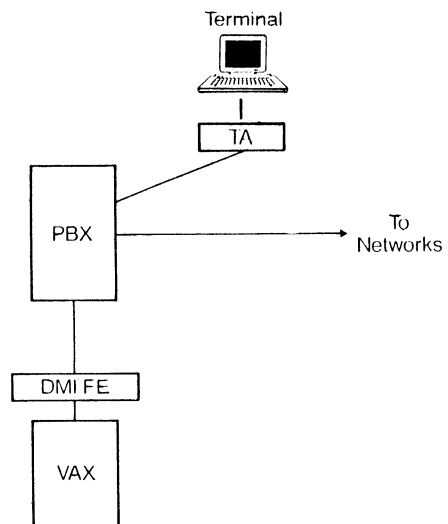
Figure 6 is a summary of the DMI data modes. There are four modes which range from the most simple, mode 0, to the most complex, mode 3. Mode 0 is a 64k bits/second facility. The electrical characteristics of the transmitted data stream, called DS1, has a zeroes density requirement. No more than seven consecutive ones may be transmitted. Mode 0 has no support for maintaining this density requirement, so a mode 0 user must supply the zeroes density. This mode is circuit switched only. This means that connections are made by physical circuits only; there is no virtual circuit support. Mode 1 supports 56k bits/second. The eighth bit in mode 1 is a status bit. This format provides compatibility with existing switching equipment. As with mode 0, mode 1 is circuit switched only. Mode 2 is intended for terminal traffic. It provides a 19.2k bits/second synchronous or asynchronous transmission rate. Frame level High-level Data Link Control (HDLC) is supported. Since HDLC has a ones density requirement, the DS1 zeroes density is satisfied by inverting HDLC data. As in the previous modes, mode 2 is circuit switched only. Mode 3 is a 64k bits/second service. It supports X.25 Link Access Procedure, D channel (LAPD) which provides virtual circuit support.

| | | |
|---|---|---|
| Mode 0 | 64 Kbps | Requires B8ZS coding<br>Circuit switched only |
| Mode 1 | 56 Kbps | 8th bit used for status<br>Circuit switched only |
| Mode 2 | 19.2 Kbps<br>synchronous<br>& asynchronous | HDLC (frame level)<br>Circuit switched only |
| Mode 3 | 64 Kbps<br>synchronous | LAPD<br>Virtual circuit support |

### DMI NETWORK CONFIGURATIONS

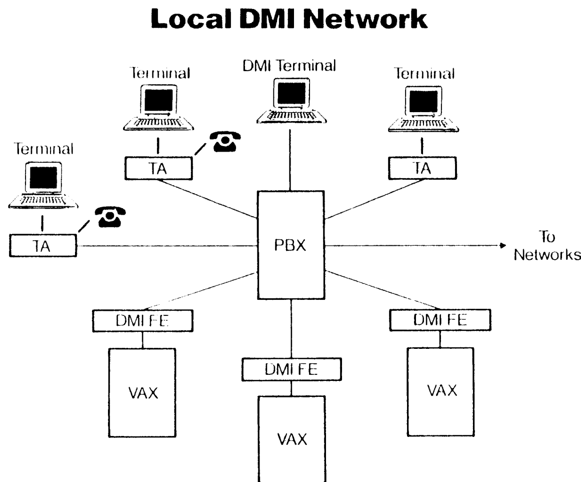
The previous section gave a low level view of DMI. This view loses sight of the utility of this service. Figure 7 gives an example of a minimum DMI configuration. It consists of a terminal, a PBX, and a host computer. The connection between the PBX and the front end is a DMI link. The terminal is connected to the PBX through a terminal adapter which puts the terminal data onto a DMI-like 2B + D transmission circuit. This 2B + D circuit uses existing telephone wiring, and the second data channel may be connected to a telephone. The PBX is also connected to a local telephone switching station. This local configuration

## Local DMI Network



can be easily expanded to include up to 23 terminals for a single DMI line. DMI compatible terminals are becoming available, as are DMI PBXs that may support more than a single DMI line. Figure 8 shows an expanded local DMI network that supports multiple local hosts, multiple terminals, and voice.

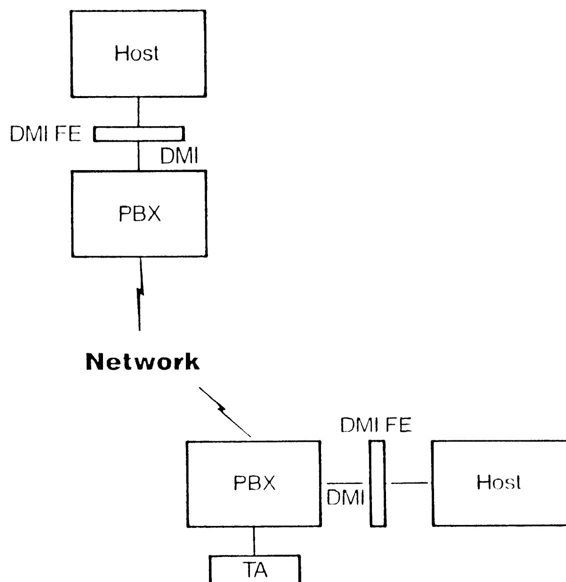
**Local DMI Network**




It may seem that this arrangement offers no advantages over other Local Area Network (LAN) technology. However, remembering that both data and voice applications share the same DMI equipment, some benefits become more apparent. For example, most offices already have a PBX, so a PBX in a configuration like that in Figure 8 could replace an existing device. The 2B + D lines use existing telephone wiring. This saves not only the cost of terminal cables, but the cost and problems of installation. Finally, the DMI link between the PBX and host reduces host cables by as much as 23 to 1.

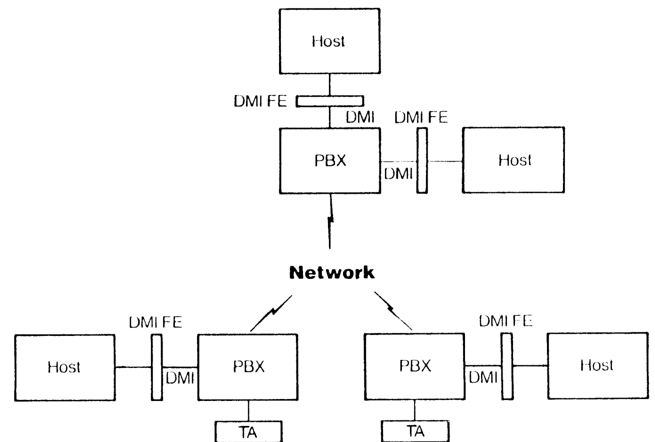
Figure 9 shows DMI in a wide area network. The network is achieved through existing T1 lines and

## Wide Area DMI Network




telephone switching equipment. A remote terminal, which would be attached through the terminal adapter (TA) at the bottom of the figure, simply places a call to the appropriate number. The connection is established, and the user logs on. The PBX and telephone switching circuits handle the routing. As in the local example, this configuration is easily expandable. Figure 10 shows an expanded network. Local site expansion happens by attaching additional equipment to the PBX. A new site results by adding another DMI PBX.

## Wide Area DMI Network




## VAX-DMI FRONT END PROCESSORS

The section discusses VAX-DMI front-end design. The most influential design consideration is the availability of DMI support parts. Three groups of parts will be reviewed here. They are AT&T's DS1 parts, Rockwell's R8071, and AT&T's Spyder.

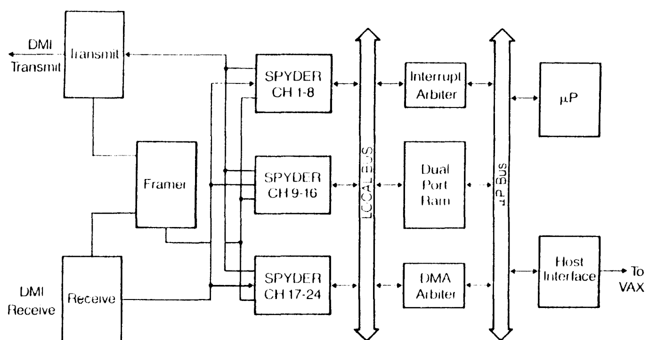
The DS1 parts provide the electrical interface to the DMI lines. In addition to the electrical interface, these parts provide frame synchronization and extract signaling bits, and they provide parallel data on the internal side. Depending on the internal interface, this part set may consist of up to six VLSI devices, two transformers, and a timing generator. The minimum interface would probably use three VLSI devices, two transformers, and a timing generator.

The R8071 provides additional internal support for the DMI channels. It provides DMA for the parallel data, additional frame processing, and HDLC framing. This part is compatible with the DS1 set. The R8071 can support either 24 or 32 channels, so only one part is necessary for a DMI application. This part is currently in development, with samples expected in the first quarter of 1987.

The Spyder provides very similar functionality. As with the R8071, the Spyder provides DMA, additional DMI frame processing, and HDLC framing. It is also compatible with the DS1 set. The Spyder supports eight channels, so three parts are necessary for a DMI application. This part is currently available in sample quantities.

Since the Spyder is available, let's consider a design using the DS1 set with the Spyder. Figure 11 shows a simplified front end architecture. The DS1 set is on the left of the figure. It consists of a transmitter, receiver, and framer. The framer performs some DMI frame functions. The three Spyder parts are to the right.

**VAX DMI Front End**




They attach to the DS1 transmit and receive parts on one side and to a local bus on the other side. This bus time multiplexes address and data. An essential feature of this design is the dual port RAM. One side of this RAM interfaces to the Spyders, and the other attaches to a microprocessor bus. The microprocessor section can include a variety of parts for microprocessor program support. The last essential element is the host interface to the VAX.

## SUMMARY

This paper has introduced DMI, and has given a summary of the DMI specification and some examples of its use. DMI is to be understood as a current implementation of the emerging ISDN standard. It offers advantages for users in streamlining redundant equipment and cabling as well as providing an interesting alternative to local and wide area networks. DMI hardware is becoming available now, with more pieces available in 1987.

# TransLAN Technical Product Overview

# And Network Configuration Guidelines

Michael R. Coker
Vitalink Communications Corporation
1350 Charleston Road
Mountain View, CA 94043

## ABSTRACT

An overview of Wide Area Network problems, implementation strategies, and planning concerns. Solutions based on Ethernet-LAN Architecture as defined by Digital's XLII Bridge Specification are presented. An overview of Digital's LAN Bridge 100 and Vitalink's TransLAN are included with detailed explanations of technology, user applications, and network design guidelines.

### TransLAN Product Background and Overview

Today there is a strong movement within the communications industry to standardize, that is, create a standard set of rules for all computers and communications devices to follow when interchanging information. These rules, or "protocols" will allow for the easy exchange of information between systems regardless of manufacturer, thus creating what is being termed as an "open systems environment."

The open systems environment is defined by a seven layer model called the International Standards Organization's Model for Open Systems Interconnect (ISO-OSI). Using this model, the IEEE has defined several committees to set these standards. They have had particular success in defining the standards for Local Area Networks (LANs) which accounts for much of the success of the LAN market in the last few years. Through the standards these committees have defined, commercial implementations, most notably Ethernet, IBM's Token Ring, and GM/Boeing's Token Bus have gained wide industry acceptance. To date, Ethernet is the most prevalent LAN implementation and should continue to grow; however, the new token technologies promise to expand the market even larger. The Yankee group estimates that by 1988 the value of the LAN market will be nearly $4 billion. According to Strategic, Inc. that represents about 125,000 Local Area Networks.

Standardizing a Local Area Network requires defining two of the seven layers of protocols as defined by the ISO model, specifically the Physical Layer and the Data Link Layer. The Physical Layer is the lowest layer of protocol in a network and defines the rules of communicating across the actual transmission media. Transmission media may be a cable, a phone circuit, a microwave system, a fiber optics path, a satellite link, or even a twisted pair of wires. Defining a Physical Layer Protocol means describing rules for physical characteristics such as the shape of the connector and the level and sequence of the electrical signals. The Physical Layer rules are not concerned with information content, only electrical and physical specifications.

In IEEE 802 Committees' terms, the "Data Link Layer" defines what is termed as a "datagram". A Datagram is a packet of information that needs to be passed through the network. Here the rules apply to the format of the information that will pass through the physical media. It is analagous to defining the shape of a parcel or envelope for the First Class Mail "network." An example of a link layer protocol for First Class mail would be an envelope which must not be larger than so long, so wide, and so deep and it must not weigh more than so much. The "mail protocol" must also specify where the destination and source of the mail will be noted on the envelope (destination in the middle -- source in the upper left corner) in order to properly forward the information through the mail network. Data Link Layer Protocols define the size and characteristics of the envelope of information, in data communications this is called a "Datagram," and it also defines the format of the envelope so that it can accurately interpret the source and destination addresses for routing through the data network.

By defining the physical media protocols and the link layer protocols, Local Area Networks such as Ethernet and Token Ring are created. All the rules necessary to carry the information across a common physical media are determined: the physical connection, the electrical signals, and the format of the information package. By adhering to these rules, any manufacturer can develop and market devices which will communicate over the Local Area Network. Thus the communications consumer can create a common data highway for connecting the various computers and communications equipment he is managing. In Local Area Networks, typically the data highway is a coaxial or fiber optics cable.

This technology, Local Area Networks, has effectively addressed many of the problems faced by communications managers today. It has provided a high performance, low cost solution to moving information between users when they are geographically local. LANs have been installed in data centers, manufacturing facilities, engineering

shops and departmental offices. Where installed, LANs proved to be efficient communications networks, easily managed and useful for almost all the local communications requirements. Vendor interfaces to LANs are rapidly available once the LAN is standardized. LANs provide high performance and offer many enhancements over traditional serial line networks. The result is best shown in the success of the LAN market to date.

Vitalink Communications Corporation recognized the impact of the LAN technology in two areas. First, as more and more LANs are installed, a need was generated to interconnect them so that information could flow from one local environment to another with the same ease and relative performance offered to users locally. Vitalink recognized the need to extend the local applications and resources over a wide area. Secondly, it was determined that the design of networks based on this new Link Layer technology may potentially displace traditional solutions in many areas of the communications marketplace other than LAN-to-LAN communications.

In 1985, the options available to the data communications manager were limited. The traditional solution for interconnecting devices was to use what was termed a **router**. A router is a protocol specific communications processor that connected stations to a wide area network of serial communications circuits. A router is a ISO Network Layer device, the third level of protocol in a data network and above the Physical and Data Link Layers. Using a router to connect LANs allowed for any device connected on one LAN to communicate using a certain protocol (or language) with other similar devices on other LANs. The problem was that almost every vendor has its own Network Layer protocol. Digital Equipment uses DECnet; Xerox has a protocol called XNS; and the government standardized on a protocol called TCP/IP. Since a LAN is defined by protocols below the Network Layer, it was capable of supporting the multiple protocols of the various vendors. The routers between the LANs, however, were limited to a subset of the stations that may exist on a LAN.

The better solution seen by Vitalink was what is defined by the ISO as a **Data Link Layer Bridge**. A Data Link Layer Bridge is a communications processor that operates only with the two lowest layers of protocols: the Physical and Data Link Layers, the two protocols which define a LAN. A "bridge" logically extends the LAN and thus can extend all of the vendor dependent networks that normally coexist on a LAN. A bridge allows any LAN station, irrespective of vendor or protocol, to communicate with other like stations anywhere in the bridged network. In short, a bridge offers another level of connectivity beyond the traditional router, one that matches the connectivity of a LAN.

Other inherent features of a bridge were identified. Because a bridge extends the LAN using only LAN protocols, all stations in all bridged LANs appear to be local, that is, the users and protocols perceive all stations as if they were connected to the same physical cable (except in terms of delay where slower speed, wide area circuits must be used for connection and may slow response time). This transparency makes the user interface to the network simple and allows for the easy management architecture of LANs to be preserved. Without the overhead of processing the Network Layer protocols it services, a bridge has inherently more performance than a router. A bridge is a relatively simple device, but extremely fast. It routes data based on Link Layer addresses which, for 802 protocols, are in flat address spaces. An address space is flat if all devices that exist in that address space have a unique address. Ethernet and 802 standard address spaces are controlled such that all devices that interface to the "standard" LAN have a unique hardware address. Taking advantage of the defined, flat address space in 802 LANs, bridges can use adaptive learning as a basis for routing decisions and simplify network operations. With adaptive learning, bridges need not be told where stations are physically located. Instead, the bridge dynamically learns the network stations' locations and the necessary information to deliver datagrams to them by remembering from which direction the bridge heard that station transmit. Listening to all traffic from all connected networks, bridges build tables of source addresses noting the receive port. When the source address is seen again as a destination, bridges forward the datagram, if necessary, to the appropriate network. This adaptive learning algorithm, patented by Digital Equipment Corporation, greatly simplifies network management; gives responsiveness to network change; and can drastically decrease overall network life cycle costs.

To meet the LAN interconnct marketplace and implement this powerful new technology, Vitalink and Digital Equipment Corporation collaborated to define the first commercial Data Link Layer Bridge. Using this definition as a basis for a co-marketing relationship, Vitalink designed and built TransLAN(R). TransLAN is the first true Data Link Layer Bridge. TransLAN was first exhibited at DECWorld 85 in December, 1984 and Vitalink began delivery in January of 1985.

Since then, the TransLAN Data Link Layer Bridge has become the premier internet processor for connecting IEEE 802.3 and Ethernet LANs. TransLAN offers a new solution to the traditional problems of connecting remote users, file transfer, and office automation. With TransLAN, a high performance, protocol insensitive Wide Area Network can be built on an international standard architecture. TransLAN extends IEEE network standards into the wide area networking environment.

TransLAN is a field tested, established product. Currently TransLAN is installed at over 300 locations world-wide supporting remote user populations through terminal servers, high performance file transfer traffic in distributed data processing environments, supercomputer and CAD/CAM applications, and geographically dispersed office automation and time-sharing applications.

Simply stated, TransLAN interconnects two or more IEEE 802.3 or Ethernet LANs with high speed synchronous circuits. The connection is protocol insensitive to the Network Layer Protocols which attach to these LANs (such as DECnet, LAT, XNS, or TCP/IP) allowing all attached stations to communicate over the same wide area circuit. TransLAN will support any protocol that adheres to IEEE 802.3 or Ethernet Version 1 and 2 Standards. For LAN-to-

LAN communications, TransLAN has become the "universal router."

TransLAN requires no system generation to make its routing decision. Through a proprietary learning algorithm, TransLAN dynamically learns the necessary information required to deliver LAN datagrams to their proper destination station. Additionally, TransLAN offers unsurpassed performance specified at over 1500 datagrams per second forwarding compared to the 150-300 datagrams per second rate of most internet routers or traditional packet switches.

Management of Wide Area Networks is a primary focus of Vitalink's Network Products strategy. TransLAN, as the foundation of our product line, capitalizes on its Link Layer architecture to supply a comprehensive, network-wide management system that offers network managers and planners new and powerful tools necessary to optimize their network resources. Through an integrated management system, TransLAN supplies Link Layer statistics on traffic levels, errors, and congestion for all connected LANs while remaining transparent to and co-residing with existing LAN management schemes and services. TransLAN management can be either centralized or regionalized, providing visibility and reporting for all or part of the Wide Area network. The design of Vitalink Management Services insures that as network management services become standard, it can easily integrate standard interfaces into its already standard bridge architecture.

Today, TransLAN III represents the third implementation of this new Data Link Layer technology. Using its unique experience in the marketplace, Vitalink has given TransLAN III a new hardware base that is faster, smaller, and less expensive than its predecessors and promisess to maintain TransLAN's lead in this new, explosive marketplace. With new software features, TransLAN III introduces capabilities that allow greater network flexibility, performance, and reliability. TransLAN III can support multiple circuits between LANs with load balancing or operator specified routing. In its latest version, TransLAN III supports AT&T's ACCUNET Switched 56 services for dial-up network requirements.

In addressing other potential applications, Vitalink has begun developing network design parameters and companion products for consolidation of SNA, LAN, HDLC, and X.25 traffic into an integrated Wide Area Network solution. Two additional Vitalink link layer servers, TransLINK and TransSDLC, provide IEEE 802 Link Layer interface to bit synchronous circuits and IBM 3270 devices. These devices, based on TransLAN technology, provide a transparent, software-defined network using an 802 standard backbone. This allows SNA, X.25, HDLC, and IEEE 802 LANs to share a single, high performance Wide Area Network. TransSDLC adds the enhancement of local polling emulation to eliminate polling from the wide area circuits thus improving response time and bandwidth utilization in IBM-based 3270 networks.

In supporting the emerging LAN technologies, TransLAN is targeted for all three major Link Layer technologies: 802.3 CSMA/CD, 802.4 Token Bus (MAP/TOP), and 802.5 Token Ring (IBM). An interesting feature of bridges such as TransLAN is the commonality of Link Layer packet formats throughout the 802 standards. Because TransLAN works only on the Link Layer, it is possible to use the bridging technology not only in all three markets, but also to interconnect the different architectures. We have product plans for 802.4 to 802.4 version, and 802.5 to 802.5 version, as well as bridges to interconnect the three different standards, as market requirements develop.

In summary, TransLAN is a new kind of Wide Area Networking Processor, a Data Link Layer Bridge based on IEEE 802 protocols. It has found applications ranging from interconnecting Local Area Networks (instead of traditional routers and gateways), supporting remote user access (instead of switching multiplexers and port selectors), packet switch applications (instead of X.25 switches), and multi-protocol Wide Area Networking. We have penetrated these markets due to high performance, protocol insensitivity, and manageability at a price-/performance unsurpassed in the industry. Add to this the security of building a Wide Area Network based on standardized interfaces, TransLAN offers a unique solution to many of the problems confronted by today's communications managers, many of whom are now faced with the problem of integrating their corporate networks.

TransLAN's initial hardware base, the Network Processor I (NP I), was the world's first true bridge. It was often called the VB/1 for the Vitalink Bridge 1. Its application, TransLAN I, was introduced in January, 1985 and was capable of interfacing to Ethernet and IEEE 802.3 LANs. It supported RS232, RS422/449, and V.35 serial interfaces at speeds up to 224 Kbps. It was based on multi-68000 microprocessors and had up to 256 kilobytes of shared memory. The NP I was upgraded with new hardware called the Network Processor II (NP II) and new software features (TransLAN II) in May, 1985. Upgrades from NP I hardware to NP II hardware are available from Vitalink. TransLAN I is compatible with TransLAN II and can co-exist in the same network as TransLAN III.

Using faster microprocessors (12.5 MHz 68000's), and a new Ethernet Interface, the NP II can support serial interfaces up to 448 Kbps. The upgraded hardware has expanded program and shared memory to support additional software features. It provides an improved packet filtering rate with the implementation of high speed cache memory. TransLAN II is currently being shipped by Vitalink and supports up to 8 serial circuit connections (at speeds up to 56 Kbps each). TransLAN II can be directly link attached to TransLAN I or communicate directly with TransLAN III with Software Release 3.7 or later.

TransLAN's latest hardware base, Vitalink's Network Processor III (NP III), is a high performance general purpose communications processor based on multiple Motorola 68010 microprocessors. The hardware is implemented with 1 megabyte user memory (shared buffer) and 640 kilobytes program memory, all parity protected. The bus that interconnects TransLAN's hardware core to the serial link and LAN interfaces is also parity protected to insure absolute data integrity. The NP III acts as a hardware

host to all Vitalink Network Products: TransLAN, TransLINK, and TransSDLC.

The Network Processor III was designed using our unique expertise and experience in Data Link Layer devices. The NP III can process over 15,000 frames-per-second (filtering) and forward over 1500 frames-per-second steady state. Peak forwarding rate exceeds over 2400 frames-per-second. Using multi-ported memory and fast DMA channels, the NP III is one of the fastest packet switches available today. The NP III is designed to be modular in architecture, allowing easy expansion and change as network requirements evolve. Software and firmware is loaded via twin 3 1/4" floppy drives on power-up or by operator request. In-band, downline load options will be available in the near future.

Currently the NP III supports a proprietary Ethernet-/IEEE 802.3 interface that meets the high speed (10 Mbps) requirement of interfacing to an Ethernet or IEEE 802.3 LAN. The LAN interface runs in either a selective or promiscuous mode. In the promiscuous mode, all frames transmitted on the LAN are received and captured by the interface for a forwarding decision. If the frame was destined for a local resource on the LAN, the frame is discarded. If the frame was destined for a resource on a remotely connected LAN, the frame is queued to the appropriate serial port for transmission. The LAN interface card was designed to allow capturing of the entire LAN frame including the originally transmitted CRC (cyclic redundancy check) characters. This allows the unique feature of forwarding the complete frame (including original CRC) to the destination station, insuring the validity of the data.

The NP III initially supports a single V.35 clear channel Customer Interface Card (CIC). A V.35 CIC has four physical CCITT V.35 ports. The board is capable of supporting a single T-1 circuit or multiple slower circuits. CIC throughput when supporting multiple circuits is limited to 2 Mbps aggregate. Each port can act as DCE or DTE, providing or taking clock as the requirement demands. The CIC transmits frames intact, that is, it transmits the LAN frames in their native format using an ISO 3309 protocol (a subset of the HDLC specification). The CIC adds only an additional CRC for transmission over the serial link. The CIC detects all Link Layer errors and reports error conditions to management software.

Support of other serial physical interfaces such as RS232 and RS422/449 are future options of the Network Processor III. Requirements which demand these interfaces are supported with Vitalink's Network Processor II. The NP II supports RS232, RS422/449 and V.35 at speeds up to 448 Kbps. DSI interfaces are not currently supported for T-1 circuits. See section on T-1 Configuration Considerations for details.

### TransLAN Network Design Considerations

Hardware

Hardware configuration for TransLAN is relatively straight forward. The following is a summary of configuration options. It should be pointed out that TransLAN is generally a factory configured system. Vitalink provides a configuration worksheet with every order for the purpose of defining the specific configuration options required. It is essential that this worksheet is completed prior to ordering TransLAN.

TransLAN II

TransLAN II can support up to eight ports with a total aggregate throughput of up to 996 Kbps. Each serial port option (SIO board) can be RS232, RS422/449, or V.35 and has two physical interfaces which can operate as either DCE and DTE. The system comes standard with one serial port option (two ports).

An SIO board can support up to 224 Kbps of throughput with both ports. This allows support of two 56 Kbps circuits per board (56 Kbps simplex times two for duplex operation, times two ports equals 224 Kbps of throughput). Speeds lower than 56 Kbps are supported on both ports as well. For circuit speeds greater than 56 Kbps, two SIO boards are used, one for transmit and one for receive. A Y-cable is required from Vitalink to marry the two simplex connections into a duplex port. Using these specifications, the following guidelines for TransLAN II circuit support:

One SIO board can support up to two 56 Kbps circuits.

Circuit speeds greater than 56Kbps require two SIO boards.

A single TransLAN II system supports up to 4 SIO boards.

Maximum circuit support for a single TransLAN II is eight.

Eight circuits up to 56 Kbps each can be supported using 4 SIO options.

Two circuits up to 224 Kbps each can be supported using 4 SIO options.

A special configuration of TransLAN II called Fast I/O allows a TransLAN II system to operate over circuits of up to 448 Kbps in a point-to-point configuration. This configuration includes two SIO boards and a Y-cable. No other connections are available. TransLAN II Fast I/O supports only one serial circuit, thus one remotely connected LAN. An upgrade from TransLAN II to TransLAN II Fast I/O is available from Vitalink. This configuration is recommended only for upgrades to existing TransLAN II systems. TransLAN III is now recommended for support of 448 Kbps and above.

Expansion of TransLAN networks beyond the discrete circuit support of a single system is simple and direct. Multiple TransLAN systems can attach to the same LAN with circuits connecting diverse LANs at different speeds. For example, one TransLAN II system may connect to another at 448 Kbps, utilizing all the available throughput of the system. The connection to additional LANs is facilitated through additional TransLAN systems. A second TransLAN II could be added to support a 224

Kbps circuit to a third remote LAN, and up to four 56 Kbps circuits to additional remotes. TransLAN systems and interfaces can be modularily added for any network expansion requirement.

TransLAN III

TransLAN III is currently sold in only one configuration: an Ethernet Version 2/IEEE 802.3 interface and a single SIO board (called a customer interface card--CIC) with four physical V.35 connections. Each port is capable of supporting a clear channel circuit of speeds up to 2.048 Mbps duplex. The CIC board is capable of supporting up to 4.096 Mbps of throughput (or one T-1 link: either 1.544 Mbps or 2.048 Mbps full-duplex). Attaching one T-1 circuit nullifies the use of the other three physical ports. When using multiple circuits slower than T-1, all four ports may be used as long as the aggregate throughput of the circuits does not exceed 2 Mbps. For instance, four 56 Kbps circuits could be supported, or four 224 Kbps circuits. All variations of circuit configurations are not yet tested, so cases where circuit throughput is close to the specification or questionable should be coordinated with Vitalink Technical Support prior to configuration.

Special Considerations for T-1 Circuits

T-1 circuits generally come in three varieties: Clear channel circuits, intra-LATA circuits and inter-LATA circuits. A clear channel circuit is a private circuit that does not require special formatting for network operations. Examples of clear channel circuits are satellite circuits, private microwave circuits, and private fiber optics circuits. In all clear channel cases, a public switching network like those of the Bell Operating Companies and AT&T ACCUNET are not used. The end-to-end clear channel circuit must be totally private.

Intra-LATA T-1 circuits require a minimum formatting for switching within the Bell system. The Bell system typically requires that the physical interface be DS1 and that every 193rd bit be a framing bit (called D4 framing) and creating the basic T-1 frame. This uses 8 Kbps of the 1.544 Mbps for framing overhead leaving an effective 1.536 Mbps for data.

Inter-LATA T-1 circuits typically use AT&T's Accunet services or similar service from another long distance vendor. In these cases, a T-1 circuit must have a DS1 physical interface, D4 framing must be implemented, and "ones density" must be maintained. Ones density requires one in every eight bits to be a one (this is due to the nature of data to have many consecutive zeros and the need of the switching system to see a bit transition periodically to maintain phase and clock).

For clear channel systems, TransLAN's V.35 interface may be directly connected to the fiber, microwave, or satellite modem. When intra-LATA circuits are used, TransLAN requires the addition of a V.35 to DS1 interface converter and D4 framing. This functionality is available with several CSU's commercially available. Avanti and Verilink have product offerings that have been tested with TransLAN.

For inter-LATA circuits, a conversion from V.35 to DS1 is required as well as framing and ones density. This functionality is provided through various devices commercially available either through Vitalink or other communication equipment vendors. Avanti's Accupac and Verilink's Clear Channel Unit (VCC) have been certified for use with TransLAN III. Caution is recommended whenever implementing T-1 networks. The T-1 market is young and several factors should be considered in network planning.

T-1 Standards Stability

It can be very confusing trying to determine the specific requirements for interfacing to T-1 circuits. There are two basic reasons. First, the various vendors and suppliers have yet to agree on exactly what formatting and features should be supplied. AT&T has its own thoughts, standards organizations another, and vendors yet another. Each is still going its own way designing the future which leaves the consumer the the task of determining how they interact. Services from multiple sources (i.e. local loop from the BOC, fiber connection from another source, and another local loop from yet another BOC) should be examined for all combined requirements. There are several methods, for instance, for insuring ones density (ZBTSI or B8ZS are two) and switches in the T-1 network may require a specific method. Using the wrong method or combining incompatible methods may significantly degrade the T-1 circuits performance or it may fail to operate at all. Vitalink has found that some CSU's invert the input signals over the T-1 link. In doing so, they functionally assure that ones density will always be maintained in TransLAN networks. TransLAN uses an ISO 3309 protocol in which HDLC flags are continuously transmitted when the link is idle and during transmission data bit stuffing occurs to prevent the occurance of flags within the data stream. The HDLC flag is six consecutive one bits followed by a zero. No more than six one bits in a row are allowed, therefore when inverted, no more than six zero's are transmitted before the ISO protocol automatically "stuffs" in a one. This eliminates the requirement of the CSU or the network switches to ever compensate for ones density. This eliminates most concerns for ones density compatability.

Secondly, T-1 standards are still evolving and new features are said to be "just around the corner." The superframe format is one example. A superframe is currently used by many vendors. A superframe consists of 12 T-1 frames of 193 bits each. The framing bits are transmitted in a certain pattern and can be checked for proper sequence. Framing errors can therefore more easily be detected. AT&T is said to be close to implementing an extended superframe specification within ACCUNET, however, one cannot expect all switches to be converted overnight. An extended superframe consists of 24 standard T-1 frames of 193 bits. The use of these bits is still not cast in ink, however it is likely that six bits of the 24 will be sync bits, six bits will be a CRC, and 12 bits will be used for BX.25 in-line control and test purposes. This divides the 8 Kbps of overhead

into 2 Kbps of sync, 2 Kbps of CRC, and 4 Kbps for signaling and testing. The BX.25 protocol is said to be favored for this purpose and will allow AT&T to signal changes to network devices in-band, acquire status and operating parameters without interupting service, and even run some testing without taking the link out of service. It may also define some of the internal channels as control channels, yet formats and specifications are still not published and finalized. One should consider the impact of future changes in services before implementing T-1 circuits. Obviously, extended superframe features may greatly enhance the T-1 circuit's reliability, but who knows when all devices in your circuit will support it.

## Circuit Functionality Considerations

Determining the functionality of synchronous circuits is not necessarily a straightforward operation. In our experience with synchronous circuits, we have found traditional Bit Error Rate Testing (BERT) is not an accurate evaluation of a circuit's ability to transfer synchronous HDLC data. In one instance, a BERT test ran perfectly for over four hours, but the line would not transfer HDLC data effectively. In troubleshooting the problem, two cables, a CSU, and an internal DDS switch were found bad, in spite of the glowing success of the BERT test. Vitalink recommends the use of a protocol analyzer where possible to ensure that the link is operational and meets specifications. BERT test success should not be taken as an indication of a circuit's ability to actually move synchronous data. As an additional tool, Vitalink also recommends using our management system statistics to verify proper operation of the link. If it reports significant abort errors, clocking and synchronization errors should be suspected. If CTS errors are reported, the CSU/DSU should be suspect. Vitalink's management statistics have proven to be a excellent measurement system to determine the quality of a given circuit. Its link layer window into a network offers a true look at any given circuit's performance in actually sending data.

## Connecting TransLAN to Various LANs

TransLAN supports any Ethernet or IEEE 802.3 connection. It is important, however, to realize the implications of the various versions of the "standard" that exist. Ethernet, the most prominent commercial implementation of IEEE 802.3, comes in basically three versions: Version 1, Version 2, and Cheapernet (or thin-wire Ethernet). All devices are compatible at the link layer, that is, they all conform to the same packet format. There is some concern in the physical connection of each device to the LAN. A connection to a LAN consists of three components: the cable, a transceiver, and a LAN controller. The controller must match the transceiver, the transceiver must match the cable. For instance, a Version 1 controller must use a Version 1 transceiver, however it can communicate with any other station on the LAN... Version 1 or Version 2. Likewise, a Version 2 controller must use a Version 2 transceiver and can communicate with any other device on the Ethernet, irrespective of version. If the LAN uses a thin cable (Cheapernet), the transceiver must be a thin-wire model and be the proper version (1 or 2) of Ethernet. Bridging thin-wire and standard Ethernet is possible as well, but the physical interconnect must also match in Version.

IEEE 802.3 is slightly different than Ethernet at both the Physical and Link Layers. IEEE 802.3 stations cannot generally communicate directly with Ethernet stations. (For instance, in Ethernet, bytes 13 and 14 of the Ethernet frame are reserved for packet type which is generally an indication of protocol. In IEEE 802.3, bytes 13 and 14 are reserved for Logical Link Control, a sublayer IEEE defined within the Link Layer. At the physical layer, there are differences in LAN topology, for instance, Ethernet prohibits more than two repeaters between any two stations. IEEE 802.3 allows up to four.) Both Ethernet and IEEE 802.3 stations can, however, share the same cable. Although there are very few true IEEE 802.3 stations currently available, many are being forecast as product offerings in the near future. As new "standard" products become available, it is expected that many may move to the new products. For these products, a IEEE 802.3 standard transceiver is required.

TransLAN is designed to bridge both sets of protocols: Ethernet and IEEE 802.3. Both may simultaneously use TransLAN's services. Because the first 12 bytes of the standard frames are always destination and source addresses, TransLAN can route either type of frame. Vitalink has implemented several features that are applicable only to Ethernet. Packet Type deflection for protocol routing is one example. Because there is no packet type field for IEEE 802.3, TransLAN cannot implement that feature in IEEE 802.3 LANs.

Connecting TransLAN to the LAN is relatively simple. TransLAN II supports any Ethernet or IEEE 802.3 transceiver. Connection to the transceiver is with any standard transceiver cable, however, the DEC transceiver cable does not mate securely with TransLAN II's Ethernet Controller and should not be used. TransLAN III can use any Version 2 or IEEE 802.3 transceiver today and will support Version 1 transceivers (through a software setable option) in the near future. Once again, any transceiver cable can be used but caution is advised when using the DEC standard cable. To insure compatibility and reliability, Vitalink recommends that you purchase the transceiver cable and transceiver for the TransLAN connection from Vitalink. Vitalink carries two kinds of transceivers: a standard transceiver and a high-reliability transceiver. The standard transceiver is manufactured by TCL and is Ethernet Version 2 compatible. The high-reliability transceiver is DEC's H4000 / H4005 transceiver and is designed with built-in redundancy. Additionally, Vitalink offers Digital's DELNI. The DELNI is an Ethernet star coupler. It can be an extension of an Ethernet cable or stand-alone as an "Ethernet in a box." The DELNI offers eight ports for connecting eight Ethernet Version 2 controllers via standard Ethernet transceiver cables. Switch selectable, the DELNI can act as a stand-alone Ethernet or connect to a Ethernet cable with a H4000/H4005 transceiver. The DELNI offers an inexpensive way to connect locai devices together and/or interconnect them with other devices connected to an Ethernet.

## TransLAN Architecture and Software Features

### Overview

The following is a discussion of what TransLAN is, how it works, where it can be used, and the implications and concerns that should be evaluated when extended LANs are designed with TransLAN. The final section lists a set of guidelines that should be followed when installing TransLAN. These guidelines are meant to be general rules, and although there are always exceptions to rules, any deviation should be thoroughly evaluated against the information provided in this document.

### Data Link Layer Bridges: Filtering, Forwarding, and Learning

TransLAN is functionally a Data Link Layer bridge. TransLAN acts as a store and forward packet switch for Ethernet and IEEE 802.3 packets. As a LAN bridge, it is a device that interconnects LANs allowing stations connected to different LANs to communicate as if both stations were on the same LAN. A collection of LANs and bridges is referred to as an extended LAN.

TransLAN has two primary operations: filtering and forwarding. TransLAN is designed to rapidly make decisions as to the destination of a datagram. TransLAN will either discard the packet or forward it to one of its attached networks.

TransLAN networks require no routing or internet information to be supplied by the sending station. Bridges make use of the Data Link Layer addresses to make forwarding decisions. They have no knowledge of any other address space, such as network or internet address space. Because of this characteristic, TransLAN is relatively insensitive to the higher layer protocols used by the communicating stations.

Bridges do not relay all packets like a repeater. Bridges isolate LANs from traffic which does not need to traverse that LAN. The bridge forwards frames based on information it has learned from the network. By observing the frames on its data links, the bridge builds a station list, called a "Forward Data Store," of all end stations that have transmitted. An entry in this list comprises a Data Link Layer address and the corresponding network it was received on, and the age of the "Forwarding Data Store Entry (FDSE)."

The decision of whether or not to forward a frame is determined by this "learned" station list. For each frame received the bridge compares the Data Link Layer destination address against this list. If the destination is local (i.e. it was previously seen as a source address on the same data link), the frame is discarded, or "filtered"). If the destination is known to be remote (i.e. it was previously seen as a source address on a different data link), it is forwarded to the network indicated by the station list. If the destination is unknown, the frame is broadcast to all remote data links.

To insure the performance of the bridge algorithm, the topology of the extended LAN must be a spanning tree (in terrestrial networks), that is, loop free. A spanning tree algorithm has been developed and is documented in Digital Equipment Corporation's XLII Bridge Architectural Specification and has been approved by the IEEE. This algorithm is used to detect and break deliberate or inadvertent loops and maintain the integrity of the extended LAN.

In summary, this bridge architecture allows for the transparent interconnection of LANs. The routing performed by the bridges does not require direct participation of the end stations. Two stations on separate physical LANs can communicate through bridges (irrespective of internet protocols) as if both were attached to the same physical LAN. The topology of the extended LAN must be a spanning tree (i.e. loop free).

### Types of Bridges

Bridges can be classified into two categories: Local Bridges and Remote Bridges.

### Local Bridges

Local bridges are inherently limited by distance and typically connect to LANs at, or near, the native LAN speed; in the case of Ethernet, at 10 Mbps. Typically, connection is over coaxial or fiber optic cables and is geographically limited to locations which can be easily connected via these media. These would include multiple LANs within the same building or campus environments where LAN locations are within 2000 meters of each other. An example of this technology is Digital Equipment Corporation's LAN Bridge 100.

Since local bridges interconnect LANs at their native speed, they typically do not interject noticeable delay nor network congestion. However, local bridges are limited to those applications where coaxial cable or fiber optic cable can physically interconnect the LANs. They are not suitable for extended LAN applications of greater distances or where commercially supplied communications facilities must be procured for the internet circuit. (A 10 Mbps circuit would be cost prohibitive).

### Remote Bridges

Remote bridges are not limited by distance and connect LANs via synchronous communications circuits of speeds from 9.6 Kbps to 2.048 Mbps (or greater). Remote bridges are designed to extend the LAN beyond the distance limitation of 2800 meters (for Ethernet). Applications include connecting multiple offices or facilities too far apart for local bridges yet where synchronous communications facilities are available. Remote bridges can utilize telephone circuits for the LAN-to-LAN connections.

### Vitalink's TransLAN is a remote bridge.

Remote bridges operate much as local bridges. The primary difference is the interconnect circuit speed. They are generally compatible and may share in the same extended LAN. Remotely bridged networks, however, have the potential of congestion and delay due to the lower speeds of the wide area internet circuits. This could potentially affect the communicating internet protocols

as well as overall network performance. When using remote bridges, it is necessary to carefully analyze and plan the extended LAN.

Although so far this document has generically referenced bridges, both local and remote, the remainder of this document will focus on remote bridges, and specifically, TransLAN.

Comparing TransLAN to Other Forms of Communications Devices

Remote bridges are a new type of network building block. Although all their applications have not been fully discovered, TransLAN can replace, and has replaced, traditional communications products due to its special features and benefits. Some of these products are:

* Multiplexers, Port Selectors, and Data Switches TransLAN supports remote user populations via terminal servers and Ethernet. The traditional solution would have implemented a non-Ethernet solution based on a system of statistical multiplexers and port selectors (or data switches). In distributed data processing applications where multiple hosts may be connected via a set of multiplexed point-to-point circuits, TransLAN can alternatively interconnect the hosts through Ethernet connections at each site. Network potential points of failure are reduced, cost is lowered, performance is often enhanced, and the network architecture is structured for easier management and growth.

* Routers and Gateways - Routers and gateways are protocol dependent. The DEC router, for instance, supports only DECnet to DECnet connections. Bridge Communication's SNA Gateway supports only XNS to 3270 protocol connections. In contrast, TransLAN will allow multiple protocols to communicate across the extended LAN. TransLAN supports DECnet, TCP/IP, LAT, XNS, Chaosnet and all other internet protocols which operate on Ethernet or IEEE 802.3 LANs. With TransLAN, any two devices which can communicate over a local LAN connection, can communicate through the extended LAN. In environments where multiple LAN protocols must communicate with remote resources, the alternative would be multiple routers or gateways, and, therefore, multiple internets. In applications requiring high speed circuits and high performance in the internet such as CAD/CAM, scientific computing, and distributed data processing, TransLAN generally can provide higher network performance than using routers or gateways. This results in a greater internet throughput as well as more efficient utilization of communications resources such as modems, multiplexers, transmission circuits, and operational staff.

TransLAN and Transmission Media

TransLAN attaches to synchronous serial communications circuits through several standardized communications interfaces:

* RS-232 for speeds up to 19.2 Kbps
* V.35 for speeds greater than 19.2 Kbps
* RS-422 for speeds greater than 19.2 Kbps

Circuit speeds are supported for 4.8 Kbps to 2.048 Mbps, typically externally clocked. Telephone circuits (dedicated or switched), microwave circuits, fiber optic links, broadband circuits, and satellite circuits are supported. TransLAN typically connects to a modem, a Digital Service Unit (for AT&T Digital Services), or a time division multiplexer (providing a channel on a T-1 circuit).

TransLAN Software Features

The basic algorithm for a Data Link Layer bridge is filtering, forwarding, listening and learning. TransLAN can complete its basic function of routing datagrams with just those operations; however, Vitalink has implemented a series of special features to make TransLAN a versatile, general purpose communications processor. With these features, a network manager has extensive control and visibility throughout his network.

Security and Protected Networks

Generally, TransLAN filters traffic based on the destination address of a datagram. When TransLAN receives a packet from the LAN, it checks its FDSE tables against the Link Layer destination address and determines the location of the destination station. If the destination is on the local LAN, the packet is discarded. Using this filtering capability, Vitalink has made it possible to implement added security in a TransLAN extended LAN. Using Vitalink Management Services, a network manager can designate specific devices as "local only resources." This action consists of telling the local TransLAN system to discard all datagrams to/from a certain device by entering that device's Ethernet address in a configuration menu. This results in TransLAN implementing a source address filter for that specific device, effectively preventing any remote session.

Source address filtering prevents access to sensitive resources from remote stations in extended LAN environments. It places a network wall between secured devices and any attempt from off the local LAN for access. Using this feature, a network which interconnects various departments of a company can be configured so that each department shares only the desired resources with remote sites.

Protected network configurations are another way to add security and limit access to devices on an extended LAN. A protected network is one that is protected against all traffic from unknown destinations. A TransLAN bridge on a protected network will discard all traffic destined for any device which is not listed in its FDSE routing tables.

Protected networks are implemented by turning off the self-learning feature and manually entering, through management menus, the addresses of devices which can legally be accessed from remote stations. A network can be protected on a link-by-link basis, that is a TransLAN bridge can be configured such that all local devices can be accessed from one remote site (unprotected) and only a subset of the local resources are available to another (protected).

Protected networks can be used to control additions and changes to the extended LAN created with TransLAN. A network can be implemented without any protection, learn all existing stations, and then through management system entries, be protected from all "new" stations by turning off the learning algorithm and declaring all links protected. Any subsequent stations added remotely would be "unknown" and therefore their traffic would be discarded by TransLAN. Management procedures could be implemented to force new stations to register their access requirements before TransLAN allows wide area access.

## Multicast Filtering

LAN protocols typically use multicast traffic extensively for network management and control activities. Through multicast messages, Transport protocols learn the location of other stations and their availability, learn of server load requirements, and pass around routing information. Multicast messages are often "keep alive" messages declaring a station's availability to the network, and sent when a station is idle. On a LAN where there is a significant amount of bandwidth available (10 Mbps), multicast overhead is relatively insignificant. For wide area environments, Vitalink has added special features to control and limit the impact of multicast traffic in the extended LAN.

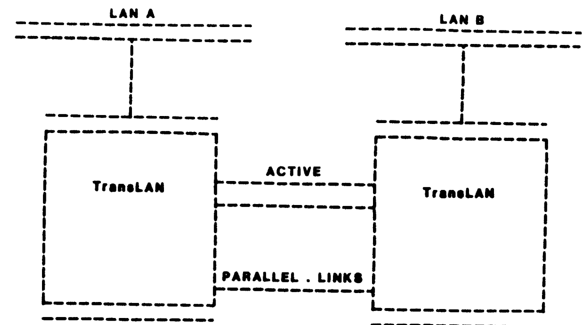
Under normal operating conditions, TransLAN forwards all multicast messages received out all transmit networks except the one associated with the receive network the frame originated on. As long as no congestion exists, multicasts are treated as broadcast messages. When the extended LAN becomes congested, TransLAN implements a congestion algorithm that discards multicast packets in order to handle known single destination traffic. It should be noted that the level of multicast traffic typically decreases as the level of single destination messages increases. Most stations do not send multicast traffic while actively communicating with other stations.

To further improve network performance, Vitalink has incorporated within TransLAN a multicast filtering capability on a link-by-link basis. The primary use for this feature is to limit unnecessary multicast traffic from the wide area circuits. Implementation is through management system menus for each circuit. An operator may declare that a certain multicast value should not be forwarded out a specific circuit. This might be used to prevent remote program loads from occurring when they could be handled locally, or to isolate a specific protocol's multicast from a network where that protocol does not operate.

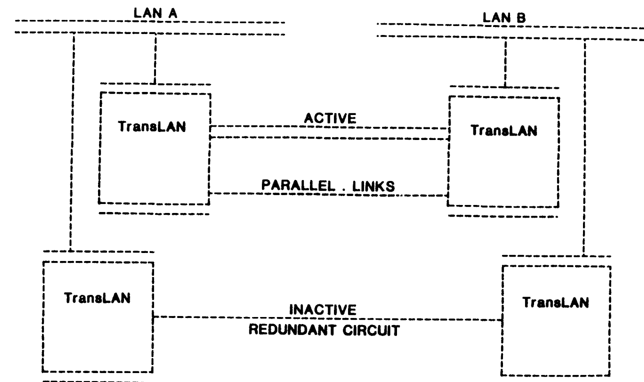
## Parallel Circuits and Redundancy

The correct choice of internet circuit speeds is essential for proper extended LAN operation. The internet circuit must be sized to support the total internet traffic between LANs. In most situations, bandwidth is available in increments of 9600 bps, 19.2 Kbps, 56 Kbps, or T-1 (1.544 Mbps) commercially available circuit speeds from the public telephone systems. Terrestrial TransLAN networks typically are connected with some combination of these services. TransLAN supports multiple circuits between two TransLAN systems which allows up to eight circuits between any two networks. This support is limited to circuits between two discrete TransLAN systems. The spanning tree topology prohibits multiple TransLAN systems interconnected and active between any two LANs. Using our loop detection algorithm, TransLAN will break any redundant path outside of the parallel circuit trunk and use it only upon failure of the primary set of circuits. See figures below.

Parallel Circuit Configuration




Redundant Configuration




Within the parallel circuit trunk, back-up is automatic. Should any of the parallel paths break (checked continuously by our network verification protocol), an alarm is issued and traffic is redistributed over the remaining operational circuits.

Routing through a parallel circuit trunk is by default on a load balancing basis. Traffic is automatically distributed between all circuits through load levelling. As an option, a network designer can allocate individual parallel circuits to a specific set of devices or protocols. A parallel link can be "protected," that is, it can be configured such that load levelling does not take place over that circuit. Instead, an operator can designate a set of packet types or a set of devices which can use the "protected link." Packet type is specified in bytes 13 and 14 of the Ethernet frame and usually specifies the internet protocol of the packet. Using this configuration option, a network designer can isolate types of traffic to individual circuits. For example, interactive terminal traffic can be routed to its own circuit(s) while all other traffic is load balanced over the remainder of the parallel circuit trunk. Or,

as another example, a circuit could be reserved for file transfer between a certain set of systems. Using these options, network managers can configure networks designed to give priority and guaranteed service to groups of users and applications. Defining a "virtual" protected parallel link which does not physically exist allows for effective filtering of traffic either by protocol or destination address.

Back-up within protected parallel links is also implemented. If a protected parallel link fails, its traffic is redistributed over all other similarly protected links (if available). If none are available and similarly protected, the traffic is redistributed over all other non-protected circuits. If all other circuits are protected, traffic is redirected over operational "protected" circuits. In all cases of parallel circuit recovery, FIFO (first in, first out) is protected, network alarms are generated to inform the network manager of the automatic reconfiguration, and the network is automatically restored to its initial configuration once the failure has been cleared.

Switched Circuits

TransLAN supports switched services within parallel circuit trunks or as stand-alone paths to remote sites. Currently, autodial support is implemented for AT&T's Switched 56 services. Switched circuits can be used to back-up production circuits, add bandwidth during peak traffic periods, or supply switched access to limited use remote sites.

Configuration of switched circuits is through management system menus. A switched circuit can be configured to be autodialed either by manual request, at a specific time or day, or on demand. Manual initiation of autodial actions require that the proper phone number be previously supplied to TransLAN. An operator may schedule a switched service by defining the dial time, phone number, and duration of the call. Or, TransLAN can be configured to automatically dial a remote location on demand when a certain destination address (known to be out the switched circuit) is presented for forwarding. This allows for a local host to initiate the autodial by attempting to connect to the switched remote resource.

## Vitalink Management Services and the Network Management Station

Functional Overview

Vitalink Management Services (VMS) is a subprocessing system resident in Vitalink Network Products such as TransLAN, TransSDLC, and TransLINK. VMS provides to the user a menu driven, interactive interface into link services that provide statistical and diagnostic functions for network management and control. Additionally, VMS allows for the dynamic configuration or reconfiguration of network variables and features.

Vitalink's Network Management Station is a member of the family of Vitalink Network Products that allows global access into VMS. The Network Management Station attaches to any network as a terminal server on an Ethernet. An operator using a terminal on the Network Management Station can connect to any local or remote VMS system located in any network processor. Through the Network Management Station, the operator can activate and monitor all of the VMS functions. The Network Management Station includes a printer for the hardcopy logging of network monitors, statistics, and alarms. Also included is a dial-in modem for off-site access to VMS services. The dial-in modem is particularly useful when remote diagnostic support is required.

Vitalink Management Services

VMS offers multiple services to the network planner, network manager, and technical control personnel. Through VMS, one is able to:

Dynamically monitor network activity at any network TransLAN location

Access statistics on Ethernet performance

Access statistics on internet communications links

Monitor and tune TransLAN Bridge performance

Configure or adjust network operating parameters

Detect and measure network bottlenecks and congestion

Isolate and correct network communications problems

VMS is both a dynamic troubleshooting tool as well as a network planning and measurement system. It is integrated into all Vitalink Network Products and provides much of the same information as independent diagnostic and monitoring systems costing tens of thousands of dollars by themselves. VMS is also the mechanism used for custom network configuration and implementation of special communications features such as security, switched circuit setup, protected networks, and parallel circuit trunks.

VMS is implemented with two basic modes: command/monitoring mode and a network management/configuration mode. Command monitoring mode features include:

A general network traffic monitor that displays network traffic statistics and performance

A multicast monitor that details specific multicast address communicating on the network

Commands to determine basic network configuration and status

Capability to test operational bridges and network management printers

Options to display bridge memory contents and network device addresses

An on-line help facility to aid network operations personnel

The network management/configuration mode is a menu

driven system that allows the user to:

- Detail specific network configurations
- Reconfigure network options and tuning variables
- Monitor communications links and Ethernet performance
- Initiate the rebooting of remote network bridges
- Gather statistical information for effective network planning and control
- Implement the special communications features available with Vitalink Network Products.
- Direct the output of VMS network monitors and statistics to printer(s) or serial devices attached to the network management stations

VMS menus detail link level statistics providing a unique view into a communications network. Applications of VMS span network management activities to dynamic network troubleshooting. These performance statistics include:

- Traffic counts at both the byte and frame level
- Communications link error counts such as:
  - CRC errors (Errored Transmission)
  - Abort errors (Synchronization errors)
  - DCD/CTS errors (Modem/CSU errors)
  - Receive overruns
  - Transmit underruns
  - Congestion statistics
- Ethernet error statistics for conditions such as:
  - Collisions and multi-collisions
  - Too long or too short packets
  - Alignment errors
  - CRC errors

Special communications features of Vitalink Network Products are invoked through VMS in its network management/configuration mode. Some of the features currently available include:

- Securing a local station from remote access
- Creating closed, secure networks
- Defining parallel circuit trunks and defining their use
- Defining and setting up switched services
- Controlling redundancy in the network configuration
- Implementing specialized filters to:
  - Enhance/tune network performance
  - Restrict specific vendor equipment/protocols to a local environment
  - Control broadcast and multicast inter-network traffic
- Define and configure network alarm paths and logging devices
- Implement password control for network management security

As with all VMS functions, the network management printer can be accessed for hard copy retention of statistical/configuration information.

Additionally, as a background function of VMS, a diagnostic subsystem continuously monitors many critical network parameters and reports any pertinent network alarms or warnings. The alarms can be selectively directed to Network Management CRT(s) or Network Management printers at desired network management stations throughout the network. These alarms include notification of any TransLAN failures, link failures, and network status information.

Network Management Station

Vitalink's Network Management Station is a hardware and software system that acts as a central access point for VMS funtions throughout TransLAN, TransLINK and TransSDLC networks. A single Network Management Station can access and control an entire network of Vitalink Network Products from a single location. Multiple Network Management Stations can be added to create true network-wide distributed control.

The Network Management Station consists of the following components:

- A four port Ethernet terminal server, Network Management Station Software, and required cables
- A CRT terminal and keyboard for interactive access
- A serial port line printer for hardcopy records
- A dial-up 1200 baud asynchronous modem for remote access

In TransLINK, TransSDLC, and other applications not necessarily requiring an Ethernet. the LAN attachment is to facilitate network management applications and for cascading multiple systems. For these applications, a Network Management attacment feature is available from Vitalink.

The Network Management Station is IEEE 802.3/Ethernet Version 2 compatible and can attach to any Ethernet in a Vitalink network using standard transceivers or Ethernet star couplers such as Digital Equipment Corporation's DELNI. Once installed, the management station logically "connects" through the production data channel to TransLAN bridges and VMS subsystems using an XNS network management protocol. The connection can be to the local bridge, that is, the bridge attached to the same Ethernet LAN, or to remote bridges across the internet communications link. A session can be switched as easily as a single keystroke. The multiple connection capability

allows technical control personnel to monitor both sides of a suspect communications link or logically trace network activity through the network. Once connected to a bridge using the Network Management Station, all the features of VMS are available to the user.

An important feature of the Network Management Station is its ability to accept dial-in access. Through the supplied dial-in modem and a customer supplied telco circuit, customer personnel or Vitalink support personnel can provide direct remote support in network diagnostic and troubleshooting environments. It is this feature of the Network Management Station that allows Vitalink to provide high quality network management support rapidly, even in remote areas.

## TransLAN Network Design Considerations

### Overview

Designing a communications network always has a unique set of issues which must be evaluated if the resultant network is to perform as expected. Bridges, including TransLAN, offer no exception to this rule. Although bridges are relatively easy to install, planning for installation is essential. The overall topology of the planned network must be evaluated. Concerns for traffic patterns and loads, potential congestion, and effects of delays on the various Transport protocols should be analyzed for both current requirements and projected future growth. Bridged networks appear to be as easy to change as the discrete LANs, so growth can occur rapidly, even uncontrollably. It is therefore essential to understand the applications, user populations, and traffic loads in order to plan the extended LAN environment prior to implementation.

### TransLAN and Other LAN Devices such as Hosts and Routers

TransLAN is a component of a network. Along with hosts, routers, gateways, and servers, bridges are one of the building blocks that will make up the networks of the future. As one pieces together a data network, different devices affect the use and functionality of other components. Bridges create extended LANs, that is, to Transport Protocols such as DECnet, TCP/IP, XNS and LAT, all stations appear to be "local" and do not require internet routing attention. Because of this, TransLAN can eliminate the need for internet routing within the extended LAN. It is therefore possible to designate DECnet hosts as end nodes, instead of routing nodes, where they were being used for routing traffic between the bridged LANs. This may result in additional host cycles for application processing that were previously used to routing activity.

Routers between bridged LANs are also affected by creating extended LANs with bridges. Since extended LANs appear as a single LAN to protocol sensitive routers and hosts, routing circuits between bridged LANs may not be active. The learning algorithms used by the Transport Protocols may determine that the least cost route to the destination is directly through the extended LAN and may not direct traffic to the internet router circuit. It is possible for a network manager to force the traffic flow through the router by adjusting network parameters (such as circuit cost), but this requires an action changing default internet protocol parameters. A limitation may exist in the number of routers that can exist in an extended LAN environment, as with DECnet, or the resultant extended LAN may need a change in the overall addressing scheme, as with XNS. XNS assigns cable numbers (or "network numbers") to each Ethernet cable and XNS routers get confused when they appear on multiple cables simultaneously (a router sysgen can solve the problem). With TCP/IP, subnet addresses might be implemented to differentiate between the connected LANs and simplify connection to other internetworks such as the ARPANET.

This is not to say that routers and bridges should not be used together in wide area networks. Routers can be used to interconnect extended LANs, act as an alternate path for traffic normally passing through bridges, or as a dedicated path for traffic that may interfere with nominal extended LAN performance. For example, a network manager may chose to implement a TransLAN parallel link or a router to isolate DECnet file transfer and prevent the potential surge in network traffic from affecting interactive users using TransLAN. Routers may also be used to connect remote resources not connected to the extended LAN or act as Network Layer barriers between extended LANs.

Since bridges such as TransLAN create a single, logical LAN, all protocol restrictions for a single LAN apply. For instance, DECnet requires that no single LAN have more than one designated router. This would apply to the extended LAN as well. Address conflict is also a potential issue. All devices must have a unique Network Layer address (DECnet Area and Node Number; TCP/IP network, subnet, and node; etc.) as well as a unique logical name (as in XNS Clearing House names). For instance, two hosts called VAX1 would cause a problem in an extended LAN, since both would be logically local to each other and users.

### TransLAN and Users

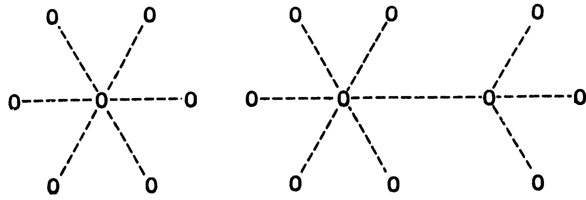
Users may expect that an extended LAN will operate exactly like a single Ethernet. All devices appear local and as if they were operating at the Ethernet rate of 10 Mbps. This offers both transparency as well as disguise to the users. For example, a LAT or DECnet user may connect directly to the desired host within the extended LAN rather than using the SET HOST command and DECnet routing through a local host. All the stations in the extended LAN appear to be local and can be handled as local resources, therefore the user interface is transparent to the location of the communicating stations. However, the interconnect link speed may affect response time and performance such that is may not match that of physically local devices. Although the users interface to all resources in the LAN is the same, performance to various resources may vary. Expectations must be appropriately set in order to insure user satisfaction.

### TransLAN and Network Topology

An extended LAN does not allow the creation of active loops in the network topology. This means that there is

a single active path between any two stations as reflected in a spanning tree topology. Bridges do not support an active alternate path. The following figure represents the possible topologies of bridged networks in terrestial environments.

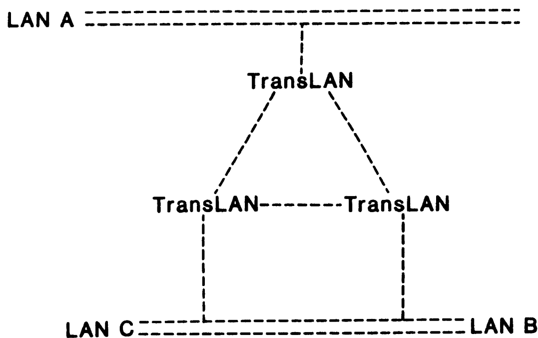
**Network Topologies Supported in Extended LANs**



Star Topology Multi-Hubbed Star Topology

In order to prevent loops, algorithms have been developed to break a loop and place the alternate path into a standby mode which will be automatically re-activated upon failure of an active link in the loop. See the following figure.

**Routing in Extended LANs**




In this configuration, one the links will be marked standby and inactive as determined by the loop detection algorithm. For TransLAN, it will be the lowest port on the last bridge activated. This implies, assuming the B to C link is inactive, that the traffic from stations on B to stations on C will route through A. Due to topology limitations of bridges, it is important to determine traffic patterns and direct connect sites to the resources most often accessed, thus eliminating as much pass-thru traffic as possible.

TransLAN and the DEC LAN Bridge 100 currently use different loop detection algorithms which means that it is possible today for loops to be automatically broken at improper places. Specifically, it is possible for a LAN Bridge 100 to detect a loop formed with TransLAN circuits and shut down its 10 Mbps local path rather than having TransLAN shut down its slower synchronous circuit. Using the guidelines presented later, this situation can be prevented. Vitalink is currently developing Digital's loop detection algorithm in TransLAN as a future option.

## TransLAN and Congestion

Synchronous circuit bridges such as TransLAN manage a bottleneck between two high speed LANs. Due to their use in remote environments using synchronous communications circuits, it is generally costly and impractical to interconnect the LANs at 10 Mbps. TransLAN is designed to manage the imposed bottleneck. TransLAN filters all local traffic, forwarding only internet traffic. Additional filters may be implemented to further reduce internet traffic. Congestion algorithms apply priority schemes to further enhance network performance. However, the potential for congestion exists, and the LAN-to-LAN internet circuit bandwidth should be sized large enough to handle all inter-LAN traffic in order to minimize internet congestion.

With most LAN protocols, sending stations learn of errors by timeouts on outstanding messages. As the level of timeouts increase, the Transport protocol adjusts its sending rate in response to the perceived congestion (indicated by the timeouts). It is therefore proper to characterize network congestion as temporary as long as the internet circuit(s) is sized too small for the traffic load or when the internet circuit is faulty, network performance can be dramatically affected. When building extended LANs using bridges, as with any network, care must be exercised in understanding network loads and sizing internet circuit data rates to insure optimum network performance.

## TransLAN and Transport Protocols

Although generally insensitive to protocol activity above the Data Link Layer, TransLAN can have some impact on internet and transport protocols. These protocols generally implement capabilities designed specifically for wide area network environments. These capabilities include:

* Congestion Reaction
* Software Checksums
* Large Sequence Number Space
* Error Control
* Selective Reject
* Pipelining
* Caching Out of Order Packets

Although DECnet, TCP/IP, XNS, and some other wide area Transport protocols for the most part implement these features, some protocols such as Digital's LAT were specifically designed for LAN Transport and does not support the above functions. An extended network constructed of TransLAN may exhibit congestion, delay, and loss of packets due to inherent transmission errors in serial circuits (Ethernet LANs have transmission errors, but are generally much cleaner than serial communications circuits such as phone lines and microwave). Because LAT does not implement the above features, the resulting performance of the extended LAN using protocols such as LAT may potentially be sub-optimal.

TransLAN is currently installed in many networks using LAT protocols and meets or exceeds the expected level of service to their users. These installed networks provide customers with a workable, cost-effective solution. In order to assure reliability in terminal server networks which use these "local" Transport protocols (such as LAT), Vitalink has placed special emphasis on implementing features for these environments. TransLAN III guarantees FIFO on all transmissions, limiting the pos-

sibility of out of sequence packets. TransLAN III supports end-to-end CRC preservation to insure data integrity without software checksums. Selective routing through the use of parallel circuit trunks can insure that terminal server traffic is not impacted by other pipelining file transfer protocols. Using TransLAN and terminal servers for remote user access offers customers an elegantnt solution that is cost-effective, manageable, reliable, and architected to easily accommodate future expansion and growth. With proper planning, any remote terminal server network is possible and is an effective network design alternative.

Appendix A includes user support information on pure LAT environments through TransLAN. The key to the model is understanding the user transaction profile. A transaction can be described as a two way operation: an input to the host and a response, measured in characters, over a given time. For example, the basic model describes the number of users supported if all users are 100% active inputting 25 characters and receiving 1000 characters every 20 seconds. The LAT Stat Gain factor given assumes the LAT server will send an average of 1.2 characters per packet (an assumption untested). The right hand side of the model indicates the various circuit speeds and the maximum number of users supported (both with and without LAT gain).

Variations of the model are included to indicate the effect of changing the various parameters. The model is most sensitive to user inputs as there is a potential 63 character overhead (minimum Ethernet packet size is 64 bytes) per byte of input for each direction of echoplex operations. For example, shortening the input from 25 to 10 characters almost doubles the potential user support whereas increasing it to 50 characters divides the user support by almost half. Changing the length of the transaction of the user loading (percent of the time users are actually active) affects the maximum number of supported users in a linear fashion.

The model examples are provided for general guidelines and to suggest an approach to network design. This emphasizes the importance of analyzing the extended LAN traffic profiles and user populations as foundational parameters in choosing the proper size internet circuit. Copies of this model, which runs on LOTUS 123, are available upon request.

It should be noted that terminal server networks are one possible solution to giving connectivity to remote users. Traditional solutions would call for a series of multiplexers, port selectors, and modems. Functionally an effective solution, terminal server networks bridged into data centers with TransLAN offer significant features and benefits. Using 802 standard technology allows the entire network to be managed under one set of tools and procedures. The self-learning feature of TransLAN and the Ethernet backbone allows for easy and quick network growth and change. The tools for managing the local area network are extended to the terminal server port, in the remote location, giving centralized visibility of the entire network. Less cables and EIA connections mean less potential points of failure (especially since these faults call for local, manual diagnosis and correction). Ethernet is generally a more efficient interface to a host requiring less computer cycles for communications. And, finally, TransLAN and terminal servers can be less expensive than multiplexer networks.

Generally, Transport Layer protocols designed for wide area networks, such as DECnet, are relatively unaffected by TransLAN. Most protocols run efficiently in existing TransLAN networks-- in most cases out-performing routers and gateways for internet communications. This does not mean that there are not concerns with the disguised wide area network that TransLAN hides with its extended LAN transparency. Retransmission rates are higher than on the LAN due to the bit error rates on the interconnect circuits, and transmission timeouts may need to be adjusted to accommodate the longer delays. Node names must be unique and the limitations of a single LAN network must be applied to the extended LAN. Once again, the emphasis is on proper network planning.

TransLAN and Communications Circuits

TransLAN utilizes synchronous communications circuits as paths between LANs. These circuits can be phone lines, microwave channels, fiber optic paths, broadband circuits, or satellite channels. All of these mediums have a higher bit error rate (between 10 E-05 to 10 E-09) than the LANs which they interconnect through TransLAN, therefore they are the "weak link" in the station-to-station path. These bit error rates translate into about one errored packet per thousand transmitted, although actual measurement in existing networks indicates closer to one in five thousand. When designing extended LANs using TransLAN, impact of these potentially error prone circuits should be taken into account.

Communications circuits are costly resources and should be used efficiently. In some respects, TransLAN can be inefficient in bandwidth utilization but in other areas TransLAN optimizes bandwidth in ways unavailable from alternative solutions. As described before, TransLAN uses passive backward learning; defaulting to broadcasting an unknown destination to all data links except the source of the frame. This results in some traffic needlessly propagated to LANs other than the true destination. Actual measurements in user networks have shown that TransLAN learns rapidly and that the unkown broadcast traffic represents less than .001% of the total traffic load. Alternatively, routers use bandwidth extensively to learn the network topology and discover network changes. However, in the case of routers, routing traffic can be a significant component of overall network traffic.

TransLAN also propagates all broadcast and multicast traffic to all remotes. This can be a significant component in total network traffic. TransLAN implements features designed to limit the impact of multicast and broadcast traffic. During congestion, TransLAN discards multicast traffic over single destination frames, effectively giving priority to point-to-point traffic when necessary. An additional set of features allow selective filtering of multicast frames on a port-by-port basis. This allows the filtering of unnecessary multi-

cast traffic from the internet. Use of these features can include filtering load requests and routing information when unnecessary, or isolating specific protocol multicast messages when the protocol is not supported on the remote network. Using these filters, non-essential multicast traffic can be controlled, and when coupled with TransLAN's congestion algorithm, multicast traffic impact can be minimized. In over 300 installed sites, Vitalink has not experienced a problem due to multicast loads where 9.6 Kbps circuits or greater have been used.

Due to its relatively high packet processing rate, TransLAN does optimize high speed circuit utilization. The typical router processing about 150 frames per second with an average frame size of 100 bytes has a maximum throughput of 120 Kbps (both inbound and outbound, therefore less than the bandwidth available from a 56 Kbps full duplex circuit). If the available or desired bandwidth exceeds 120 Kbps, the excess bandwidth is unused. In contrast, TransLAN can process about 1500 frames per second, which can efficiently utilize high speed circuits.

In multi-protocol networks, TransLAN logically multiplexes all internet traffic into a single circuit or parallel circuit trunk. In comparison, to utilizing discrete networks for each protocol, TransLAN provides enhanced bandwidth utilization for these environments. With TransLAN, a single circuit or group of circuits can be sized for all internet traffic. Since circuit size selection is limited and various traffic loads change dynamically, the multiplexing of all traffic into a single circuit or trunk offers a more cost-efficient solution than separate internetworks.

The most important part of network planning for bridged networks, as with most networks, is proper allocation of bandwidth. It is important to note that bridges interconnect networks of users and thus require adequate bandwidth to meet all the users expectations. The many features and options available with parallel circuit support, switched circuits, and priority/protocol routing make TransLAN a versatile and efficient solution in meeting this need; however, pre-planning is essential if expectations are to be met. It is our experience that consistency of service is critically important to the user's perception of the value of the network. A network planner should evaluate his current and future requirements with special consideration to the application supported and the users expectations of performance. The planning is complicated by the following:

* One rarely knows how much bandwidth is required. The tools simply are not available to precisely predict internet traffic loads until after TransLAN is installed and the network manager has access to the statistics a Data Link Layer bridge provides.

* TransLAN networks tend to open a new host of applications and functionality to a large number of users. Once the new functionality and connectivity is created, more users will want to use the remote access thus increasing internet traffic loads.

* Bandwidth availability and pricing tends to promote over-engineering. DDS circuits of 56 Kbps and T-1 are aggressively priced and often are the most cost efficient solution. For example, if the internet traffic requirements are estimated at 30 Kbps, the implementers most cost effective option is a 56 Kbps circuit, a 40% excess of the expected demand and a relatively safe network design decision. The real point is that you will always need more than you think you will unless you deliberately over engineer.

Fortunately, the versatility of TransLAN networks allows for relatively easy reconfiguration. As in most cases, the best medicine is prevention- evaluate the user/application requirements and plan parallel links or switched services in advance.

## TransLAN and Transmission Delay

TransLAN buffers all traffic and retransmits the frame, if necessary, out a synchronous circuit at the circuit speed. This injects a delay in delivery of some time inversely proportional to circuit speed. Assuming no queuing delay, this translates to about 15 ms. delay transmitting a 100 byte packet at 56 Kbps. Queuing adds additional delay as TransLAN may have several messages queued for transmission on the same data link simultaneously. It is therefore important to consider the delay implications on protocols using TransLAN for internet transport. In existing networks using terrestrial circuits, no change of protocol parameters have been required to compensate for delay. However, some transport protocols expect very little delay on a LAN (they do not see the extended LAN) and may require adjustment for optimum performance. The concern is compounded when multiple bridges must act as an intermediate relay for a message between communicating stations. Care must be taken to connect remote LANs directly to the most often used resources and limit multiple bridge paths to applications infrequently used (or for redundancy). With proper analysis and design of network topology, these concerns can generally be accommodated.

## TransLAN and Loop Detection Schemes

As noted before, TransLAN utilizes a different loop detection algorithm from the DEC LAN Bridge 100. Although the algorithms are compatible in most configurations, the guidelines suggested later in this document should be followed for networks incorporating both bridge types if proper network topology is to be maintained.

## TransLAN and Network Management

TransLAN's management system, Vitalink Management Services, offers a unique link layer window into network performance and status. Based on DEC's original bridge management specification, it is designed to readily adapt to management standards as they become available. In the absence of standards, Vitalink has structured an effective tool for network planning and operation. The other true bridge, DEC's LAN Bridge 100 uses RBMS, a VAX-based bridge management system. Both systems pro-

vide separate and extensive network performance statistics that are useful tools for network management and planning in the absence of a standardized management protocol. But the different tools are separate and discrete systems, each requiring its own set of procedures and training. Recognizing this as a potential problem when LAN Bridge 100s and TransLANs co-exist in the same network, Vitalink has committed to supporting RBMS in the near future. Until such time, the two management systems offer no more than an operational complication when both types of bridges exist in the extended LAN environment.

TransLAN and Data Integrity

TransLAN III preserves the original Ethernet CRC throughout the transmission from LAN station to LAN station. The CRC is checked by TransLAN when the packet is received and stored with the frame. A new CRC is regenerated when transmitted to another data link calculated to include the original CRC. When received from another TransLAN, the second CRC is checked and discarded and the entire frame, including the original CRC is forwarded to the destination LAN. For protocols which use software checksums such as DECnet, TCP/IP, and XNS, any problem resulting from not forwarding the original CRC would have been detected at the Transport Layer. However for local area transport protocols such as LAT which do not implement software checksums, this feature protects against all possibilities of data corruption. TransLAN II hardware cannot capture the original CRC, just check it, therefore the potential exists for undetected data corruption. Our experience in over 300 installations has not presented this problem; however, critical communications requirements should consider the potential and the solution offered by TransLAN III.

## Design Guidelines for TransLAN Networks

The following guidelines are recommended for installation of TransLAN networks. The guidelines are designed to be conservative and to be applicable to most environments. Variations from the guidelines should only be implemented after extensive network analysis and coordination with Vitalink.

The Use of TransLAN Should Be Accompanied by Careful Network Planning

Use of TransLAN should always be accompanied by careful network planning and analysis. This document should be read and understood before attempting to design or install a TransLAN network. Vitalink is available to work with its customers to design working, reliable networks that meet their expectations and requirements.

Use Local Bridges where Possible

Use local bridges such as the LAN Bridge 100 for remotes less than 1.5 Km whenever possible. The LAN Bridge 100 offers a transparent connection and should be used whenever a 10 Mbps circuit is available (Fiber Optics or Coaxial cable).

Careful Planning Should Insure that Adequate Link Speed is Provided for the Internet Circuit(s)

Typically, extended LANs with TransLAN will require a minimum of 56 Kbps for the internet circuits. A complete analysis of the internet traffic requirements should be completed prior to implementation. In some situations, slower speed links supporting, for instance, a single remote LAT server and a limited number of users, may be applicable. However, use of any internet circuit below 56 Kbps should be properly considered in light of the concerns detailed in this paper. As a general rule, over allocating bandwidth will insure that the network meets today's and tomorrow's requirements.

Minimize Multiple Bridge Paths

Each TransLAN in a station-to-station path adds delay and therefore increases the potential for affecting transport protocols in the extended LAN. As a guideline, TransLAN networks should be designed such that there are no more than two synchronous circuits between any two stations that require regular communications. Extending the LAN beyond two serial links should be the exception and should be carefully evaluated for delay considerations.

It is important to note, that this requirement applies specifically to a single extended LAN. Once a router is used as an interconnect, the extended LAN is logically terminated. This allows extensive interconnecting of multiple extended LANs via routers or gateways.

A single point of entry is required between TransLAN extended LANs and the LAN Bridge 100 extended LANs

This results from the two different spanning tree algorithms being used. A single point of entry allows TransLAN to perform its loop detection without assumptions regarding the LAN Bridge 100 and vice versa. If multiple points of entry do exist, the topology of the extended LAN becomes indeterministic.

Do NOT filter the multicast messages which are used by Digital's Spanning Tree Algorithm or Vitalink's Loop Detect Algorithm

The multicast messages are required for correct operation, regardless of which algorithm is used. For extended networks comprised of TransLAN and DEC LAN Bridge 100s, undetected loops can occur if the multicasts are filtered. It is understood that a single point of entry will prevent undetected loops from occurring, however a second level of security against undetected loops can be achieved by not filtering spanning tree multicast traffic.

Evaluate the applications and demands that will be placed on the extended LAN

In particular, analyze the impact of sharing the same synchronous circuit for both file transfer and terminal server interactive traffic. Over a 56 Kbps circuit, a relatively light file transfer load of maximum size packets can have a detrimental impact on the response time

for terminal server traffic. This could cause user dissatisfaction from the varying delay times on keyboard echos. Proper planning could include protected parallel links or switched services during peak periods of traffic.

Do not implement terminal server echoplex protocols directly over satellite circuits

TransLAN has a special configuration to support broadcast satellite circuits. When using satellite circuits, certain considerations need to be examined. There is a one-half second delay in echoing traffic through satellite circuits. This delay imposes an unacceptable user perception of the network. The use of more sophisticated protocols over satellite is acceptable. For instance, in DEC terminal server networks based on LAT protocols, using the SET HOST command allows CTERM to create a DECnet connection over the satellite link and thus manage the delay effectively. This requires a local host such as a MicroVAX to establish the connection and precludes remote terminal server only environments in satellite networks at this time.

Consider switched services in advance for back-up and peak period bandwidth

This powerful feature cannot only offer improved reliability to a production network but allows for misplanning in network bandwidth allocation.

Plan for verifying circuit performance

Sending synchronous data over a circuit is totally different than having a vendor-blessed "operational circuit." As pointed out previously, BERT testing and other traditional test sets are often useless in determining how a circuit performs when sending real data instead of bit patterns. A protocol analyzer is extremely useful for this purpose. The higher the line speed, the more likely that traditional test equipment will not be able to isolate the problem.

Have a dial telco line available during installation.

TransLAN support is based on TransLAN's capability to be remotely accessed via a dial link. Through standard telco ciruits, Vitalink can access installed TransLANs throughout the customer network and effectively diagnose and correct most problems. Without the dial circuit, Vitalink support is blind. As in most networks, support is most critical and frequent during installation. Proper preparation can eliminate many of the problems.

Guidelines for T-1 Circuit Connection

As discussed previously, terminating a T-1 circuit can be confusing. The following are suggested guidelines for ordering the correct equipment:

* Order a local line driver such as the Avanti 2300 if all of the following are true:

  1) The total circuit is less than 2 miles
  2) The circuit is end-to-end shielded metallic pair
  3) The circuit has end-to-end DC continuity

* Order a full feature T-1 clear channel unit or multiplexer if any of the above conditions are not true. The Avanti Accupac or Verilink VCC is recommended. The Avanti equipment can be purchased through Vitalink.

**Miscellaneous Information**

Complex TransLAN Networks

TransLAN is one type of network building block. It is useful in interconnecting LANs in various applications, however, it is not the solution to all networking problems. TransLAN can co-exist with repeaters, local bridges, routers, and gateways. Ultimately, large networks may be made up of LANs, extended LANs, and routing internetworks. Where multiple protocols are involved; performance is critical; routers are unavailable for the required protocol; or where various LANs serve closely related functions, TransLAN may be the best answer to the internet problem. However, extremely large, complex extended LANs are untested at this time. The management tools are not yet available nor are the Transport Layer Protocols written with bridges in mind. For the present, extended LANs should be designed according to the previous guidelines limiting the number bridge relays and setting a minimum circuit speed at 56 Kbps. Other configurations should be coordinated with Vitalink technical support before implementation. One should note that the guidelines do not limit the number of TransLAN bridges in a network. It is perfectly acceptable to connect hundreds of sites into a central hub or switching center. This simply uses a star topology and there is never more than two serial circuits between any two stations. A complex network might have many hubs and many redundant paths (loops) and should be designed in conjunction with Vitalink expertise.

Satellite Connections

Vitalink's primary business is data communications networking and two of their product lines offer satellite communications equipment. TransLAN, as well as supporting terrestrial applications, was designed with satellites in mind. Satellites are like an Ethernet in the sky-- a broadcast medium-- and what is broadcast to the satellite can be heard by anyone listening. TransLAN utilizes this medium by allowing full mesh networks to be created easily. An earth station transmitting a TransLAN output can be heard by many stations, where the local TransLAN listens to all traffic and filters out traffic not destined for its local LAN. The coupling of these two technologies, Link Layer bridges and satellite communications, offers users in geographically dispersed locations a cost effective communications alternative. Additionally, the filtering capabilities of TransLAN allows Vitalink to listen to all of its stations for diagnostic and station status information from its central Network Management Center in California. This management architecture provides a powerful, inline, and dynamic management system for centralized control of wide area networks. Where the distances between LANs exceed about 400 miles and the application is not exten-

sively delay sensitive, this kind of solution should be considered.

The Architecture Extended

Vitalink offers two companion products to TransLAN. TransLINK is a synchronous communications server which accepts bit synchronous (X.25, SDLC, HDLC) links as inputs. TransLINK buffers frames as they are input on the communications link, packetizes the frames into Ethernet frames, and forwards the frame to a software defined destination (another TransLINK port on a remote Ethernet). TransLINK is attached to the Ethernet and TransLAN captures the frame and forwards it to the proper remote LAN. The frame is then de-capsulated and retransmitted out the proper synchronous port on the destination TransLINK system. Thus, TransLINK is a software defined networking product that adds the capability of routing bit-synchronous point-to-point traffic through the same internet as Ethernet LAN-to-LAN traffic.

A second product is called TransSDLC. TransSDLC provides the same function for IBM 3270 devices as TransLINK does for X.25, HDLC, and SNA Host-to-Host traffic. TransSDLC packages 3270 SDLC PIUs into Ethernet frames and forwards them-- through TransLAN-- to a software defined destination. In addition, TransSDLC offloads polling from the wide area network by accepting host polls locally and locally polling the remote 3270 cluster. This results in improved use of bandwidth (polling is a heavy overhead), less delay in polling response (especially in satellite networks), and further integrates 3270 traffic into an extended LAN internet which may already be transporting LAN protocols, X.25 circuits, HDLC circuits, and SNA Host-to-Host traffic. All ports on both TransLINK and TransSDLC are software defined dynamically through Vitalink's Network Management System for both characteristics as well as circuit destination.

## Bibliography


Vitalink Communications Corporation, "TransLAN Installation and Reference Manual," Manual No. 005752, October 25, 1985.

Vitalink Communications Corporation, "TransLAN User's Guide," Manual No. 005744, March 28, 1985.

Vitalink Communications Corporation, "TransLAN 5.1, Installation and Reference Manual," Manual No. 011107P, April 25, 1986.

John H. Hart, "Bridges' Smooth Troubled Waters for Wide-Area Networking," DATA COMMUNICATIONS, March, 1985.

Digital Equipment Corporation, Intel Corporation, XEROX Corporation, "The Ethernet, Version 2.0," November, 1982.

M. Soha, Digital Equipment Corporation, Architecture-/Advanced Development, Distributed Systems; "Guidelines for TransLAN Bridges;" May 21, 1986.

B. Stewart and B. Hawe, "Local Area Network Applications;" TELECOMMUNICATIONS, Vol. 18, No. 9, September, 1984.

B. Hawe, et al., "Transparent Interconnection of Local Area Networks with Bridges;" Journal of Telecommunications, Vol. 3, No. 2, Summer, 1984.

B. Hawe, et al., "Local Area Network Connections," TELECOMMUNICATIONS, Vol. 18, No. 4, April, 1984.

B. Hawe and George Varghese, "Extended Local Area Network Management Principles," Digital Equipment Corporation, Technical Submission to IEEE 802 LAN Standards Committee, San Diego, CA, October 29, 1984.

B. Hawe, et al. "An Architecture for Transparently Interconnecting IEEE 802 LAN Networks," Digital Equipment Corporation, Technical Submission to IEEE 802 LAN Standards Committee, San Diego, CA, October 29, 1984.

R. Perlman, "Incorporation of Multiaccess Links into a Routing Protocol," Eighth Data Communications Symposium, MA, October, 1983.

R. Perlman, "Fault-Tolerant Broadcast of Routing Information," Computer Networks, Vol. 7, 1983.

R. Perlman, "An Algorithm for Distributed Computation of a Spanning Tree in an Extended LAN," Digital Equipment Corporation, Technical Submission to IEEE 802 LAN Standards Committee, San Diego, CA, October 29, 1984.

Digital Equipment Corporation, "XLII Bridge Architecural Specification," Version 1.8, March 1984.

Digital Equipment Corporation, "DNA Routing Layer Functional Specification," Version 2.0.0, Order No. AA-X435A-TK.

Digital Equipment Corporation, "LAN Bridge 100 Installation Guide," Order No. EK-DEBET-UG-001, December, 1985.

XEROX Corporation, Internet Transport Protocols, XSIS 028112, December, 1981.

Y. Dalal, Robert Printis, XEROX Corporation, 48-Bit Absolute Internet and Ethernet Host Numbers, OPD-T8101, July, 1981.

Institute of Electical and Electronic Engineers, Inc., IEEE Standards for Local Area Networks: ANSI/IEEE Std. 802.3, Carrier Sense Multiple Access, 1984.

Institute of Electical and Electronic Engineers, Inc., IEEE Standards for Local Area Networks: ANSI/IEEE Std. 802.2, Logical Link Control, 1984.

# MAP/OSI PROTOCOL PACKAGE FOR VAX COMPUTERS

Stan Froyd
Advanced Computer Communications
Santa Barbara, California

## ABSTRACT

The Manufacturing Automation Protocol (MAP) environment imposes a substantial communications traffic burden upon VAX computers acting as hosts within the manufacturing enterprise. The MAP environment and application demands are explored, and a high-performance, front-end architecture to meet these demands is described.

## INTRODUCTION

Efficient operation of a manufacturing enterprise involves timely processing of vast amounts of varied data. To date, this processing has been hindered by the inability of the computers and factory-floor control systems to exchange their local information with each other, due to a dearth of accepted standards permitting this communication. Recently, the combined efforts of manufacturing leadership companies, automation equipment vendors, and computer suppliers has led to a specification known as Manufacturing Automation Protocol (MAP). The widely hoped-for (and ambitious) goals of MAP are to permit information to be readily transferred between computers, people, and equipment; to be able to procure off-shelf equipment that "plugs in" to factory networks, and to readily port factory automation software application programs between dissimilar computers.

## THE MAP ENVIRONMENT

The MAP specification developers did not set out to write a new set of communications standards. Rather, the objective was to select from among existing standards a set of protocols that would be most appropriate to the manufacturing environment. The resulting specification adheres to the networking protocol structure defined by the International Standards Organization (ISO). This model for Open Systems Interconnect (OSI) is supported by most of the active national and international standards organizations. Within this model, it was necessary for the MAP specification to select from among various options provided. Where standards were not available for specific services (as in manufacturing messaging) new standards were created.

### MAP User Functionality

It must be noted that MAP is a *communications* protocol specification; it does **NOT** provide any operational functionality to the user. The user is required to develop (or otherwise procure) application software to actually operate his factory, using the application layer services defined at the upper layer of MAP. The current MAP specification supports a set of generalized services known as the Common Application Service Elements (CASE), and two specific application services, Manufacturing Messaging Service (MMS), and File Transfer, Access, and Management service (FTAM). MMS provides services to communicate with (and between) factory floor automation devices, while FTAM provides a mechanism for distributed file access among systems. As MAP continues to mature and evolve in response to automation demands, more application layer services (such as Virtual Terminal) will be added, and the functionality of today's services will be expanded.

### MAP Applications

The applications of MAP in the factory are limited only by the imagination of factory automation technologists. MAP is an enabling technology allowing computers to have instant access to the information and/or status of factory floor equipment, material handling systems, inventory and scheduling systems, etc. Armed with this information, computer systems are able to provide real-time control information, download part programs or control programs, coordinate dispatch of tooling and materials on a just-in-time basis, calculate and effect optimal production scheduling, and on and on.....

Among the immediate applications most frequently cited for MAP are data collection, production planning

and control, program upload/download, real-time control, distributed database management, quality control, and distributed numerical control. These applications are basically electronic implementations of current manual procedures. With a MAP system in place, the various functional organizations within a manufacturing enterprise will innovate programs and procedures which can multiply their effectiveness and permit the company to recognize bottom-line benefit.

Although this paper is emphasizing MAP, it must be noted that this is but a single set of applications atop the ISO protocol stack. Other non-manufacturing industries (e.g., banking) have similar objectives in terms of high-performance communications stressing vendor-independent interoperability and application portability. These objectives may well be realized by an appropriate set of different application services above the same ISO stack. One example of this is in the Technical Office Protocol (TOP) activity, where the services germane to an engineering office, such as electronic mail, document generation, and virtual terminal, are implemented using the same ISO model, but with different application and physical layers.

### MAP Physical Network

The physical implementation of MAP was chosen to be amenable to the manufacturing floor, where extreme atmospheric conditions, including temperature, humidity, and contamination co-reside with a high electrical noise environment. A factory is generally spread over a fairly significant floor space, and relocation of equipment is common. The chosen topology is that of a branching tree, with the root of the tree at a "head-end remodulator", (one required per MAP network -- usually in the computer room) and main trunks distributed through the factory, with drops at various computers, work centers, or subnetworks. Logically, of course, the various nodes on the network appear to have direct point-to-point connections with each other.

The signals are carried in RF over a broadband physical network built of CATV system components; these components are mature in production, can withstand harsh environments, and utilize cable rugged enough to be pulled through cableways. A MAP channel utilizes less than 10% of the usable bandwidth within the cable, with the remaining bandwidth available for other services such as surveillance, HVAC, etc.

## MAP NODE PERFORMANCE

There are three measures of performance relative to a MAP node: application layer functionality, data throughput, and the number of connections that can be supported. In the case of a general-purpose computer such as the DEC VAX, the application layer functionality is whatever programs the user implements atop MAP. Thus the overall effectiveness of the computer in the factory may be a strong function of the MAP communications throughput.

Realizing that the real underlying motivation of MAP is to acheive factory automation for the purposes of reducing economic order quantities, work in process, touch labor, and lead times, one must be aware that a VAX is very likely to be in the middle of quite a lot of factory data traffic related to individual parts, machines, tools, and events, all of which must be assimilated, coordinated, and responded to, while still providing necessary support to personnel and the needs of other nodes on the network.

As an example of the sort of data traffic that might be encountered in a MAP network, consider machine systems that consume (or generate) significant quantities of three-dimensional data related to the contours of a physical part, such as an airframe component, or automotive body die, or a shoe mold. Such apparatus has a steady appetite for data at around 500 characters per second per machine.

A simple gray-scale image transmitted to a VAX from a factory-floor camera for recognition of a part, or for determining its orientation, requires about a megabyte of data for a 1k-by-1k resolution picture. A graphic image sent to a graphic screen for operator instruction has similar size at the pixel level.

A screen-full of simple alphanumeric characters represents over 2 kilobytes of data.

In addition, there is likely to be a steady stream of smaller manufacturing messages reporting machine activity, tooling events, quality data, component inventory changes, etc.

And, of course, all of this is happening at the same time. In many cases, connections are left open to reduce the overhead (and traffic) of opening and closing connections for frequent exchanges of data.

## MAP IMPLEMENTATION FOR VAX

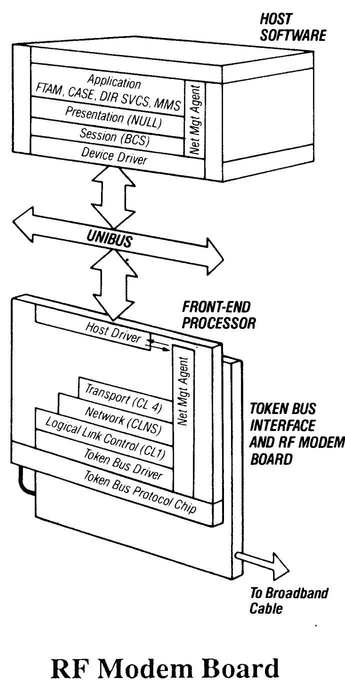
The current MAP specification provides (in Appendix 3) for an "Interim Network Interface Unit", permitting use of an existing, off-the-shelf modem to be used to connect a host computer to a MAP network using an RS-449 serial connection between the host computer

and the modem. This technique has been used in the public demonstrations thusfar, and has enabled initial implementations to proceed. For demanding applications, however, this approach limits performance in two ways: first, the host computer is burdened with protocol processing; and second, there is a data-rate bottleneck at the intermediate serial point between the host and the modem. Furthermore, the modem is an external device.

A VAX in a MAP environment can be utilized more effectively by being aumented with a communcations "front end" which processes part of the protocol stack, thus unburdening the host, while at the same time eliminating the serial bottleneck, raising the data throughput substantially. In addition, it is possible to implement such an approach so that the connection resides entirely in the VAX backplane, occupying only two slots.

A block diagram of such an architecture is shown in figure 1. The two boards are the front-end processor board and the modem board. These two boards implement the lower four layers of the ISO protocol stack, where the most CPU-intensive activities take place, while the upper layers (application, presentation, and session) remain in the VAX. Offloading the lower layers from the host returns processing time to the host, thus permitting the VAX to support more traffic or applications.

**Figure 1.** *System Block Diagram*




## RF Modem Board

The modem board serves to translate digital data to and from the Duo-binary AM-PSK radio-frequency signals utilized by the broadband network. It accomodates a 10 Mbps data rate within a 12 MHz bandwidth, transmitting in a band representing two standard television channels, and receiving in another pair of channels 192.25 MHz higher in frequency. All nodes transmit at the lower frequency, sending the signals up to the head-end remodulator at the root of the tree. The remodulator up-shifts the information to the higher frequency, and retransmits it down the tree to all of the nodes.

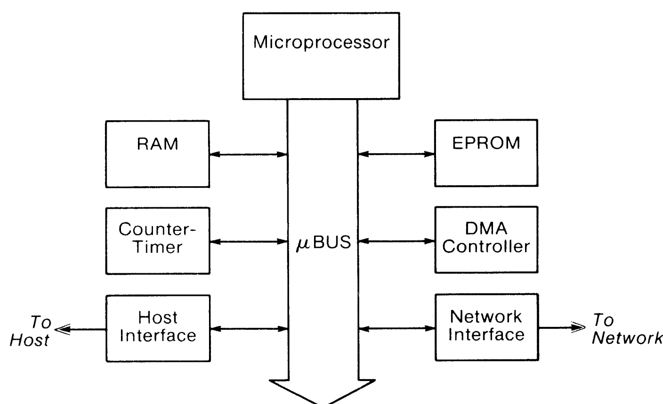
Of course, only one modem is permitted to be transmitting at a given time. Transmission time-slots are allocated to individual devices on the network by a token-passing protocol. The MAP specification specifies use of the IEEE 802.4 token-passing protocol, as it permits a very high utilization of the channel bandwidth, as well as deterministic availability for each node. The token bus logic is contained on the front-end processor, to be described momentarily.

The modem board includes scrambling and descrambling logic, for distributing the energy of the signal across the available bandwidth, and detection and correction of errors. It also includes a "jabber timeout", which causes it to disconnect from the network if any of its transmissions exceed 0.5 seconds. Such an extended transmission would indicate a failure within the transmitting node.

## Front-End Processor

Maximizing data throughput on the front-end processor requires an optimized architecture. In data communications applications where simultaneous connections are carrying traffic, it is clear that a parallel processing architecture can perform much more effectively than a serial, sequential one. The architecture of a "typical" microprocessor-based front-end is shown in figure 2. In this structure, processing is constrained to execute in a sequential manner, as the incoming data, outgoing

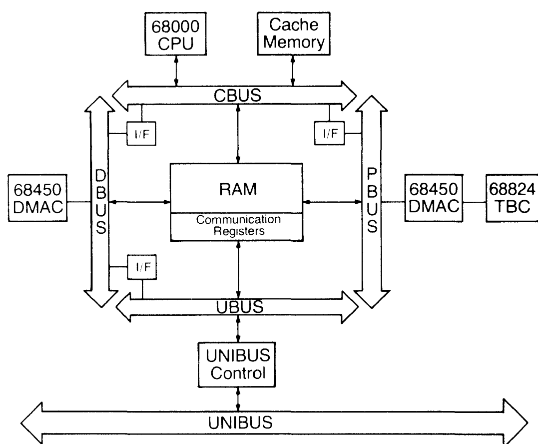
**Figure 2.** *Typical Front-end Processor*



data, and instruction and data fetches of the microprocessor all require possession of the microprocessor bus nearly exclusively to maintain a high throughput. As the microprocessor bus is clearly a singular resource, it is clear that there can be effectively no parallel processing of data.

A design that effectively eliminates this "bus bottleneck" problem is shown in figure 3. This architecture is the basis of the ACP 6000 series of communication front-end processors. Its central feature is a four-port RAM; each port is accessed by a separate segment of the 68000 bus. The segments, named the CBUS, DBUS, PBUS and UBUS, attach or detach in accord with the activity taking place.

**Figure 3.** *High-performance Architecture*




By supporting separate bus segments for the incoming data, outgoing data, and microprocessor data access, each segment is granted full bus bandwidth independent of the others. It is thus possible for each of the data paths to be running at a high bus bandwidth, and parallel processing can occur. In this configuration, it would appear that since the common RAM is a shared resource, we would again encounter resource contention. This is indeed the case, but as RAM arbitration can be executed substantially faster than bus arbitration, it permits each bus to operate at essentially full rate. To optimize the architecture still further, the "cache" memory can serve as both instruction and temporary memory for the microprocessor, so that shared RAM fetches need only be made to operate on protocol data.

To validate the approach, data has been recorded for a processor board utilizing this architecture, though with a serial communications chip used, rather than the token-bus controller chip used on the MAP board. This data provides insight into the performance of the four-way bus architecture. Table 1 reports the results of this testing. The front-end processor (ACP 6000) was doing complete HDLC protocol processing, while the host (VAX-785) was creating data to be transferred, requesting transfer by the ACP 6000, receiving and checking returned data, and logging a running total of (correct) received data. Rates are aggregate; both outgoing and incoming data are counted.

| Packet Size (Bytes) | Packet Rate (P/S) | Data Rate (kB/S) | CPU Load (%) |
|---|---|---|---|
| 256 | 195 | 50 | 38 |
| 1024 | 195 | 200 | 40 |
| 2048 | 128 | 260 | 30 |
| 4096 | 67 | 275 | 20 |

The product for which the data was recorded is able to support X.25 communications at line rates up to T1 (1.544 Mb/s). Reference [1] reports additional data for different line rates and CPUs.

The MAP front-end processor board will carry a larger protocol processing burden than the board for which these tests were run. As this processing is software-intensive, the burden will fall on the microprocessor, so that it is essential that parallel processing capability available in the four-bus architecture is be available.

The MAP front-end processor incorporates a 12 MHz 68000 microprocessor and a 1 MB DRAM to permit a large number of open connections. The Network, Transport, and Data-Link layers of the ISO protocols are handled by this microprocessor, so that the front-end performs packetizing and packet re-assembly, checksumming, and routing; in the event of faulty transmissions, detection and retransmission is all performed on the front-end, providing the host with guaranteed reliable data.

Also on the front-end processor is a VLSI component, a Token Bus Controller chip which performs the logical functions of the token bus system used in MAP. The token bus has no centralized controller node, thus requiring fully distributed logic. Each node must contain the logic for entering and reconfiguring the ring, detecting and reacting to network faults, and performing the token passing functions. This logic is performed in the token bus controller chip.

In the Technical Office Protocol (TOP) environment mentioned earlier, an 802.3 (Ethernet-like) physical

transmission protocol is used, rather than the 802.4 token-bus protocol. For these applications, the VLSI Token Bus Controller chip can be replaced with a Local-Area Network Controller-Ethernet (LANCE) chip, and the same performance benefits can be realized in those applications. This configuration obviously requires no modem board.

### Host Functions

The upper ISO layers are in the host VAX. These include the Session layer for managing individual sessions, the Presentation layer for translating from host-specific information representation to network standard representation (currently implemented as a null layer, requiring implementation agreements as to representation), and the application layer services visible to the programmers and maintainers of the system.

## CONCLUSIONS

Examination of the roles taken by VAX computers in MAP factory automation networks reveals a substantial communications burden potential which can reduce the computer's effectiveness. More processing power can be returned to the VAX by the addition of a front-end communications processor; substantial additional throughput improvement can be acheived through a front-end processor architecture incorporating concepts which permit protocol processing to occur concurrently with data transfer.

## REFERENCES


[1] Russ, Roger, "VAX Communications Controllers" *Proceedings DECUS USA,*
pp 423-423 (Fall, 1985).

[2] *Manufacturing Automation Protocol,* (MAP) Specification Version 2.1; General Motors Corp., Warren, MI (1986).

[3] *Technical and Office Protocols* (TOP) Specification Version 1.0; The Boeing Co., Seattle, WA (1985).

# UTILIZING THE VAXCLUSTER AS A NETWORK HUB

**John Dennis**
**Texas Instruments, Incorporated**
**Dallas, Texas**

## ABSTRACT

**VLSI** (Very Large Scale Integration) circuit design research and development requires judicious selection of general purpose and highly specialized computers. Proprietary interface systems as well as availability constraints must be considered when a data communication network is implemented to link the multi-vendor environment. Existing corporate IBM processing facilities also represents a major asset to the laboratory and efficient inter-computer access to this system is mandatory.

For Texas Instruments flagship circuit design research group, Semiconductor Process and Design Center's VLSI Design Laboratory (VDL), the goal was to create an environment capable of supporting circuit design, characterization, and testing, as well as developing advanced design automation techniques. To realize this concept, a series of diversified hardware centered around a VAXCluster was collected which reflected the flexibility and strengths required to accomplish engineering milestones. Four ethernet protocols (Calmanet, CHAOSnet, DECnet, and TCP/IP), an SNA Gateway, and various serial interfaces all connected to a VAXCluster-centered computing environment is described with emphasis on the operational ease of each networking product.

### Computer Aided Design (CAD) for VLSI

Texas Instruments circuit design flow incorporates a diverse collection of state-of-the-art CAD programs and hardware, each performing an integral part of our advanced VLSI design research. Massive amounts of CPU (Central Processing Unit) cycles are required in these VLSI design efforts for various types of modeling and simulation, from the very high-level abstract behavioral (functional) down to the extremely detailed transistor-level electrical descriptions. VLSI is generally defined to be the incorporation of between 100K and 1M devices on a single chip. Various stages of conceptualization must be integrated into dependable procedures facilitating top-down or bottom-up architectural expansion. Simulation, or dynamic design verification, is the process of evaluating the behavior of a circuit within the constraints defined by the engineer. The objective of simulation is to ensure the performance and proper functionality of a design before entering the costly fabrication cycle. Each type of simulation and analysis is critical to assure the success of a design on the first pass, succeeding passes are prohibitively expensive and time inefficient.

### Hardware Selection

Unfortunately, no one computer or single-vendor system adequately satisfies each specialized requirement for every stage of VLSI circuit design. Optimal performance for each phase is achieved by a different mix of resources: terminal I/O, disk I/O, CPU, printing, plotting, *etc.* Our solution, therefore, was to select several different systems, each capable of efficiently handling certain aspects of design flow.

### VAXCluster

The hub of our computing resource environment is a VAXCluster consisting of four DEC [1] VAX superminicomputers; one VAX-11/750, two VAX-11/780s, and one VAX-11/785. The VAXCluster was chosen for its maturity as an engineering and CAD tool throughout the industry and academia, the ease of expandibility, as well as the flexible, user-friendly VMS operating system. Many of our staff had extensive prior experience on VAX systems and are comfortable utilizing its features. Another powerful advantage of configuring a homogeneous VAXCluster is that all batch and printers queues are available from all the cluster nodes. The VAX-11/750 has the maximum 8 MegaBytes of main memory, the VAX-11/780 systems have 12 MegaBytes each, and the VAX-11/785 has 16 MegaBytes. Every VAX has its own 9-track tape drive for backup, archival, or transfer of data as a fail-safe method if the networks become inoperative. Hardcopy output is provided for the clustered machines from a DEC LN01 laser printer, a Talaris [2] 1200 laser printer, two 600 LPM Printronix line printers, and a Versatec [3] ECP-42 color plotter. Most specialized design hardware and software can be directly accessed from our VAX systems.

### VAXStation II

Three VAXStation II systems were purchased for their superior price/performance ratio. The VAXStation II systems are based on the MicroVAX II computer, have high-resolution monochrome graphics multiwindowing monitor and software, three RD53 71 MegaByte disk drives, an RK50 95 MegaByte streaming tape drive, 9 MegaBytes of main memory, LA50 graphics printers, and DECnet ethernet [4] for a stable com-

munication environment. One VAXStation II is being utilized for graphics application programming and software release testing. Another is used for EBEAM testing control and analysis, and the last one for device physics and characterization research. Small to medium SPICE jobs, parametric device yield estimation programs, VDL's office automation (e.g., local and worldwide electronic mail, corporate electronic news interface, desk calculator programs, word and text processing, meeting scheduler, notification and calendar functions, spreadsheet, etc.), project and personnel administration, staff attendance data collection and reporting, as well as a host of other organizational and design related tasks are executed locally on the VAXCluster and VAXStation II machines.

### CAD Workstations

Engineering workstations provide the designer interface for schematic capture, subcircuit compilation, logic diagramming, functional modeling, as well as some small SPICE simulation. A variety of small minicomputers and software packages were selected to facilitate the workstation concept. The TI designed and manufactured Designer Terminal (DT) is a hybrid built from a standard TI990 computer and a custom High Speed Display Processor subsystem. Our laboratory utilizes three DTs as well as an Engineering Production System (EPS), another TI990 with extra disk space and two 9-track tape drives, for a design concentrator and manipulation machine. From the EPS, designs can be translated from TI's proprietary database to Calma's GDSII stream format and placed on tape for transfer. The EPS and DT systems are connected to each other by XNS ethernet and to corporate IBM [5] mainframes by a proprietary 56Kbps network. TIPCs can also access the DTs and EPS system via XNS ethernet.

Currently, our lab utilizes nine Apollos [6], four DN600s and five DN660s, along with DSP80 and DSP160 server processor nodes. A typical Apollo node has 4 MegaBytes of main memory, a high resolution color monitor, 167 MegaByte Winchester disk drive, and an 8-plane color graphics processing system. Extra disk storage is provided by two 300 MegaByte removable disk drives connected to the DSP80, three 500 Megabyte Winchester disk drives connected to the DSP160. Tape archival is available through a 6250 bpi and two 1600 bpi 9-track tape drives. Hardcopy output is provided by an Imagen [7] Imprint-10 laser printer, a Versatec V-80 11 inch electrostatic printer/plotter, and a Versatec ECP-42 42 inch electrostatic color plotter. The Apollos are connected to each other by the Domain Ring network, to corporate IBM complex by a 56Kbps BDT, and to the VAXCluster via TCP/IP Ethernet. An alternate RS232 interface to the VAXCluster exists for redundancy.

Augmenting our advanced design automation project as well as our circuit design efforts, five Symbolics 3600-class LISP machines are also being used in VDL. They, along with seven TI Explorers, have been used to develop software emulators for circuits and will play an ever increasing role in future design environments with their AI capability. The Symbolics and TI Explorers use both TCP/IP and CHAOSnet ethernet for interface to each other and other computers. Future personal computer/workstations will need to have the powerful user interface that is standard on these machines.

After each subcircuit design has been completed, it must be verified, combined with other subcircuits previously defined, and plotted. To ensure this large global interconnect handling capability, provide local design rule verification and fast electrostatic color plotting, a Calma S-280 CAD system was chosen. The system has two 300 MegaByte removable disk drives, two color workstations with a 300 LPM Printronix printer-plotter attached, and two 9-track 800/1600 bpi tape drives. It provides a high-speed interactive graphics processor, physical layout graphics editor, symbolic layout editor, local Design Rule Checking, and host rasterization for the Versatec ECP-42 color plotter. The system is generally fed with GDSII stream data translated from the Apollo or DT output format. Calma developed a proprietary networking scheme known as CalmaNet that interfaces the Calma to the VAX and Apollo computers.

### VLSI Tester

The extremely large pin count of our parts necessitated the purchase of a state-of-the-art tester capable of supporting scores of independent drivers and 256 pin devices. High speed dynamic test and error capture as well as other detailed criteria went into defining the specification of a qualified VLSI tester. Selection of the UNIX 4.2 bsd based MegaTest MegaOne tester was made. MegaOne uses TCP/IP ethernet for networking between our Apollo nodes, VAXCluster, Symbolics, TI Explorers, and TIPCs.

### The Networks

Proprietary interface systems as well as availability constraints must be considered when a data communication network was implemented to link our the multi-vendor environment. Many systems had only their own proprietary network available and others had several suppliers that could provide a networking solution. Since we were relying on the VAXCluster as a hub of the computing environment, all systems had to interface to the cluster in a reliable fashion and have reasonable vendor support available.

### Computer Interconnect (CI)

All of the VAXCluster systems are connected to each other and to two central disk interface controllers (HSC50) via DEC's CI network. The HSC50s currently control fifteen RA81 456 MegaByte disk drives that can be accessed by all of the CI connected VAXes. The CI is a 70 Megabit per second clock network that can be used as a computer interface to disk and tape drives as well as carry DECnet traffic. All our VAXCluster nodes are interfaced with DECnet ethernet for user traffic, allowing the CI to be used exclusively for disk and tape I/O only, thus further increasing their overall performance.

### DECnet Ethernet

DECnet ethernet is the most heavily used protocol in our laboratory. It features high data transfer rates with all utilities normally associated with DECnet. DECnet has a powerful managment facility, Network Control Program (NCP), that allows troubleshooting and monitoring capabilities. DECnet utilities include virtual terminal access via the VMS command "SET HOST", transparent file transfer

via the VMS command COPY, and Digital Command Language (DCL) object execution (remote VMS node execution of DCL command files.) Invoking DECnet ethernet via commonly used VMS commands adds significantly to the ease with which users are able access this ethernet network.

### DECnet/SNA Gateway

56Kbps DECnet/SNA Gateway connection provides remote batch job entry (RJE), interactive IMS and TSO session, and application program interface access to Texas Instruments worldwide IBM complex. The IBM systems are locally used for large SPICE circuit simulation batch jobs, executing large-scale device characterization and physics programs, its powerful electronic mail system, creating circuit pattern generation tapes for reticles, as well as various other design and administrative functions. The gateway has a PDP-11/24 based DECSA networking system downloaded and managed from the VAXCluster via DECnet ethernet. The SNA Gateway is connected to the IBM complex by an IBM-3725 via a T1 multiplexer linked to an 18 GHz microwave system. Invoking the RJE application on SNA Gateway is easily accomplished via the VMS SUBMIT/SNA command. IMS and TSO are accessed by entering a SET HOST/SNA command. Again, DEC has integrated this networking tool seemlessly with VMS and communicating across the vendor barrier has become much less painful.

### TCP/IP Ethernet

TCP/IP ethernet is supported either by the manufacturer or a third party vendor for most computer systems. Each implementation of TCP/IP can provide different utility support, however, most have bi-directional ASCII and binary file transfer capabilities (File Transfer Protocol - FTP) and bi-directional TELNET virtual terminal access.

- Currently, we are using the Wollongong [8] TCP/IP ethernet shared-DEUNA package on the VAXCluster. A strong advantage of the Wollongong package is that it uses DEC's DEUNA or DEQNA ethernet interface card already resident in the DECnet ethernet VAX system rather than requiring an additional card slot.
- Apollo supports their own TCP/IP package.
- The Texas Instruments Explorer [9] LISP machine has TCP/IP supported by Texas Instruments.
- MegaTest's MegaOne [10] tester has a UNIX 4.2 bsd operating system that supports the TCP/IP ethernet package.
- Symbolics [11] Incorporated supports TCP/IP on their LISP machines.
- The Texas Instruments Professional Computer (TIPC) has the Fusion [12] MS-DOS uni-directional FTP and TELNET TCP/IP capability.

### Calmanet

Calmanet [13] is another ethernet protocol on the VAXCluster for interface to the Calma and Apollo machines. Calma Company sells and supports Calmanet on the Calma interactive color graphics workstations, VAXen, and Apollo workstations. Calmanet requires an additional card be installed in each system the software resides, on the VAX this takes one Unibus slot and 7.5 Amps of 5 Volt power. There are several Calmanet management utilities as well as the major file transfer utility, NFMOVE. There is currently no virtual terminal capability.

### CHAOSnet

CHAOSnet is an ethernet protocol sold by Symbolics Incorporated for use from their stand alone LISP machines. CHAOSnet is also supported by Symbolics on the VAXCluster and by Texas Instruments for the TI Explorer. TELNET virtual terminal access and CFTP file transfer utilities are available under CHAOSnet. CHAOSnet can also use the shared-DEUNA approach that lowers the Unibus card count necessary in the VAX.

### Strategic Support Considerations

VDL Computer Operations must make the most efficient use of our small staff to ensure timely implementation of crucial capabilities. Implementation and integration of industry standard programs and utilities is preferred to internal development efforts whenever possible. When off-the-shelf solutions are not available, however, as in the case of translating Apollo generated design data to GDSII stream format and visa-versa, we maintain the software expertise to quickly respond in these areas. VDL Computer Operations personnel work shifts are staggered to provide the highest degree of immediate software and hardware support possible.

Extensive backup methods and procedures have been established to provide minor and major disaster recovery. When network failures occur, 9-track tape drives, attached to most of the critical design engines, are capable of transferring data to other machines. In some cases, such as the multiple IBM interfaces, redundant links capable of providing effectively the same function are available. Redundancy for every large block in the design path is nearing reality and will decrease the possibility of a single point failure.

Dealing with multiple suppliers necessitated development of several steps to assure capitalization of the resources each has to offer. With VCO's diverse installed base of hardware and software, a monumental effort would be required to collect field service experts and an inventory sufficient to keep all systems functioning properly. Ours has become a management and coordination function with the major manpower being provided by the vendors themselves. Full service maintenance contracts are purchased to ensure the maximum machine availability. Figures indicate an average system availability in the high 90 percent range for the worst case month with several systems obtaining 100 percent uptime.

Meetings are held regularly with vendor representatives to discuss trends, new product information, failure issues and avenues for their resolution. A partnership atmosphere is fostered with to enhance our understanding of the product market and to increase our ability to judiciously plan equipment additions as our needs expand. Key national conferences, seminars, vendor sponsored classes, and trade shows are at-

tended for quality current awareness.

## Expansion and Future Trends

Major concerns, seen as bottlenecks in our operation, are for extended local processing power and increased transparency in the networking strategy. In the minicomputer price range, the market has been slow to produce single processors with large increases in performance. The general gain in single processor scalar machines has only been about 1.7 to 6 times that of a VAX-11/780. Latest trends in computing machinery have gone in five major directions. First, the parallel processor route which will allow multiple CPUs to be hung on a high-speed bus. Second, small vector machine development. Third, a loosely or tightly coupled multiple CPU clustering approach. The major difference between the two types of clusters is that the tightly coupled systems boot and fail together, loosely coupled systems generally boot and fail separately. Fourth, attached specialized array processors that generally run FORTRAN. And fifth, the multiple distributed processor or workstation strategy. There are advantages and disadvantages to each of these methods of gaining local processing power.

Currently, the most transparent networking scheme we have available revolves around DECnet. This is not likely to change in the near future due to the powerful management and integration tools DECnet possesses. Several vendors, such as Symbolics, are now adding DECnet to their systems. This network enhancement increases the multi-vendor computing synergism enormously and allows major productivity advantages.

To keep our operation world-class in its capability, a dynamic machine type and count adjustment will always be necessary. Certainly, there is room in our laboratory's computing environment for all of the five major directions. The maximum gain will most likely be felt as high powered (true VAX-11/780 performance) workstations with an AI flavor become affordable and available for every engineer. Application servers, residing as an ethernet host, may need to be developed that are tuned specifically for an application such as a multiple CPU SPICE engine, or for single language execution. This type of environment would allow computationally complex jobs to be transparently farmed out to these application servers and the designer's workstation will be free to handle general interactive interface as necessary. This strategy also lends itself to purchasing stand-alone, small, affordable vector machines that can be networked as servers.

Having centralized computing resources avails itself to support coordination, management of large data storage, backup and archival, and shared large expense items, such as tape drives. Advantages of increasing the VAX node count in a cluster include establishing a growth pattern that utilizes a stable system architecture, an improved price/performance ratio per node with such machines as the VAX 8800, compatibility with existing mass storage, and increased redundancy.

## Credits and Disclaimer

Mention of any company name or product in this paper does not in any way constitute an endorsement of that company or product by Texas Instruments, Incorporated or the author and is made entirely for informational purposes. The opinions expressed herein are those of the author and, although reasonable verification has been attempted, are not guaranteed to be factual.

[1] Digital Equipment Corporation, Maynard, Massachusetts. CI, DCL, DEC, DECnet, DECnet/SNA Gateway, DECSA, DEUNA, DEQNA, HSC50, LA50, LN01, MicroVAX II, NCP, RA81, RD53, RK50, Unibus, VAX, VAX-11/780, VAX-11/785, VAX-11/750, VAX 8800, VAX-Cluster, VAXStation II. and VMS are registered trademarks of Digital Equipment Corporation.

[2] Talaris Systems Incorporated, 7840 Herschel Avenue, La Jolla, California 92038 (619) 454-3363 Talaris is a registered trademark of Talaris Systems Incorporated.

[3] Versatec, a Xerox Company, 2710 Walsh Avenue, Santa Clara, California 95051 (408) 988-2800. Versatec V-80 and ECP-42 are registered trademarks of Veratec.

[4] Ethernet is a registered trademark of Xerox Corporation.

[5] International Business Machines Corporation, Poughkeepsie, NY, 12602. IBM, IBM-3780, IBM-3271, IBM-3277, IBM-3725, IMS, RJE, SNA and TSO are trademarks of International Business Machines Corporation.

[6] Apollo Computer Incorporated, 330 Billerica Road, Chelmsford, Massachusetts 01824 (617) 256-6600. Apollo and VACCESS are registered trademarks of Apollo Computer Incorporated.

[7] Imagen Corporation, 2660 Marine Way, Mountain View, California 94043 (415) 496-2100. Imagen and Imprint-10 are registered trademarks of Imagen Corporation.

[8] The Wollongong Group, Incorporated. 1129 San Antonio Road, Palo Alto, California 94303. (415) 962-7100.

[9] Texas Instruments, Incorporated. Dallas, Texas. Explorer, TIPC, and TI Professional Computer are registered trademarks of Texas Instruments, Incorporated.

[10] MegaTest Corporation, 1321 Ridder Park Drive, San Jose, California 95131 (408) 971-8378. MegaOne is a registered trademark of Megatest Corporation.

[11] Symbolics Incorporated, Four Cambridge Center, Cambridge, Massachusetts 02142 (617) 576-2600. Symbolics and CHAOSnet are registered trademarks of Symbolics Incorporated.

[12] Fusion is a trademark of the Network Research Corporation, 2380 North Rose Avenue, Oxnard, California 93030, (805)485-2700.

[13] Calma Company, 11022 Winners Circle, Los Alamitos, California 90720 (213) 594-8681. Calmanet is a trademark of Calma Company.

# OFFICE AUTOMATION SIG

# DEVELOPMENT OF AN IN-HOUSE TRAINING PROGRAM

Jennifer L. Rieck
USAA
San Antonio, Texas

## ABSTRACT

Working with electronic office systems requires new attitudes and skills. Training employees to accept the changes and to handle the new technologies is not easily accomplished. Resistance to technology, under-utilization and mishandling of office systems can prevent their full potential from being realized. In a large company, relying on the vendor to provide all the training is usually not cost effective. So, the development of a quality, in-house training program, tailored to meet the company's special needs, may be the answer.

In September of 1983, USAA formed it's own in-house training program for Office Automation. Two instructors were hired to determine training needs, develop a curriculum, and deliver training to a user population of nearly 1,000.

It took one year to train all users on the basic operation of their computers. Additional classes were then developed and offered which were more advanced in nature. Refresher courses, housecalls, and self-study programs were also added to the O.A. curriculum.

Office Automation soon became a way of life at the USAA Home Office building. It wasn't long before there was a need for us to be able to communicate electronically with our Regional Service offices which are located in:

Colorado Springs, Colorado
Sacramento, California
Tampa,Florida
Fairfax, Virginia

After hardware and software needs were determined, training at each location was provided. A complete O.A. network now exists among USAA's Home Office building and it's four Regional Service offices.

Since our inception, the number of users we serve and the number of classes we teach has grown considerably. With hardware and software constantly changing, the need for course revision and new course development is always present.

USAA is very proud of the high quality in-house training program we've developed. We've learned many lessons along the way and will, no doubt, learn many more as we try to keep up with the rapidly changing world of Office Automation.

# From User Documentation to Sharing Information: Problems and Solutions in User Communications

Daniel Barrett
Digital Equipment Corporation
Nashua, New Hampshire

**Abstract**

This paper discusses some of the problems associated with traditional, hardcopy documentation in the office marketplace. Because of the conflicting and sometimes unrealistic expectations that people have about documentation, there is no perfect solution, but rather a variety of possible solutions. One point, however, is clear: the less that a computer system depends on documentation to explain its workings, the better. In other words, the best computer system is one that has an intuitive and transparent human interface—something writers can and should help to design.

**Introduction**

Conventional documentation sets are something like the public school system in the United States: a great deal is expected of them, far more than they can possibly deliver. This may sound like heresy, but customers and other users have so many different demands to make of documentation that it is impossible to satisfy them all.

Instead of finding ways to write better books, we need to think more about sharing information, using a variety of ways to instruct people how to use a particular system. Since one of the best forms of learning is self-instruction, people should be encouraged to explore and discover a system for themselves, guided mostly by on-line information and using "documentation" only when necessary.

Before discussing solutions, let's review the problems that seem to be leading traditional documentation to a dead end.

## The Insider's View

These are the expectations on the "inside"—what writers hear from engineering groups as software is being developed and the documentation is being written.

### Documentation as Functional Specification

According to this line of thinking, the main purpose of documentation is to provide a detailed description of every facet of a computer system. For example, if a programmer has mapped five different keys so they perform the same function, all five keys should be documented more or less equally. If there are certain restrictions, no matter how obscure, that apply to a particular function, those restrictions must be recorded. In other words, documentation should state every single fact about a computer system, regardless of whether that information is of any practical value to the user.

In this way, computer documentation resembles the specs written for sophisticated types of machinery, everything from a 2-cycle engine to the space shuttle. And in a company that is heavily oriented toward engineering, where a great deal of this type of documentation is written, it's not surprising that this expectation exists.

However, "documentation as functional specification" misses the primary audience—people who want to use software quickly and efficiently, without needing or wanting to know every fact about the product. The situation is similar to the old joke about the boy who asked his father if he knew anything about Morocco. The father said, "Why don't you ask your teacher?" and the boy replied, "I don't want to know that much." Customers generally don't want to know that much either, just enough to get their work done as soon as possible.

The purpose of documentation is to make a system *usable*. But to include every fact about a computer system would make a typical doc set about twice as long and half as useful as it is now—and doc sets are already long enough. Completeness and thoroughness are noble goals, but only when qualified by reasonable expectations of the readers' needs. Eventually the law of diminishing returns sets in, and the reader is faced with information overload. The same rule should apply to both documentation and nutrition: Less Is More. Or as a friend put it: Enough is Enough!

### Documentation as Safety Net

Here's a typical scenario. Someone rushes into a writer's office and asks whether an obscure printing restriction is documented somewhere. We both look through the appropriate chapter, and maybe we find the relevant information, maybe we don't. Either way, as someone put it recently, "It's nice to be able to tell our customers that it was in the book; they just didn't look hard enough."

In my opinion, whether or not we're "covered," the documentation has failed. Assuming that the necessary information is there at all, it's worthless to a customer if we make it practically impossible to find. From the customer's perspective, the result is the same: delays, confusion, and frustration.

Along with the task of choosing what information should be documented, the writer also needs to consider where that information can be accessed

most logically and easily. Part of the answer is to use a greater variety of means to share that information—"solution messages," on-line Help, CBIs. But an even better answer is to design "documentation-free" systems which don't require time-consuming searches through weighty manuals.

### Documentation as a Selling Tool

Writers sometimes hear that "documentation sells products." I've never really understood what this means. When I use a computer, the last thing I want to do is start reading the instructions. I want to get some work done and discover this system for myself. And from what I've seen of customers, I think many people operate the same way.

The only example I've seen of marketing really using documentation to sell products is the well-known Macintosh TV commercial, in which a large set of binders is stacked up against the slender and far less intimidating user's guide for the Mac.

Whether or not documentation sells anything, there's a valuable lesson here: "The less documentation you offer, the more appealing it looks."

### Nobody Uses Documentation Anyway

This is a line that developers are particularly fond of using, and like most outrageous things they say, there's an element of truth lurking there. Namely, people don't *willingly* use documentation. It is usually a last resort, and often undertaken with no great hope of success.

Another factor is that most people are not particularly good readers. Many studies have shown that although an increasingly high percentage of people use computers in their daily work (as high as 70%, according to one report), many of these people have poor reading skills at best, and are illiterate at worst. In fact, we seem to have become largely a nation of viewers rather than a nation of readers. So people may be *unable* as well as unwilling to read the instructions we write.

The sooner that writers acknowledge the situation, the better chance we have of reaching our target audience and sharing information with them.

## The Customer's View

The problems I've just discussed with the present state of documentation—there's too much of it, the information is hard to find, users can't or won't read what we write—are reflected in the remarks of customers as well. Here are some comments I've heard:

- "Who has the time to read? We've got work to do here."
- "I just need to learn a few things to get going, and you've given me five binders."
- "I'm already an experienced word-processing user. Can't you just tell me the differences between your system and the one I'm familiar with?"
- "I'm 90% sure I won't find what I need, even before I start looking."

As convenient as it might be for writers to ignore these criticisms, we have to start being more creative in our solutions. And before we can evaluate the best methods for moving from hardcopy documentation to sharing information, we need to first consider how the systems themselves are designed, and how to ease the load that documentation must now bear.

## The Ultimate Goal: Zero Documentation

The most obvious solution to the problem is what I call "Zero Documentation"—similar to zero population growth, and with about as much chance of succeeding. The goal here is to produce systems that are so "transparent," so carefully designed and intuitive, that the user can perform most tasks without recourse to documentation.

This means a radical change in the way systems are typically designed, and a greater role for writers in designing the system, so in fact they become "support engineers," a term suggested in a recent article entitled "The New Wave of User Documentation." The author of this article, Edmond Weiss, makes the point that writers are among the most qualified people in designing a human interface, because they constantly deal

with the question of how best to convey information. This doesn't necessarily mean that they supplant programmers in developing the user interface, but that they work with them to build a truly user-friendly system.

Theoretically, this sounds highly desirable. At a practical level, we're finding that this is a difficult proposition that involves a great deal of give and take, since programmers, writers, and other interested parties all see the interface world quite differently. From the writer's viewpoint, one rule should be sacred: that system is best that requires the least external documentation. The system should be so well designed that users are able, and even encouraged, to learn it intuitively, without having an open book next to them.

## Toward Zero Documentation: Some Intermediate Goals

Zero documentation may be more of an ideal than a real possibility. But as we move toward developing more self-supportive systems, we can also improve the ways we share our information. Here are a few modest proposals about the ways traditional user documentation might be revised and supplemented.

### Getting Started

Probably every doc set needs some sort of primer, since there are still a lot of new computer users out there. Theoretically, the Getting Started guide should be full of practical lessons that introduce the user to components of the system, one by one. Practically, I wonder how many users actually have the time to go through these lessons before they get started with their own work, which is their main goal. So that we don't make the mistake of writing a very good book but missing our target audience, I recommend one of two solutions:

- Produce a book that can be read in an hour or less. Assume if it's longer than that, the user won't get back to it or maybe read it at all.
- Write a training manual, something that can actually be used in a classroom session for a group of people who are learning the new system. We know that sometimes a company will buy our documentation, then use it to compose a training manual for company personnel. Ideally, this shouldn't be necessary. By recognizing that many companies train people in classroom situations rather than individually, we can produce a document that would work well in this environment.

### Getting Started for Experienced Users

Since many people learning a new system are already experienced computer users, I propose a book that assumes a certain amount of knowledge, and proceeds from there.

The tricky part is that "certain amount of knowledge." Where do you draw the line? For PC ALL-IN-1, I think a useful book might have been a *Getting Started for ALL-IN-1 Users*, since many PC ALL-IN-1 users are also using ALL-IN-1 systems and want to know what the differences are, as well as what new features PC ALL-IN-1 offers. We've received exactly that request from customers, so we have to realize that Getting Started guides are not only for novice users.

### Applications

One common complaint about doc sets is, "You tell us how the system works, but not what we can *do* with it." Translated, I believe this means that people want to know how different parts of the system can be used together to get tasks accomplished most easily and efficiently.

Documentation tends to treat each part separately. Printing, filing, command procedures, editing—all tend to get their own chapters, with little crossover between them. An Applications book would demonstrate how to coordinate different parts of the system to solve typical but complex office problems.

No Applications book could be exhaustive, of course, but it could at least give users ideas that they could adapt to their own situations. Ideally, this book would be written "in shorthand," addressed to users already familiar with the system, to save both space and the reader's time.

### User's Guide

It would be nice to say that the dense ring binders will be a thing of the past, but the time when all documentation will be on-line still seems to lie in the future. On the one hand, the goal should be to document everything in the User's Guide that's worth documenting about the system. The other goal should be to leave the User's Guide on the shelf as much as possible (again, the best system is one that doesn't require constant reference to the doc set).

Since this is the sort of documentation that we've traditionally written, the only thing that bears repeating is the requirement for comprehensive and accurate indexes and tables of contents. Once readers have made the commitment to use the hardcopy documentation, our job is to get them in and out of the book as quickly and painlessly as possible, with the needed information in hand.

## How Can On-Line Assistance Help?

While we are trying to improve our hardcopy resources, writers can also do more toward providing better on-line assistance, so users can find necessary information without taking their eyes off the screen. Here is a survey of the possibilities and problems with current on-line facilities.

### Help

Anyone who has used VMS Help knows what a godsend this kind of on-line assistance can be. Help can give many people all they need to know about a menu option, key location, or form field, and can serve as both instruction to the novice and a reminder to the experienced.

Since it saves a user the fatigue and disappointment of searching through one binder after another, I am obviously an advocate of "the more Help, the better." Unfortunately, engineering groups don't see it that way, and are quick to point out that Help requires disk and memory resources that some systems don't have, especially PC-based systems. In those cases, Help tends to be cut to the bone, so it offers users only the most basic (and usually inadequate) information.

Perhaps as disk and memory restrictions continue to rise, this will become less of a problem. Whatever the situation, documentation groups need to be willing to fight for more Help screens, not because it makes our jobs easier (it doesn't, necessarily), but because it makes the user's job easier.

### Computer-Based Instructions

CBIs are a particularly good teaching tool, since they let the person learn by "hands-on" experience. Whether they are a valid alternative to documentation remains to be seen. My guess is that most people use CBIs as an introduction to the system (or certain features of it). But once they have gained a basic familiarity, they rarely go back to the CBIs, which take time to navigate through.

CBIs also share a similar problem with Help: they require substantial disk space, so they tend to be cut by Engineering even more readily than the Help screens. This is unfortunate, for it would be hard to improve on CBIs in their ability to help the user overcome that initial intimidation we all feel when we face a new computer system for the first time.

### On-Line Documentation

By this time, you can tell what I'm going to say: like Help and CBIs, on-line documentation takes up A LOT of disk space, and until this problem is overcome, it won't be a feasible alternative to the hardcopy doc set.

ALL-IN-1 has tried the experiment of customized documentation—not only putting the documentation on-line, but allowing customers to modify it and make it more suitable to their circumstances and requirements. As far as I know, the verdict is still out, but this seems to be a promising direction for the future. How far in the future remains to be seen.

### Error Messages

This is one of the most useful but neglected areas for making systems more self-supportive. The term itself ("error message") needs to be rethought. Most error messages are too defensive and so intent on telling users that they have indeed made an error that they forget to help the user around it.

What we need are "solution messages." Instead of informing users that they have made an "Invalid entry," why not display a list of the valid entries? When users press the wrong key to stop a print job, the message should indicate the right key. No matter how user-friendly a system is, people are going to make a lot of mistakes. But good solution messages can keep them going without the long but far from refreshing pause that a documentation search entails.

## Summary

A few conclusions and recommendations:

- Documentation is no answer to a poorly designed system. The more difficult a system is to use, the more documentation it requires; and the more it requires, the less chance the user will find the information he or she needs.
- The reverse is also true: if a system is well designed and "explains itself," the documentation can also be more streamlined and efficient, with fewer restrictions, caveats, and warnings.
- As we design a system, the goal should be to rely on documentation as little as possible. The more that people can learn and use our products without having to open a book, the happier they will be.
- Computer systems should invite people to explore them. Besides being more tolerant of user mistakes, they should direct users with clear and precise Help screens, CBIs, and solution messages.
- Writers should take a greater part in the design of human interface. They are (or should be) experts on communication, on what goes on between that computer screen and the user's eye and mind. No one is better qualified to write those Help screens and solution messages that are so crucial to a user-friendly, self-supporting system.

By working closely with software developers, we can perhaps move inch by inch toward systems that are ever more independent of documentation. Although we will probably never reach the "paperless office" (Zero Documentation) we can certainly advocate "less paper, more information" in the way we present computer systems to our customers.

# PERSONAL COMPUTER SIG

# Remote Operation of the DEC Rainbow Using MS-DOS

**Larry D. Scott**
New Mexico DEC PC LUG
Albuquerque, New Mexico

**Abstract**

In this short article, I will explain one way in which you can set up the DEC RAINBOW 100B so that it can be operated as a "dial-up HOST computer." (No I have not tested this on the 100A yet). The remote terminal, connected via the telephone line, can have full (with some reservations) control of all the RAINBOW capabilities, as if, the remote operator was sitting at the console keyboard. Another familiar form of remote access is that of a Bulletin Board System running a program like FIDO. I will not be discussing that mode of operation.

You may ask, "Why would you want to operate the RAINBOW in a dial-up host mode?" I have found it useful, for example, in cases where I want to have secure remote access to my computer at the office. I want to do more than just look at a data file. I may want to actually run a program to do a calculation, write a letter or manipulate a data base. To do this, you must have access to at least a modem and terminal for your remote system. I prefer a DEC VT-100 or VT-220 terminal, but these models or their emulators are not absolutely necessary.

To accomplish remote operation of your RAINBOW you must have attached to it, an auto-answer modem like the Hayes SmartModem 1200 or 2400 or the US Robotics Courier 2400. In addition, you need some specialized software to switch the computer to remote mode, condition the auxiliary port, place the modem in the proper auto-answer mode, and protect your system from unauthorized or accidental access with a password program.

All of this is currently accomplished on my 100B running MS-DOS 2.11 by using the US Robotics or Hayes modems and initially setting up the modems and the RAINBOW AUX port as indicated in tables 1 and 2 followed by running a batch file called HOST.

**Table 1: MODEM Preliminary Set-ups**

**A. Hayes 1200 baud Modem Switch Settings:**

| # | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| up | 0 | | | 0 | 0 | 0 | 0 | |
| down | | 0 | 0 | | | | | 0 |

**B. Hayes 2400 baud Modem Setup Commands:**
**(command sequence be processed once to set proper power-on defaults)**

```
AT&F      ;reset to factory default
           settings
ATS0=1    ;auto-answer
ATL1      ;low speaker volume
AT&D2     ;DTR response mode
AT&W      ;write above into EPROM as
           power up defaults
```

**C. US Robotics Courier 2400 Switch Settings:**

| # | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| off | 0 | | | 0 | 0 | 0 | 0 | | | 0 |
| on | | 0 | 0 | | | | | 0 | 0 | |

**Table 2: RAINBOW 'Set-Up' Configurations**

```
Screen #2:

AUTO XON/XOF
  set for 0 = OFF
MODEM STOP BITS
  set for 0 = 1
CHAR CODES
  set for 0 = DEC-8

Screen #3:

8N = DATA B/P
1200 or 2400 = XMT BAUD
1200 or 2400 = RCV BAUD
FDXA = PROTOCOL
(speed depends on your modems;
normally rcv and xmt are the same)
```

The HOST batch file contains the commands shown

[1]Larry D. Scott is with Sandia National Laboratories, Albuquerque, NM 87185

in table 3.

```
Table 3: HOST.BAT:

AUXRES      restore AUX port to
            'Set-Up' configuration
            in Non Volatile Memory

DTR_ON      set dtr on so MODEM.SET
            will be sent to HAYES &
            HAYES will answer phone

TYPE E:MODEM.SET > AUX:
            tells modem not to echo &
            not to send result codes
            ( For Hayes 1200 or
            Courier 2400 this file
            contains:
              ATE0,Q1
            for the Hayes 2400 it
            contains:
              AT E0 S0=1 Q1 &D2)

CTTY AUX    DOS command to transfer
            control to AUX

PASS        invokes password program
            (This is an 8 chr password
            program written is MASM with
            no sign-on or echo)

DDOG ON   (optional)
            invokes watch dog program
            to cause re-boot is phone
            line is dropped prematurely
```

Remote operation involves calling the HOST phone and when the two modems have acquired each other's tones, you would type in the 8 character password prescribed by PASS. The way PASS has been written, you will see absolutely no response from your host system, until you have successfully entered the password.

When the you want to log out, before you hang up, a batch file called BYE is run. After that, the host will hang up the phone on it's end and reconfigure for the next call. If you want to permanently return local control to your host system, then a batch file called LOCAL is run. These batch files are shown in tables 4 and 5 below:

```
Table 4:  BYE.BAT:

DDOG OFF  (optional)
                turn off DDOG

ECHO OFF        don't echo commands

DTR_OFF         hang up the phone

DTR_ON          enable modem for answer

PASS            run password program

DDOG ON   (optional)
                turn on DDOG after
                password is honored
```

```
Table 5:  LOCAL.BAT:

DDOG OFF  (optional)
                turn off DDOG

DTR_OFF         hang up phone

CTTY CON        transfer control back to
                console
```

Operation of your RAINBOW while in the host mode is essentially the same as sitting at the local console with a few exceptions. You will have problems, if you try to run a program or utility that directly accesses the video RAM to place information on the video monitor screen. Such programs will cause response to your remote inputs to appear on the local monitor, which of course you are unable to see. In extreme cases, you will completely lose control of and access to your system. At that point all you can do is hangup and hope your system will re-boot if you are using the optional DDOG program or manually do it when you get back to the office. It should be noted, that the system is very likely just as unaccessible to anyone else.

Below is a list of a few programs that have been tested for remote access:

- **Programs that Do Not Run** — SUPERCALC III, DBASE3, DED, FV, AME86 the CPM-86 EMULATOR, SETPORT or any program the accesses the video RAM directly
- **Programs that Do Run** — WORDSTAR, dBASE II, REDT, MICROSOFT FORTRAN, EDLIN or any program that limits it's I/O to standard DOS calls

If you are accessing your RAINBOW host from another RAINBOW, you can transfer any files, ASCII or binary, with error checking by using KERMIT and the following procedures:

- First have LCTERM or better yet, KERMIT running in the terminal mode (use CONNECT) at the remote site (your home).
- Next run KERMIT at the HOST site (your office).
- Then set the HOST KERMIT to SERVER mode and your local KERMIT to command mode by entering ctrl -]C. File transfers can then be accomplished with the following commands:

```
>GET FILE.EXT
    or
```

```
>SEND FILE.EXT
```

Note: Wild cards may be used!

- When you are finished with the file transfers, type FINISH to shut down the HOST server, then CONNECT to put you back into the terminal mode and finally EXIT to return to DOS on the HOST.
- At this point you can type BYE to terminate the session.
- If you are not familiar with the KERMIT commands you will have to study the KERMIT.DOC file.

Finally, we need to consider the remote terminal configurations for optimum operation of your dial-up system. Table 6 illustrates two such terminal configurations that have been found satisfactory.

```
Table 6:  REMOTE TERMINAL SETUPS

DEC VT220 SETUP

--- Display Set-Up ---
    Jump Scroll

--- Communications Set-Up ---
   Transmit=1200*  Receive=Transmit
   No XOFF     8 bits, no parity
   1 stop bit     No Local Echo
   EIA Port, Data Leads Only
   Disconnect, 2 s Delay
   Limited Transmit


Lear Siegler ADM-3A

auto new line off
1200* baud
8 bits
no parity
1 stop bit

* Note: The baud rates must be consistent
with the capability of your modem and
that established in your PC-100 Set-Up.
```

If you are using another RAINBOW for the remote terminal, then the AUX port setting should be set the same as for the HOST RAINBOW settings.

If you feel you need additional security, you can use the optional program shown above as part of the batch files HOST, BYE and LOCAL called DDOG by Dan Pleasant, Lower Falls Software to secure the host system in the event you lose the phone line and someone else dials in before you can re-establish the connection and secure it with BYE. DDOG provides complete protection, but at the inconvenience of having your system re-boot as if setup- ctrl-setup was invoked. For the PC-100B, one could set the system to auto-boot from the hard or floppy disks and have the autoexec.bat file do everything as usual, except that it invokes HOST without setting the date and time. The date and time could be set at the next time you called in. If you are not going to use DDOG, then be sure it doesn't appear in any of the batch files mentioned above.

In conclusion, I have been using this relatively simple system for about a year now and am not aware of any bugs at this time. I wrote the password program in assembly language and it is very easy to change the password by editing the source code and reassembling it with MASM and using LINK to produce the PASS.COM file invoked by the HOST batch file. You will note that the password program does not provide a sign-on screen when you dial into the system. This is done to enhance security by not providing any information to the caller.

Future enhancements might be to provide a program that would incorporate AUXRES and DTR-ON to setup the auxiliary port configuration that is now done interactively by the operator in 'Set-Up' Mode or perhaps it would be nice to have an auto-baud rate sensing program similar to LOGIN by Jay Jervey, Fountain Valley CA, to accommodate multiple baud rates for the remote caller.

AUXRES.ASM

```
title   auxres
;*** macro definitions *******************************************
;
dir_console_input       macro
                        mov     ah,07h
                        int     21h
                        endm
;
display                 macro   string
                        push    ax
                        push    dx
                        mov     dx,offset string
                        mov     ah,09h
                        int     21h
                        pop     dx
                        pop     ax
                        endm
;
get_time                macro
                        push    ax
                        mov     ah,2Ch
                        int     21h
                        pop     ax
                        endm
;
ms_dos                  macro   call_number
                        mov     ah,call_number
                        int     21h
                        endm
;
convert                 macro   value,base,destination
                        local   table,start
                        push    ax
                        push    bx
                        push    cx
                        jmp     start
table                   db      "0123456789ABCDEF"
start:                  mov     al,value
                        xor     ah,ah
                        xor     bx,bx
                        div     base
                        mov     bl,al
                        mov     al,cs:table[bx]
                        mov     destination,al
                        mov     bl,ah
                        mov     al,cs:table[bx]
```

```
                        mov     destination[1],al
                        pop     cx
                        pop     bx
                        pop     ax
                        endm
;
;*** equates *****************************************************
;equates
;
cr              equ     0Dh
lf              equ     0Ah
io_control      equ     44H
dos_return      equ     4CH
;
;*** data segment ************************************************
data                    segment
auxfun                  db      (?)
time                    db      "00:00:00   ",cr,lf,"$"
ten                     db      10
message_1               db      "RESTORING NVM AUX PORT VALUES!",
                        db      cr,lf,"$"
data                    ends
;
;*** stack segment ***********************************************
stack                   segment stack
                        dw      128 dup(?)
stack_top               label   word
stack                   ends
;
;*** code segment ************************************************
code                    segment
                        assume cs:code, ds:data,ss:stack
start:                  mov     ax,data
                        mov     ds,ax
                        mov     ax,stack
                        mov     sp,offset stack_top
                        display message_1
;
auxres:
                        push    bp
                        mov     al,2            ;IOCTL fancy stuff
                        mov     auxfun,1        ;function code (reset to NVM)
                        mov     bx,3            ;AUX device handle
                        mov     dx,offset auxfun;addr of IOCTL packet
                        ms_dos  io_control      ;do it
                        pop     bp
;
                        get_time
                        convert ch,ten,time
                        convert cl,ten,time[3]
                        convert dh,ten,time[6]
                        display time
                        ms_dos dos_return
code                    ends
                        end     start
;*****************************************************************
;last line of code
```

DTR_ON.ASM

```
title   dtr_on/off
;*** macro definitions ******************************************
;
outb                    macro   out_port,out_byte
                        push    ax
                        push    dx
                        mov     al,out_byte
                        mov     dx,out_port
                        out     dx,al
                        pop     dx
                        pop     ax
                        endm
;
ms_dos                  macro   call_number
                        mov     ah,call_number
                        int     21h
                        endm
;
;*** equates ****************************************************
;
dos_return      equ     4Ch
ccport          equ     02h     ;DEC Communication Control Port
dtron           equ     11111011b       ;bit pattern to turn on DTR
;
;*** data segment ***********************************************
data                    segment
data                    ends
;
;*** stack segment **********************************************
stack                   segment stack
                        dw      128 dup(?)
stack_top               label   word
stack                   ends
;
;*** code segment ***********************************************
code                    segment
                        assume cs:code, ds:data,ss:stack
;
start:
                        outb    ccport,dtron
;
                        ms_dos dos_return
code                    ends
                        end     start
```

```
;
;*** notes ****************************************************
; set DTR, RTS high on DEC 100b RS232 aux port      Bits for DTR
; --- DEC Communications Control Port ---------        on     off
; bit 0: Speed Select line, 0= low                     1       1
; bit 1: Sec Request To Send line, 0= asserted         1       1
; bit 2: Data Terminal Ready Line, "     "             0       1
; bit 3: Request To Send Line,     "     "             1       1
; bit 4: 0 will light LED                              1       1
; bit 5:    "          "                               1       1
; bit 6:    "          "                               1       1
; bit 7:    "          "                               1       1
;*************************************************************
;last line
```

DTR_OFF.ASM

```
title   dtr_on/off
;*** macro definitions ******************************************
;
outb                    macro   out_port,out_byte
                        push    ax
                        push    dx
                        mov     al,out_byte
                        mov     dx,out_port
                        out     dx,al
                        pop     dx
                        pop     ax
                        endm
;
ms_dos                  macro   call_number
                        mov     ah,call_number
                        int     21h
                        endm
;
;*** equates ****************************************************
;
dos_return      equ     4Ch
ccport          equ     02h         ;DEC Communication Control Port
dtroff          equ     11111111b   ;bit pattern to turn off DTR
;
;*** data segment ***********************************************
data                    segment
data                    ends
;
;*** stack segment **********************************************
stack                   segment stack
                        dw      128 dup(?)
stack_top               label   word
stack                   ends
;
;*** code segment ***********************************************
code                    segment
                        assume cs:code, ds:data,ss:stack
;
start:
                        outb    ccport,dtroff
;
                        ms_dos dos_return
code                    ends
                        end     start
;****************************************************************
;last line ---
```

PASS.ASM

```
title   pass
;*** macro definitions ******************************************
;
dir_console_input       macro
                        mov     ah,07h
                        int     21h
                        endm
;
display                 macro   string
                        push    ax
                        push    dx
                        mov     dx,offset string
                        mov     ah,09h
                        int     21h
                        pop     dx
                        pop     ax
                        endm
;
get_time                macro
                        push    ax
                        mov     ah,2Ch
                        int     21h
                        pop     ax
                        endm
;
ms_dos                  macro
                        mov     ah,4Ch
                        int     21h
                        endm
;
convert                 macro   value,base,destination
                        local   table,start
                        push    ax
                        push    bx
                        push    cx
                        jmp     start
table                   db      "0123456789ABCDEF"
start:                  mov     al,value
                        xor     ah,ah
                        xor     bx,bx
                        div     base
                        mov     bl,al
                        mov     al,cs:table[bx]
                        mov     destination,al
                        mov     bl,ah
                        mov     al,cs:table[bx]
                        mov     destination[1],al
                        pop     cx
                        pop     bx
                        pop     ax
                        endm
```

```
;*** equates ****************************************************
;equates
;
cr       equ     0Dh
lf       equ     0Ah
code_1   equ     "a"       ;first element of password
code_2   equ     "b"       ;you can use any character
code_3   equ     "c"       ;here for your password
code_4   equ     "d"       ;case is important!
code_5   equ     "e"       ;
code_6   equ     "f"       ;
code_7   equ     "g"       ;
code_8   equ     "h"       ;last element of password
;
;*** data segment ***********************************************
data                    segment
time                    db      "00:00:00   ",cr,lf,"$"
ten                     db      10
message_1               db      "PASSWORD INSTALLED, PLEASE DISCONNECT!",
                        db      cr,lf,"$"
message_2               db      "PLEASE INVOKE BYE WHEN FINISHED.",
                        db      cr,lf,"The time is ","$"
data                    ends
;
;*** stack segment **********************************************
stack                   segment stack
                        dw      128 dup(?)
stack_top               label   word
stack                   ends
;
;*** code segment ***********************************************
code                    segment
                        assume cs:code, ds:data,ss:stack
start:                  mov     ax,data
                        mov     ds,ax
                        mov     ax,stack
                        mov     sp,offset stack_top
                        display message_1
;
test:                   dir_console_input
                        cmp     al,code_1
                        jne     test
                        dir_console_input
                        cmp     al,code_2
                        jne     test
                        dir_console_input
                        cmp     al,code_3
                        jne     test
                        dir_console_input
                        cmp     al,code_4
                        jne     test
```

```
                        dir_console_input
                        cmp     al,code_5
                        jne     test
                        dir_console_input
                        cmp     al,code_6
                        jne     test
                        dir_console_input
                        cmp     al,code_7
                        jne     test
                        dir_console_input
                        cmp     al,code_8
                        jne     test

                        display message_2
                        get_time
                        convert ch,ten,time
                        convert cl,ten,time[3]
                        convert dh,ten,time[6]
                        display time
                        ms_dos
code                    ends
                        end     start
;*****************************************************************
;last line of code
```

ADVANCED PRO TOOLKIT

ROBERT ULESKI
BAKER INSTRUMENTS CORPORATION
P.O. Box 2168, Allentown, PA 18001

ABSTRACT

All of the examples herein are provided as a means to accomplish a certain task; there may be more efficient ways to accomplish your particular task.

This document is based on the current versions of P/OS and PRO/Tool Kit as of October 3, 1986. Please note that all references to unsupported procedures may not be available with future versions of the PRO/Tool Kit or the P/OS operating system.

Special thanks to Diane LoGuidice of the Atlanta Support Center and Steve Ducharma for providing me with this material.

1.0 V2.0 SPECIFIC INFORMATION:

1.1 DCL's use of Logical Names

- During DCL startup the following logicals are created: APPL$DIR: equates to DCL's application directory. This directory contains all of the DCL utility tasks. When the "$" character is specified in the RUN and INSTALL commands this is the device and directory which is used to locate the task images.

The user could supercede the logical name APPL$DIR to change the system directory for DCL.

IND$COMMANDLIBRARY: equated to "LB:[1,2 ]" This logical is used as the default device and directory by the Indirect Command file processor when no device or directory is specified in the command file specification. This logical can be used to change the library device/directory to another value.

1.2 Task Naming Conventions:

All DCL commands are processed by utility tasks which must be installed in the System Task Directory. For example, if the DIRECTORY command is issued DCL invokes the PIP utility task installed in the System Task Directory (STD) as ". . . PIP" and it runs with a task name of PIPTn where n is the terminal number. This is also true when the SPAWN command is used to start a command executing in the background. To abort the background operation you would specify ABORT XXX to abort the command where XXX is the three character name of the utility. For example if the SPAWN MACRO command is used and you decide to abort the assembly issue the command ABORT PMA. To determine what utility would be invoked for a specific command issue the following DCL command: SET DEBUG/FULL/EXECUTE. When commands are entered now, the first three characters of taskname will be listed along with the MCR command line to be passed to the utility.

Example:

```
$ SET DEBUG/FULL/EXECUTE
$ DIRECTORY
S109
PIP /LI
Directory DW1: [USERFILES]
" . . ."
$ SET NODEBUG
```

When the RUN command of the taskname entered is 6 characters or less DCL will attempt to invoke a task with the installed taskname as entered. If no task is installed with this name AND the taskname entered is 3 characters or less (1->3 chars) the DCL will attempt to invoke a task name of ". . . XXX" where XXX is the taskname entered. If this fails then DCL will attempt to perform an INSTALL/RUN/REMOVE using the taskname entered as the filename to install. The task is run with an installed task name of TTn where n is the terminal number. If this fails an error message of INSTALL -- File not found. will be displayed.

2.0 V3.0 SPECIFIC INFORMATION:

3.0 PACKAGING

Command Language DCL and the PRO/Tool Kit DCL task are now the same image.

HELP is optional now for both Command Language and the Tool Kit.

All application task images are now placed into the directory LB: [ZZPRODCL] instead of the application directory. This means that if you are running the Command Language application that all of the Tool Kit utilities are accessable to you. For example you could invoke the MACRO-11 assembler by entering the command line RUN $PMA, or you could use the DCL MACRO command if you first install MACRO-11 via the DCL command INSTALL $PMA. Additionally, this means that if both Command Language and the Tool Kit are installed

you could PURGE the directory LB: [ZZPRODCL] and recover disk space. Remember, before doing the purge to re-boot your system, perform the purge operation, and then re-boot again. If your system is a P/OS Server, remove the DECnet board so that no P/OS Workstations are using the Command Language or the Tool Kit application.

The application directory now only contains the application script file (INB file) and the per-user startup and exit command files. This allows each user to customize DCL for their own needs instead of effecting the application for all users.

The Tool Kit and the Command Language are shareable applications. This means that the components of the application are shared amoung all users of the application. (The per-user startup and exit command files and the INB files are exceptions) The PRO/ Tool Kit must be copied to the Public Library and then user's must perform an "Install from Library" to use the application.

4.0 NEW FUNCTIONALITY

4.1 Logical Name Support

Support is now included to create and maintain logicals in the USER, SESSION and SYSTEM logical name tables. When you create a logical it is placed by default in the SESSION logical name table. This is so that the logical will not be deleted when you exit the application. To specify a different logical name table use the qualifiers /USER or /SESSION.

Support is also included to create a concealed logical name. Concealed logical names allow you to access rooted areas on a hard disk. An example of a concealed logical is the logical FR001:. To display this logical, just type SHOW LOGICAL FR001:. The concealed equivalence will also be displayed. If a concealed logical name is created in the SYSTEM logical name table on a P/OS Workstation then access through the logical name to the root device is through the System Channel. This means that the protection UIC applied to file access will be [377,xxx] where xxx is a unique number per-workstation.

Support is now included to create logical names which include quoted strings and lower case characters. This allows the ability to create logical names that include a DECnet nodename and access string as the number of aliases is limited. Also you can now delete a logical name created by an application that is lower case.

5.0 THE PECKING ORDER ". . ."

There are two cases to consider when talking about the pecking order when translating logical names.

File access: If you pass a logical name to one of the file systems (either FCS or RMS) the USER logical name table is searched, if not found the SESSION logical name table is scanned, and lastly the SYSTEM table is scanned. In all three cases, all modifiers are allowable.

Using TLOG$ directive or the higher-level language PROLOG call: You can limit the search for a logical name either by table and/or by modifier. In other words I could limit the scan for a logical name to the USER logical name table and only those with a modifier of 2 (which is a LOGIN logical name).

(SEE FIGURE 1)

Logical names can now include all portions of a file specification. Both RMS and FCS will process logicals which contain all portions of a file specification. If your application is to backward compatible you must first translate a logical name which contains more than a device specification before it is passed to RMS or FCS.

6.0 NEW AND MODIFIED COMMANDS AND QUALIFIERS

- ABORT command has been replaced and now functions like the RSX ABORT command. This allows you to abort SPAWNed commands by their command name instead of trying to remember what task performs what DCL commands.

- CLEAR the clear command now only clears the screen w/o performing a reset-to-initial state. In previous versions it performed a reset-to-initial-state also. This is now accomplished with the /RESET qualifier. This was changed to preserve any user defined keys used through the communications package.

-DEBUG allows the ability to set the T bit in an executing task to aid in debugging. The specified task must have a debugging aid linked in.

- FIX allows the ability to load and lock a task into it's partition.

- FORMAT allows the ability to FORMAT a volume. (On hard disks a BAD BLOCK scan is automatically when the FORMAT command is used.)

- INITIALIZE allows the ability to initialize a volume and optionally specify a check point file size for the volume.

- LOAD allows developers to LOAD a device driver.

- UNLOAD allows developers to UNLOAD the code portion of a device driver, the data base remains in the system. Reboot the system to remove the data base.

- PHONE if the DECnet PHONE utility is installed on the system then the PHONE command will start up the PHONE utility.

- SET PROTECTION allows the ability to specify the default file protection to be specified on any files created during the current logged on session.

- SHOW PROTECTION displays the current default file protection to be applied to all files created during the current logged on session.

- CREATE/DIRECTORY allows the ability to specify the owner UIC of the directory file.

- DELETE/DIRECTORY allows the ability to delete an empty directory.

- MCR allows the ability to execute tasks from Indirect and pass commands directly to the task CA2. If the MCR command verb is 3 characters or less, DCL will attempt to first pass the command line to a task installed as the verb (XXX). If there is no task installed with this name then DCL will attempt to first pass the command line to the task "...XXX." If this fails, the command line is passed to the task CA2.

If the MCR command is 6 characters DCL will attempt to use the task installed as XXXXXX, if this fails the command is passed to the task CA2.

Example: $ MCR SET /VT100TI:

- MOUNT allows the ability to MOUNT a volume FILES-11 or FOREIGN. Optionally the volume name can be displayed (using the /SHOW qualifier).

- DISMOUNT allows the ability to DISMOUNT a mounted volume from DCL.

- PRINT File(s) can now be submitted to the default print. A default print queue must be defined before this command will complete successfully. (Use Print Services to define a default print queue.)

6.1 New Qualifiers to the File Commands

/SHARE allows others to view a file while you are accessing it.

/[NO]NEWVERSION enables control over creation of new file versions.

/[NO]PRESERVEDATE allows ability to control the creation date when copy files (PRESERVEDATE is the default).

/[NO]WANINGS disables the "No such file" error if the input file(s) don't exist.

/ALLOCATION:n allows the ability to specify the size in disk blocks for the file.

/OVERLAY specifies that input file overwrite whatever is currently in the file. The old file ID is preserved.

/OWN specifies that the output file contains the same file owner as the directory it is entered in.

Help files and documenation now include information on the DECnet qualifiers.

New Control Keys in the DCL single line editor (these are not documented).

- E brings the cursor to the end of a command line - H brings the cursor to the front of the command line.

New features in the Indirect Command File Processor.

- The symbol <NETNOD> now returns the DECnet node name of the system.

- The symbol <ACCOUN> now returns information available about the current logged on user. (Note that accounting is not available on P/OS hence there is no accounting information returned in <ACCOUN>).

- The symbol <PRIVIL> now returns whether or not the terminal is privileged.

- The symbols <LOGDEV> and <LOGUIC> return the current default device and UIC when Indirect was invoked.

- The symbol <UIC> is assigned the current UIC.

- The symbol <SYSID> returns the current P/OS baselevel.

- The symbol <TISPED> returns the same values as on RSX. If the symbol is used while running on TT1: then the value is 0.

- The .TESTSYSTEM directive includes support to return the values of new system feature symbols. These include HF$WS and HF$FS to determine if the system is a P/OS Workstation or P/OS Server.

Object Module Patch Utility is now provided. (PAT) PAT allows you to update, or patch, code in a relocatable binary object module.

Convert utility is now provided (CVT). CVT evaluates an arithmetic expression, converts that expression into the following formats, and then displays all the formats on your terminal.

| | |
|---|---|
| - decimal | identified by a period (.) |
| - hexidecimal | identified by the dollar sign ($) |
| - octal | identified by the expression oooooo |
| - RAD50 | identified by the percent sign (%) |
| - ASCII | identified by quotation marks (") |
| - Two Byte | decimal in the form XXX., YYY.<br>The maximum value is 255., 255.<br>octal in the form XXX, YYY<br>The maximum value is 377,377 |

Tool Kits now include tools to develop Synergy Application.

- A seperate diskette is included with the Tool Kits so that the tools necessary to develop Synergy Applications which can be optionally installed.

New manuals in the Tool Kit documentation include:

- IAS/RSX-11 System Library Routines Reference Manual
- Positional Device Interface Programmer's Manual
- PRO/ReGis Manual
- PRO/Document VDM Manual

6.2 PRO/DCL (Toolkit and Command Language) Logicals

DCL's Use of Logical Names

- During DCL startup it creates the following logical names: DCLAPPL$DIR: - equates to the value of APPL$DIR when DCL was invoked. This logical is useful to edit the per-user application startup and exit command files (start.cmd and exit.cmd) (created in the USER logical name table).

APPL$DIR: - equated to LB: [ZZPRODCL]

APPL$DIR (created in the SESSION logical name table as a LOGIN logical) Contrary to V2.0 if the logical APPL$DIR is created DCL will still use the login logical for APPL$DIR.

The APPL$DIR logical w/o the colon is present for compatibility with existing command files which perform a TRANSLATE indirect directive.

IND$COMMANDLIBRARY:

- equated to "LB:[1,2]" This logical is used as the default device and directory by the Indirect Command file processor when no device or directory is specified in the command file name. This logical can be used to change the library device/directory to another value.

DCL$EDITOR - equated to "EDT". This logical determines your default editor. It can be one of two values. Either "PROSE" or "EDT". If the logical doesn't exist then the default editor becomes PROSE. If the equivalence is not PROSE or EDT then the default editor also becomes PROSE.

PROLOD$MSG - equated to "1". This logical is used to control the display of error messages when using the LOAD and UNLOAD commands. If this logical is deleted only numeric values will be displayed on the terminal if an error is encountered during execution of the LOAD and UNLOAD commands.

DCL$DISABLE$EXIT - This logical is used to disable the DCL EXIT command on specified terminals. For example if the equivalence was set to "T1" then you could not exit DCL on terminal TT1: If you want to disable the exit command on multiple terminals use a comma "," to seperate the entries. The equivalence "T1, T3" disables the exit command in DCL when DCL is running on the terminals TT1: and TT3:.

6.3 Task Naming Conventions:

All DCL commands are processed by utility tasks which must be installed in the System Task Directory. For example, if the DIRECTORY command is issued DCL invokes the PIP utility task installed in the System Task Directory (STD) as ". . .PIP" and it runs with a task name of DIRTn where n is the terminal number.

Note that this is different from V2.0. This is so that if the SPAWN command is used for example to start an assemble with the MACRO command in the background you can abort the command w/o knowing what utility is processing the command.

The RUN command processing has not changed in P/OS V3.0 except that if an INSTALL/RUN/REMOVE is required it is performed in one operation through a call to the PROTSK system service. In V2.0 a seperate INSTALL, RUN, and REMOVE commands were generated to process the un-installed task.

6.4 Customization:

- Modifying the installation command file (INB file)

- The INB file could be modified to install tasks that you normally use during your DCL session.

- The length of time to enter the PRO/Tool Kit can be decreased by modifying the installation command file. To decrease the amount of time it takes to enter the application you could delete the FILE and MOUNT lines. Additionally, the /NOREMOVE qualifier could be placed on the INSTALL lines which currently do not contain the /NOREMOVE qualifier. This would eliminate the necessity to re-install the application tasks the second time it is chosen from the menu system as the tasks are already installed. The Task Control Blocks needed for these installed tasks are placed into secondary pool because they are prototype tasks. If secondary pool becomes depleted you could tune it's size by using the method described in the System Release Notes. The tasks that are not often (or never) used could be commented out and executed on as needed basis by using the RUN $ utility command. For example, the Source Language Processor (SLP) is installed for the EDIT/SLP command. If you do not use this command you could comment out the corresponding INSTALL line for SLP.

Whatever changes you make to the INB file, the original should be preserved and restored in the event that you want to REMOVE the application.

The INB file could be modified for all users of the DCL application or on a per-user basis.

1) The copy of the INB file that is copied to the DCL application directory for all users who perform an "Install" of the PRO/Tool Kit from the Application Library can be found in the device and directory accessable by the logical APL$NETWORK: When you are in the PRO/Tool Kit application.

The System Manager or anyone with a privileged account may modify the installation script file in APL$NETWORK:

If you modify this INB file, then any user who performs an "Install" from Environment Services will obtain this modified INB file.

2) The copy of the INB file that is processed when a user invokes the PRO/Tool Kit application can be found in the device and directory represented by the logical DCLAPPL$DIR: Each user may customize this INB file affecting on his DCL application.

- Startup and Exit command files: When DCL is invoked two startup command files are processed. Similarly, when you exit the application, two exit command files are executed.

- per-system: When the PRO/Tool Kit application is chosen from the menu system the file LB: [ZZPRODCL]START. CMD is processed by the Indirect Command fiel processor. This command file also invokes the per-user startup command file. When the PRO/Tool Kit is installed this file contains only commands that should not be deleted or modified. If an optional language (such as BASIC-PLUS-2) or the DECnet or Synergy development tools are installed the layered product may include statements to install components. These INSTALL statements could be placed into the installation script file to decrease the amount of application startup time. If this is done, the INSTALL lines should be placed in the order in which they are listed in the command file and after the current last INSTALL line to decrease the amount of time it takes to perform the directory look-up on the task images.

The System Manager or anyone with a privileged account may modify the startup or exit command files in LB: [ZZPRODCL].

- per-user: When the PRO/Tool Kit or Command Language application is installed onto a user's menu a copy of the installation script file and the START and EXIT command files are placed into the application directory (referenced by the logical DCLAPPL$DIR:). Each user may customize their START and EXIT command files.

6.5 Enabling DCL Fallthrough

You can enable DCL fallthrough of unrecognized commands by performing a simple ZAP on the DCL task image. With fallthrough enabled unrecognized commands will be processed in the same manner as the DCL MCR command. That is, if the MCR command verb is 3 characters or less DCL will attempt to first pass the command line to a task installed as the verb (XXX). If there is no task installed with this name then DCL will attempt to pass the command line to the task ". . . XXX." If this fails the command line is passed to the task CA2.

If the MCR command is 6 characters DCL will attempt to use the task installed as XXXXXX, if this fails the command is passed to the task CA2.

Example with fallthrough enabled:
$ DEV TI:
will display the current information for the device TI: on the terminal which was processed the task installed as ". . . CA2" (assuming that there is no task installed in the system named DEV or ". . . DEV").

To enable fallthrough perform the following:

```
$ RUN $ZAP
ZAP>LB:  [ZZPRODCL] PRODCL.TSK
Z:30062/ 000000
1
X
$
```

The exit the application and invoke it again. Fallthrough should now be enable.

- Uses of DCL Fallthrough:

With DCL fallthrough enabled, you could write a catchall task an install with a task name of ". . .CAZ". The task that is now installed as ". . .CAZ" (LB: [ZZPRODCL] CAZ.TSK) should be installed with some other name such as ". . . CA." DCL would pass any unrecognized command to your catchall task (assuming that there was not a task installed as xxx or ". . .xxx" where xxx is the recognized command verb). If your catchall task does not understand the command verb, the command line should be passed to the task CAZ.TSK with the new installed task name you used.

If this is done your catchall task should either catch out-of-band control-C's (described in the system reference manual in the terminal driver) or catch abort attempts and abort the CA2 task when either of the two cases occur.

Note: There is one restriction. If an unrecognized command is entered which is greater than 6 characters the command is passed to the task CA2. Additionally, if the command verb is greater than 3 but less than or equal to 6 characters then only a scan for the task name of xxxxxx will be performed. For example, if I had a utility task installed as ". . .SEA" and I entered the command "SEARCH" then the command would be passed to the task CA2 instead of ". . .SEA." This will be corrected in a future release. To work around this,enter "SEA" as the command verb.

6.6 Restrictions

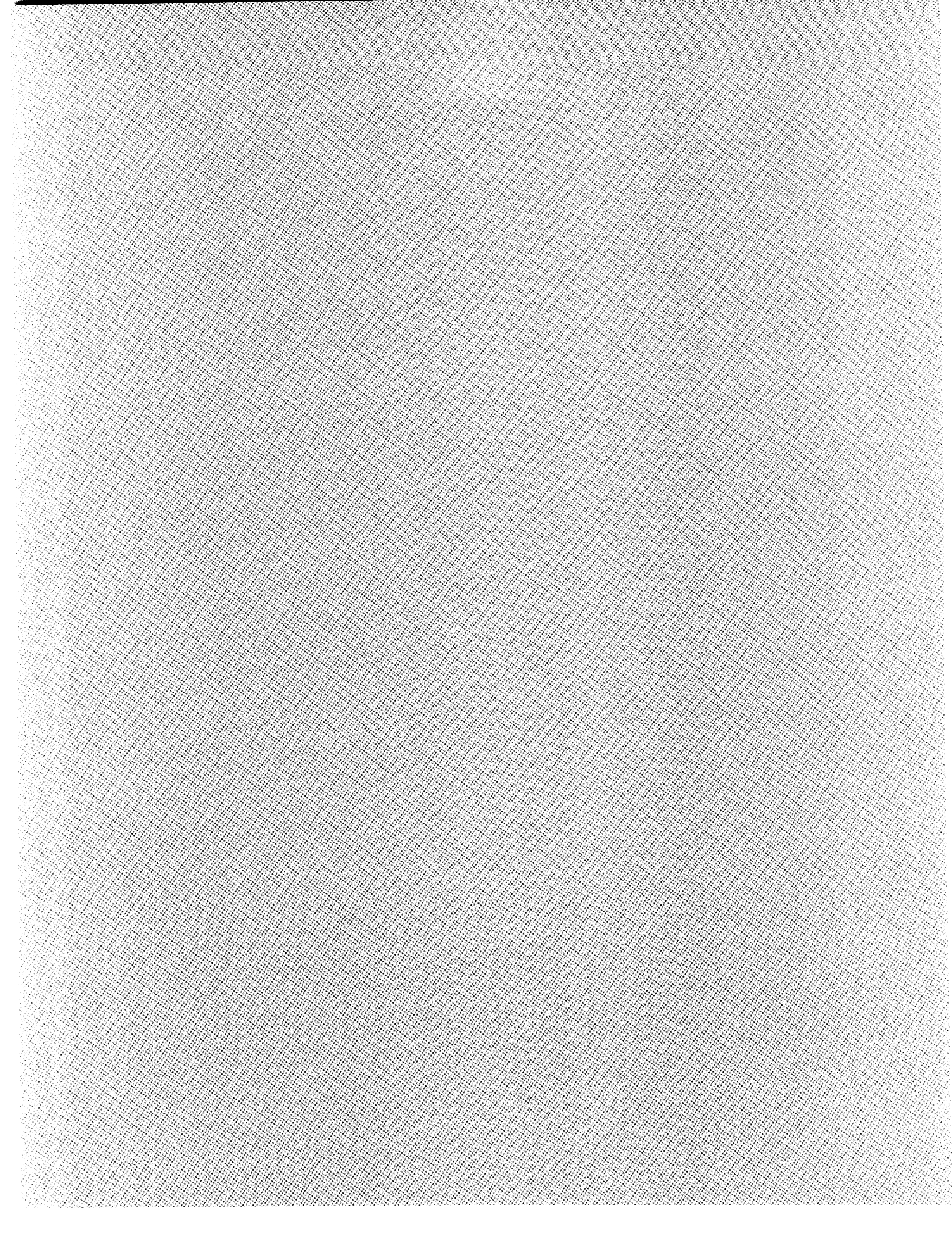
P/OS V3.0 now supports virtual terminals. The DCL task cannot be started on the virtual terminal as DCL will enter an infinite loop while trying to receive command lines from the virtual terminal. This is because the single line editor does not check to see if the functions necessary are supported on the current terminal. This will be corrected in a future release.

FIGURE 1

Logical Name Tables presented as an array:

```
                Table      Modifier
                  |  ---- modifiers 0 and 1 are equivalent
                  |          | ---- LOGIN logical name
                  |          |       | ---- specifies a concealed logical name
                  V          V       V
--------------------------------------------------------------------------------------
| USER      |  0 & 1  |  2  |  3  |  4->128.  reserved by DEC  |  129.->256.  appl use |
--------------------------------------------------------------------------------------
| SESSION   |  0 & 1  |  2  |  3  |  4->128.  reserved by DEC  |  129.->256.  appl use |
--------------------------------------------------------------------------------------
| SYSTEM    |  0 & 1  |  2  |  3  |  4->128.  reserved by DEC  |  129.->256.  appl use |
--------------------------------------------------------------------------------------
```

# RSX-11 SIG



# Lotus Blossoms Under RSX-11M+
# or
# How to cultivate a blooming nightmare

**Art Hurst**
3M Company Magnetic Media Division
Camarillo CA 93010

You are the proud system manager on a shiny new Micro PDP-11/73 running RSX-11M+ and supporting all the nifty utilities and support software including Datatrieve, DecNet, EDT, Runoff, PolyForth, Kermit, and all of the many other goodies supplied by enthusiastic DECcies. Your 4 Mb of main memory sports a blindingly fast virtual disk in its upper regions, and RMD's broad expanse gives you the feeling of having a universe at your command. A few of your TT: ports are dedicated to absorbing data from the ubiquitous "Tester with RS-232 output" to TESTER.DAT. You proudly display the equipment and pronounce the wonders of your magnificent system to your users. You explain how they may grab a copy of their tester input file, pass it through an analysis program, DataTrieve it for management reports, and do all this with the aid of the batch and indirect processors so that they have more time for coffee breaks. As you prepare for a thunderous applause of awe and respect for such a powerful and dynamic system that can save them from their daily drudgery, they respond with "That's neat! Looks like my IBM PC/XT set on end. Does it run MS-DOS or Lotus? By the way, how can I get a printout of that tester data file so I can key the data into my IBM PC Lotus spreadsheet."

After stuffing your wounded ego into your back pocket you respond with "Just say 'Print TESTER.DAT', then go to the printer and pick up your printout, but be sure to put the printer offline and do a local form-feed first and don't forget to put it back online afterwards". As you trudge back to your desk you mutter "Why would anyone want to go to all the trouble of copying a printout into a spreadsheet?" The answer of course is that the "User" is finally and at long last unchained from that damned Data Processing Department. The one that wants every request in triplicate, with cost justification before it can be scheduled into the FINO queue (First In Never Out). It is a New Dawn for the user. Through the kind generosity of Big Blue, Mr. User can now be his own DP department and schedule those jobs for this morning or afternoon whichever is more convenient. All he needs is a PC. But DP must authorize such purchases, DP "knows about all the PC's on site because purchasing reports all PC purchases".

He quickly submits several requisitions covering his newly required "tester" to purchasing for various "test cards", "power supplies", "chassis", "tester instructions", "tester color monitor", "tester hard disk", and "tester keyboard" which magically assemble into what most people recognize as an IBM PC/XT with all the bells and whistles. After loading his "tester instructions" and executing "LOTUS", he is off on the wondrous road to ultimate freedom and power. He is now courteous to you in order to be assured that you will diligently keep the paper supply in the system printer full so that he will never miss his morning printout, and of course you will keep a fresh ribbon installed so as to never cause him eye strain during Lotus keypunching.

Before you buy a cup, a gross of pencils and scan for a busy street corner, take heart! As the saying goes "When your basket runneth over with lemons, sell lemonade!". Examine the opposite side of the coin and you will see that your users are training themselves to help you! The reason you had an infinite queue is that you were spending all your time writing specs for an application and interviewing the user, and while all this was going on, the application was constantly changing. You could NEVER catch up! Now the user and analyst come in the same wrapper. He is writing and debugging the program for you, all you have to do is find a way to take his WICKET.WK1 and run it on your system.

But what's his advantage in this? He doesn't want to get locked in again after all those frustrating years of subservience to the DP masters. How about "Automatic Lotus", How about "Totally Automatic Lotus"?? Try that on your PC!! How about being able to install your user's debugged and working spreadsheet program into your system, have it access the TESTER.DAT file each day at 1:05 AM, process the data, and queue the Lotus printout for printing at 7:00AM. Now your user picks up a finished product from your system printer, AND, if he wishes to have program changes, he does it, not you! The changes are to the WICKET.WK1 file for which he has the responsibility. Suddenly you are able to support many more users than ever before.

"But" you say "how do I stuff an IBM PC on my bus??" Elementary my dear Watson! There is a company in Nashua NH that is dying to supply you with one. Fortunatly I had the foresight to spec one of these beauties into the original system before the Big Blue Bug had infected most of our technical troops. The onslaught was inevitable! The basic problem was to provide more than "just another PC". A time shared PC to a user who al-

ready has his very own is no incentive. Therein lies the tale of "A Lotus Blossoms Under RSX-11M+".

How does one get a Lotus to blossom under RSX-11M+? First you clear away some Q-Bus real estate, a dual-wide slot to be exact. Next you plant your QCP-11+ card, and distribute supplied software carefully as directed. Then via your A: disk drive you add your Lotus software and voila you can be Lotusing with or without floppy disks, from any terminal anywhere. It's no real trick to get that far, but now you have to start scheming!

Do you allow all users to access the same "disk C:" with the possibility of an unknown user deleting "COMMAND.COM" accidentally? No, so you set C: to Read Only, and then one of your software packages refuses to work because it must write to temporary files, and you can't redirect it. Do you feel surrounded by impossible choices Bunky? Suddenly bugles sound, hoofbeats are heard in the distance. It's Indirect to the rescue!

Imagine that you could lock up C: as Read Only and limit it's use to system functions, then create a shared scratch disk D: with Read/Write access. DOSFLX which acts like our old friend FLX lets RSX magically create the disk of your choice, for instance DISKD.DOS with all the fancy file support you're used to. Calm reigns as your users busily play with EDLIN and other such trivia on the lowly public disk. But of course after three hours of satisfied complacency, your first call for a private disk will arrive. Now what? Not to worry. Now you start the fancy footwork, you create MYDISK.DOS in the user's account and via indirect in [3,54] you create IBM.CMD which examines the account requesting access to the PC, determines if he has his very own disk MYDISK.DOS and links him to it as D:. If he isn't sufficiently aggressive enough to badger you into allocating disk space to him, he gets the public scratch disk DISKD.DOS. Want to see [3,54]IBM.CMD?

```
.enable quiet
.enable substitution
.enable global symbol
.testfile '<uic>'mydisk.dos
.if <filerr> eq 1 .goto 20
.goto 10
.20:
CRD IBMPC.LST =[7,46]diskc/R,
  '<uic>'mydisk/I/B/M:77./N:0./V:7452.
.goto exit
.10:
CRD IBMPC.LST =[7,46]diskc/R,
  [7,47]diskd/I/B/M:77./N:0./V:7452.
.exit:
```

Let's now look at AUTOEXEC.BAT on DISKC.DOS.

```
echo off
cls
path c:\;d:\;e:\;f:\
prompt $p$g
if not exist d:init.bat goto end
d:
init
end:
```

This little jewel will keep everyone's paws off of C: by switching them to D:. As an added plus, If D: happens to be an applications disk such as LOTUS.DOS, you can be more careful and install an INIT.BAT that will protect things. First, let's see how we would bring up Lotus. The following is [7,47]LOTUS20.CMD

```
.enable quiet
.enable substitution
.enable global symbol
.testfile '<uic>'mydisk.dos
.if <filerr> eq 1 .goto 20
.goto 10
.20:
CRD IBMPC.LST =[7,46]diskc/R,
 [7,47]lotus20/R,'<uic>'mydisk/I/B/V:7452.
.goto exit
.10:
CRD IBMPC.LST =[7,46]diskc/R,
 [7,47]lotus20/R,diskd/I/B/V:7452.
.exit:
```

As you can see, this expands on our IBM.CMD by bringing up Lotus pointing to either a personal disk or the public scratch disk depending on the user's account with Lotus Read Only for protection. Additionally you will note that only DISKD.DOS or MYDISK.DOS are accessible for Write thus protecting you from having a corrupted LOTUS.DOS. Of interest as well would be INIT.BAT which will be executed from AUTOEXEC.BAT.

```
cd\123
crd123v2 >nul
123
cd\
bye
```

Now that we are able to use Lotus, what shall we do with it? Funny you should ask. As you remember earlier, we were sucking up gobs of data from the Wicket Tester MK II via the "Standard RS-232" link into TT5: and ultimately into TESTER.DAT. Now suppose for a moment that the data coming in represents five readings each from seven Wickets, and further let's examine what we'd see if we executed the following:

```
PIP ANALDATA.PRN=TESTER.DAT
TYPE ANALDATA.PRN

1  200  206  210  300  310
2 1200 1210 1195 1182 1205
3 9000 9050 8950 9100 9200
4  200  300  150  120  100
5  120  121  130  131  150
6  119  207  300  500  600
7  250  300  495  600  750
```

What we'd like however is to see the mean values and the standard deviation for each unit 1 thru 7. Aha we say – Lotus is the way. So now we must create a spreadsheet. @LOTUS20 will get things under way nicely and anon we are merrily creating the worksheet. Essentially this consists of a Title "Wicket Data Analysis" with the Date to be printed, and various column headings as needed etc. As they say in the more academic journals, "the generation of the spreadsheet is left as an exercise for the student". I include here copies of both the blank spreadsheet and the cell-formulas. The sheet covers A1..J20. This will allow up to 20 units to be calculated and printed. Notice what the Macro starting at I5 dictates. /wgzy~causes all zeros to be replaced by blanks. Next, /finanaldata~causes importation of ANALDATA.PRN as numbers with the left top corner of ANALDATA.PRN entering the cell at which the pointer was located the last time a File Save Replace was executed. One must either be cautious or precede this import macro by a gotoA2 macro to be sure that the file imports into the right location. Now comes /ppral..h20~gq which prints to the printer (IBMPC.LST) range a1..h20 followed by a quit. We can now Name this macro and execution of it will bring in the data file and output the results. When you Quit Yes, and return to RSX land and PRINT IBMPC.LST, you will have your final product.

But wait! our file ANALDATA.PRN is sitting in RSX land and we're in MSDOS land. Not to fear. Before we can successfully execute this macro, we must go back to RSX and use our friend DOSFLX (cleverly installed as DFX) to pop a copy to the Lotus disk. One must take care to set up the Worksheet Global Default Directory to point to where ANALDATA.PRN will be placed and of course be certain to save the worksheet with that pointer. While we're at it, let's save this whole mess as TESTANAL.WK1.

The following is a printout of the worksheet showing only the first seven rows. The rest through row 20 is all the same, so no need to bore you with endless repetition. The Macro of interest is within this range and is included.

```
( A Printout of A1..J7
"CELL-FORMULAS" .. info
through J20 is the same)

Wicket Data Analysis
  02-Oct-86


A1: ^Unit #
B1: ^Test #1
C1: ^Test #2
D1: ^Test #3
E1: ^Test #4
F1: ^Test #5
G1: (F3) ^Avg Val
H1: (F3) ^Std Dev
J1: (T) '

A2: (F0) 0
B2: (F2) 0
C2: (F2) 0
D2: (F2) 0
E2: (F2) 0
F2: (F2) 0
G2: (F3) @AVG(B2..F2)
H2: (F3) @STD(B2..F2)

A3: (F0) 0
B3: (F2) 0
C3: (F2) 0
D3: (F2) 0
E3: (F2) 0
F3: (F2) 0
G3: (F3) @AVG(B3..F3)
H3: (F3) @STD(B3..F3)

A4: (F0) 0
B4: (F2) 0
C4: (F2) 0
D4: (F2) 0
E4: (F2) 0
F4: (F2) 0
G4: (F3) @AVG(B4..F4)
H4: (F3) @STD(B4..F4)

A5: (F0) 0
B5: (F2) 0
C5: (F2) 0
D5: (F2) 0
E5: (F2) 0
F5: (F2) 0
G5: (F3) @AVG(B5..F5)
H5: (F3) @STD(B5..F5)
I5: '/wgzy~

A6: (F0) 0
B6: (F2) 0
C6: (F2) 0
D6: (F2) 0
E6: (F2) 0
F6: (F2) 0
G6: (F3) @AVG(B6..F6)
H6: (F3) @STD(B6..F6)
I6: '/finanaldata~

A7: (F0) 0
B7: (F2) 0
C7: (F2) 0
D7: (F2) 0
E7: (F2) 0
F7: (F2) 0
G7: (F3) @AVG(B7..F7)
H7: (F3) @STD(B7..F7)
I7: '/ppra1..h20~gq

( A Printout of A1..J7
"AS-DISPLAYED" ..info through
J20 is the same)
```

```
Unit  Test #1  Test #2  Test #3  Test #4
0     0.00     0.00     0.00     0.00
0     0.00     0.00     0.00     0.00
0     0.00     0.00     0.00     0.00
0     0.00     0.00     0.00     0.00
0     0.00     0.00     0.00     0.00
0     0.00     0.00     0.00     0.00

/wgzy~
/finanaldata~
/pprai..h20~gq
```

So now we are able to move data files about with wanton abandon and create and execute spreadsheets with macros. All is well and good. "But" you say, "I thought you were talking about 'hands-off Lotus'?". To be sure I was. So how do we get to that point? Have you forgotten your Batch processor. Let's think about what must happen now. We must first move a copy of our ANALDATA.PRN to its proper MSDOS disk, then we must execute the @LOTUS20 right? Wrong! There's more to be done now. We need a smarter INIT.BAT, we also need to have TESTANAL.WK1 loaded automatically, and lastly we have to execute our macro automatically and sign out! Now what??

Well, how about the following for starters:

- Go back to your TESTANAL.WK1 and modify the macro by adding in I8 /qy . Then Name the macro I5..I8
  0 which is THE "auto-macro" which executes upon loading the spreadsheet. File Save all of this as TESTANAL.AUT. DO NOT Replace or you're DEAD!
- Create an INIT.AUT which is a modified copy of your original Lotus INIT.BAT except now, just before executing 123, copy TESTANAL.AUT to AUTO123.WK1. What this subtle operation does is cause your automatic version of TESTANAL.WK1 to be loaded upon execution of 123, and of course your 0 macro then executes - which does all your work then signs off. Next, add to INIT.AUT a deletion of AUTO123.WK1 so as to eliminate any automatic shenanigans the next time you want access to Lotus. This must be done before INIT.AUT says BYE.
- Now we're ready to roll. We will now create an RSX batch job file to employ DFX to move our ANALDATA.PRN and INIT.AUT to their proper locations, execute a command string to cause the CRD software to execute, BUT this time we subtly add another option to our CRD command string. That is the /D which causes direct connection to the serial port on the QCP-11+ for terminal I/O. There is nothing connected to it, so all the information we would have seen, and which would have locked up the VT: driver is conveniently dumped into the bit bucket.

Attached is the piece de resistance, a printout of the LOTUS.LOG file as the result of the command SUB /NOPRINT=LOTUS.BAT. It details every move of a fully automated hands-off execution of Lotus under RSX-11M+. It demonstrates completely that you can provide your users with a new service that they can't have with their PC's clones or whatever. You can link this all by the clock. We can even have a program run daily by resubmitting itself or on a particular day by testing for the day's name, or any other deviant variant that your warped mind can create.

I considered deleting all the repetitious DOSFLX chatter, but this would have messed up the output page numbering and formatting of the LOTUS.LOG file. As a result, I have included an untouched copy of all that transpired. Because of the way DOSFLX operates, I couldn't include comments within the batch file as they occurred and consequently had to group them prior to executing DFX, so you will have to review the groups of comments and trace their actions independently. As you can see, if DOSFLX were smarter and allowed parameter passing, this file could be about one tenth it's size. Interaction with DOSFLX quickly becomes boring because of all the waiting for menu printout.

I would like to take this opportunity to bless the many DECUS contributors who have been kind enough to share their knowledge, help, and software thus making this job actually possible if not altogether pleasurable. Without EDT and RUNOFF, this would have been a real bear!

As you can see, this work is recent. In fact it is still going on. There are many problems I haven't stumbled across yet. One that I know of in particular is that if you are planning a print queue for your Lotus output, you will have to rebuild the LPP driver according to the System Management Guide section 7 regarding /FORMS:n. You will probably need to set aside a printer for these users since they will inevitably set the page width to the max allowable since they are using Epsons. I haven't had time to get this far yet, but will. I was going to include a section on batch files submitting batch files which we are using regularly and even resubmitting on schedule.

Unfortunately, one cannot as yet share a DOSFLX disk via DecNetDOS although I have linked them in prior rev levels of DOSFLX. The results are unpredictable. This would have interesting possibilities. With all fairness, DOSFLX predates DecNetDOS and would therefore not necessarily be expected to be compatible. I did once read a file on DISKC.DOS which was accessed as D: on a remote PC/XT via DecNetDOS' NDU facility. These are indeed exciting times!

```
08:13:56        $JOB/TIME:5 LOTUSDEMO

        ====================================
        User Job - LOTUSD       Terminal VT1:
                  UIC = [7,67]
        ====================================

        TERM
          .    RSX-11M-PLUS V2.1 BL15E  [1,54] System     DRPSRC
          .
          .         Welcome to the Solvent Recovery Micro PDP-11/73 System
          .
          .         Please be sure to logout when you will be gone from you
          .    terminal for more than a few minutes. Remember that others want
          .    the system and we are limited in the number of available ports.
          .    co-operation is necessary to provide the most service to the la
          .    number of people.
          .
          .
          .
          .
          .         Art Hurst X4170  or mail to A_HURST or SYSTEM
08:13:58       $!
08:13:58       $!      Set bomb-out evacuation path to nearest Exit
08:13:58       $!
08:13:58       $ON ERROR THEN GOTO EXIT
08:13:58       $DCL SET DEFAULT DU0:[7,67]
08:13:59       $!
08:13:59       $!      Let's look at the two files we have in [7,67] for
08:13:59       $!      INIT that will be passed to [7,47]LOTUS20.DOS
08:13:59       $!      as well as the ANALDATA.PRN that we PIPped from
08:13:59       $!      TESTER.DAT
08:13:59       $!
08:13:59       $TYPE INIT.BAT
        TERM   cd\123
          .    crd123v2 >nul
          .    123
          .    cd\
          .    bye
          .    >
08:14:00       $TYPE INIT.AUT
        TERM   cd\123
          .    copy testanal.aut auto123.wk1
          .    crd123v2 >nul
          .    123
          .    del auto123.wk1
          .    cd\
          .    bye
          .    >
08:14:01       $TYPE ANALDATA.PRN
        TERM   1      200     206     210     300     310
          .    2      1200    1210    1195    1182    1205
          .    3      9000    9050    8950    9100    9200
          .    4      200     300     150     120     100
          .    5      120     121     130     131     150
          .    6      119     207     300     500     600
```

```
        TERM    7       250     300     495     600     750
          .     >
08:14:03        $!
08:14:03        $!      Now let's strip for action and bring DOSFLX into the
08:14:03        $!      fray to move between RSX-land and MSDOS-land
08:14:04        $!      First, we'll see what [7,67]DISKC1.DOS's AUTOEXEC.BAT
08:14:04        $!      file looks like ..
08:14:04        $!
08:14:04        $!      Then while we're in DOSFLX, let's switch to [7,47]LOTUS
08:14:04        $!      and examine the current INIT.BAT that would normally
08:14:04        $!      execute ..
08:14:04        $!
08:14:04        $!      Next we'll pull a switcheroo and put our cleverly
08:14:04        $!      modified INIT.AUT in place of the normal INIT.BAT ..
08:14:04        $!      First we'll replace the INIT.BAT ..
08:14:05        $!
08:14:05        $!      Then we'll see the new INIT.BAT that will execute
08:14:05        $!      on Boot up ..
08:14:05        $!
08:14:05        $!      Next let's switch directories on [7,47]LOTUS20.DOS
08:14:05        $!      and see what's in that directory
08:14:05        $!
08:14:05        $!      Notice that you don't have an ANALDATA.PRN because
08:14:05        $!      we haven't put one there yet. So let's proceed
08:14:05        $!      to put a copy there for our Import command to find.
08:14:06        $!
08:14:06        $!      And just to be sure it got there OK, let's type it.
08:14:06        $!
08:14:06        $DFX
        TERM    DOSFLX -- MSDOS <> RSX file transfer utility    V2.02
          .         Copyright (c) 1986 by Logicraft, Inc.
          .
          .
          .        [C]reate            -- Create a new DOS logical disk
          .        [D]irectory         -- Directory of the currently open logical
          .        [E]xit              -- Exit from DOSFLX
          .        [F]ile-transfer     -- File transfer: DOS -> RSX or RSX -> DOS
          .        [I]nitialize        -- Initialize Logical disk (destroys conter
          .        [M]kdir             -- Make a sub-directory
          .        [O]pen              -- Open an existing logical disk
          .        [P]urge             -- Purge a file from the current directory
          .        [R]mdir             -- Remove an empty sub-directory
          .        [S]et-directory     -- Set default directory
          .        [T]ype              -- Type an MSDOS TEXT file
          .
          .     DOSFLX >> o
          .
          .     Enter RSX filename specification: [7,67]diskc1
          .
          .        [C]reate            -- Create a new DOS logical disk
          .        [D]irectory         -- Directory of the currently open logical
          .        [E]xit              -- Exit from DOSFLX
          .        [F]ile-transfer     -- File transfer: DOS -> RSX or RSX -> DOS
          .        [I]nitialize        -- Initialize Logical disk (destroys conter
          .        [M]kdir             -- Make a sub-directory
          .        [O]pen              -- Open an existing logical disk
```

```
TERM      [P]urge             -- Purge a file from the current directory
  .       [R]mdir             -- Remove an empty sub-directory
  .       [S]et-directory     -- Set default directory
  .       [T]ype              -- Type an MSDOS TEXT file
  .
  .     DOSFLX >> t
  .
  .     Enter DOS filename specification: autoexec.bat
  .     echo off
  .     cls
  .     path c:\;d:\;e:\;f:\
  .     prompt $p$g
  .     if not exist d:init.bat goto end
  .     d:
  .     init
  .     end:
  .
  .       [C]reate            -- Create a new DOS logical disk
  .       [D]irectory         -- Directory of the currently open logical
  .       [E]xit              -- Exit from DOSFLX
  .       [F]ile-transfer     -- File transfer: DOS -> RSX or RSX -> DOS
  .       [I]nitialize        -- Initialize Logical disk (destroys conten
  .       [M]kdir             -- Make a sub-directory
  .       [O]pen              -- Open an existing logical disk
  .       [P]urge             -- Purge a file from the current directory
  .       [R]mdir             -- Remove an empty sub-directory
  .       [S]et-directory     -- Set default directory
  .       [T]ype              -- Type an MSDOS TEXT file
  .
  .     DOSFLX >> o
  .
  .     Enter RSX filename specification: [7,47]lotus20
  .
  .       [C]reate            -- Create a new DOS logical disk
  .       [D]irectory         -- Directory of the currently open logical
  .       [E]xit              -- Exit from DOSFLX
  .       [F]ile-transfer     -- File transfer: DOS -> RSX or RSX -> DOS
  .       [I]nitialize        -- Initialize Logical disk (destroys conten
  .       [M]kdir             -- Make a sub-directory
  .       [O]pen              -- Open an existing logical disk
  .       [P]urge             -- Purge a file from the current directory
  .       [R]mdir             -- Remove an empty sub-directory
  .       [S]et-directory     -- Set default directory
  .       [T]ype              -- Type an MSDOS TEXT file
  .
  .     DOSFLX >> t
  .
  .     Enter DOS filename specification: init.bat
  .     cd\123
  .     crd123v2 >nul
  .     123
  .     cd\
  .     bye
  .
  .       [C]reate            -- Create a new DOS logical disk
  .       [D]irectory         -- Directory of the currently open logical
```

```
TERM     [E]xit              -- Exit from DOSFLX
  .      [F]ile-transfer     -- File transfer: DOS -> RSX or RSX -> DOS
  .      [I]nitialize        -- Initialize Logical disk (destroys conter
  .      [M]kdir             -- Make a sub-directory
  .      [O]pen              -- Open an existing logical disk
  .      [P]urge             -- Purge a file from the current directory
  .      [R]mdir             -- Remove an empty sub-directory
  .      [S]et-directory     -- Set default directory
  .      [T]ype              -- Type an MSDOS TEXT file
  .
  .     DOSFLX >> f
  .     Direction of transfer [I]nput to/[O]utput of MSDOS disk: [I]  i
  .     Mode of transfer [T]ext/[B]inary: [T]  t
  .
  .     Enter RSX filename specification: init.aut
  .
  .     Enter DOS filename specification: [INIT    .AUT] init.bat
  .
  .      [C]reate            -- Create a new DOS logical disk
  .      [D]irectory         -- Directory of the currently open logical
  .      [E]xit              -- Exit from DOSFLX
  .      [F]ile-transfer     -- File transfer: DOS -> RSX or RSX -> DOS
  .      [I]nitialize        -- Initialize Logical disk (destroys conter
  .      [M]kdir             -- Make a sub-directory
  .      [O]pen              -- Open an existing logical disk
  .      [P]urge             -- Purge a file from the current directory
  .      [R]mdir             -- Remove an empty sub-directory
  .      [S]et-directory     -- Set default directory
  .      [T]ype              -- Type an MSDOS TEXT file
  .
  .     DOSFLX >> t
  .
  .     Enter DOS filename specification: init.bat
  .     cd\123
  .     copy testanal.aut auto123.wk1
  .     crd123v2 >nul
  .     123
  .     del auto123.wk1
  .     cd\
  .     bye
  .
  .      [C]reate            -- Create a new DOS logical disk
  .      [D]irectory         -- Directory of the currently open logical
  .      [E]xit              -- Exit from DOSFLX
  .      [F]ile-transfer     -- File transfer: DOS -> RSX or RSX -> DOS
  .      [I]nitialize        -- Initialize Logical disk (destroys conter
  .      [M]kdir             -- Make a sub-directory
  .      [O]pen              -- Open an existing logical disk
  .      [P]urge             -- Purge a file from the current directory
  .      [R]mdir             -- Remove an empty sub-directory
  .      [S]et-directory     -- Set default directory
  .      [T]ype              -- Type an MSDOS TEXT file
  .
  .     DOSFLX >> s
  .
  .     Enter DOS filename specification: 123
```

```
TERM
  .      [C]reate          -- Create a new DOS logical disk
  .      [D]irectory       -- Directory of the currently open logical
  .      [E]xit            -- Exit from DOSFLX
  .      [F]ile-transfer   -- File transfer: DOS -> RSX or RSX -> DOS
  .      [I]nitialize      -- Initialize Logical disk (destroys conter
  .      [M]kdir           -- Make a sub-directory
  .      [O]pen            -- Open an existing logical disk
  .      [P]urge           -- Purge a file from the current directory
  .      [R]mdir           -- Remove an empty sub-directory
  .      [S]et-directory   -- Set default directory
  .      [T]ype            -- Type an MSDOS TEXT file
  .
  .    DOSFLX >> d
  .    .        .             0    <dir>
  .    ..       .             0    <dir>
  .    LOTUS   .COM        5817
  .    INSTALL .EXE       55152
  .    INSTALL .SCR       43728
  .    INSTALL .DVC        4470
  .    TRANS   .COM       35026
  .    DIF     .XLT       26496
  .    DBF2    .XLT       35792
  .    DBF3    .XLT       41664
  .    JZZLOTUS.XLT       11936
  .    VCWRK   .XLT       18640
  .    WR1WRK  .XLT       19792
  .    WRKWR1  .XLT        1024
  .    WR1WKS  .XLT       29840
  .    UTIL    .SET       10048
  .    PGRAPH  .EXE       66336
  .    PGRAPH  .CNF         384
  .    PGRAPH  .HLP        6997
  .    LOTUS   .FNT        8679
  .    BLOCK1  .FNT        5737
  .    BLOCK2  .FNT        9300
  .    BOLD    .FNT        8624
  .    FORUM   .FNT        9727
  .    ITALIC1 .FNT        8949
  .    ITALIC2 .FNT       11857
  .    ROMAN1  .FNT        6863
  .    ROMAN2  .FNT       11847
  .    SCRIPT1 .FNT        8132
  .    SCRIPT2 .FNT       10367
  .    123     .DYN       10913
  .    123     .SET       32966
  .    COPYON  .BAT        1699
  .    COPYOFF .BAT        1389
  .    COPYON1 .BAT         117
  .    COPYOFF1.BAT         116
  .    S123ON  .COM        1818
  .    S123OFF .COM         233
  .    COPYHARD.COM       40112
  .    INSTALL .LBR      266562
  .    123     .CMP      133848
  .    123     .COM        2048
```

```
TERM   123      .HLP          114366
  .    123      .CNF             265
  .    VDF0203 .VDF             2192
  .    CML0203 .FCL            13136
  .    CRD123V1.EXE             1865
  .    CRD123V2.EXE             1812
  .    123V2X1 .DT1             8192
  .    123V2X2 .DT1             8191
  .    123V2X1 .DT2             2560
  .    123V2X2 .DT2             2559
  .    TESTANAL.WK1             4153
  .    TEST    .PIC              805
  .    TESTANAL.AUT             4196
  .
  .
  .    Directory statistics for current DOS volume:
  .
  .            Free directory entries   :          9
  .            Free data blocks         :        651
  .            Number of bad blocks     :          0
  .            Number of reserved blocks:          0
  .
  .      [C]reate           -- Create a new DOS logical disk
  .      [D]irectory        -- Directory of the currently open logical
  .      [E]xit             -- Exit from DOSFLX
  .      [F]ile-transfer    -- File transfer: DOS -> RSX or RSX -> DOS
  .      [I]nitialize       -- Initialize Logical disk (destroys conter
  .      [M]kdir            -- Make a sub-directory
  .      [O]pen             -- Open an existing logical disk
  .      [P]urge            -- Purge a file from the current directory
  .      [R]mdir            -- Remove an empty sub-directory
  .      [S]et-directory    -- Set default directory
  .      [T]ype             -- Type an MSDOS TEXT file
  .
  .    DOSFLX >> f
  .    Direction of transfer [I]nput to/[O]utput of MSDOS disk: [I]  i
  .    Mode of transfer [T]ext/[B]inary: [T]  t
  .
  .    Enter RSX filename specification: analdata.prn
  .
  .    Enter DOS filename specification: [ANALDATA.PRN]
  .
  .      [C]reate           -- Create a new DOS logical disk
  .      [D]irectory        -- Directory of the currently open logical
  .      [E]xit             -- Exit from DOSFLX
  .      [F]ile-transfer    -- File transfer: DOS -> RSX or RSX -> DOS
  .      [I]nitialize       -- Initialize Logical disk (destroys conter
  .      [M]kdir            -- Make a sub-directory
  .      [O]pen             -- Open an existing logical disk
  .      [P]urge            -- Purge a file from the current directory
  .      [R]mdir            -- Remove an empty sub-directory
  .      [S]et-directory    -- Set default directory
  .      [T]ype             -- Type an MSDOS TEXT file
  .
  .    DOSFLX >> t
  .
```

```
        TERM   Enter DOS filename specification: analdata.prn
          .    1       200     206     210     300     310
          .    2       1200    1210    1195    1182    1205
          .    3       9000    9050    8950    9100    9200
          .    4       200     300     150     120     100
          .    5       120     121     130     131     150
          .    6       119     207     300     500     600
          .    7       250     300     495     600     750
          .
          .      [C]reate          -- Create a new DOS logical disk
          .      [D]irectory       -- Directory of the currently open logical
          .      [E]xit            -- Exit from DOSFLX
          .      [F]ile-transfer   -- File transfer: DOS -> RSX or RSX -> DOS
          .      [I]nitialize      -- Initialize Logical disk (destroys conter
          .      [M]kdir           -- Make a sub-directory
          .      [O]pen            -- Open an existing logical disk
          .      [P]urge           -- Purge a file from the current directory
          .      [R]mdir           -- Remove an empty sub-directory
          .      [S]et-directory   -- Set default directory
          .      [T]ype            -- Type an MSDOS TEXT file
          .
          .    DOSFLX >> e
08:14:36       $!
08:14:36       $!      All finished, now we can go back to RSX land
08:14:36       $!
08:14:36       $!
08:14:36       $!      Let's set up a new error pointer because I'm
08:14:36       $!      going to ask for a non existent file just to
08:14:36       $!      prove that none exists prior to doing the
08:14:36       $!      CRD trick!
08:14:37       $!
08:14:37       $ON ERROR THEN GOTO CRD
08:14:37       $DIR ANALDATA.LST
        TERM   PIP -- No such file(s)
          .
          .    >
08:14:38       $!
08:14:38       $!      See -- No file yet. This will be the PC output file.
08:14:38       $!

        'ERROR' exit status returned - enabling action in "ON" command

08:14:38       $CRD:
08:14:38       $!
08:14:38       $!      And now ladies and gentlemen, as you can see
08:14:38       $!      my sleeves are rolled up to a discreet length
08:14:38       $!      and I have no visible support. I will invoke the
08:14:38       $!      magic "CRD" and smoke will rise as my slave slowly
08:14:39       $!      comes to life in a virtual mode pouring much valuable
08:14:39       $!      data into the infinite bit bucket.
08:14:39       $!
08:14:39       $CRD ANALDATA.LST=[7,67]DISKC1,[7,47]LOTUS20/I/B/D/V:7452.
        TERM   CRD -- CARDWARE Configuration Error -- received command 21
08:15:08       $!
08:15:08       $!      Wait! it stirs, did anything happen??
08:15:08       $!
```

```
08:15:08        $!      (you must ignore the nasty "configuration error"
08:15:09        $!      message-- I've been so busy that I haven't been
08:15:09        $!      able to upgrade my CRD software. Please ignore
08:15:09        $!      this warning)
08:15:09        $!
08:15:09        $!      But -- did we finally get our card output file
08:15:09        $!      ANALDATA.LST?? The suspense is unbearable.. Let's
08:15:09        $!      take a look..
08:15:09        $!
08:15:09        $TYPE ANALDATA.LST
        TERM
          .
          .                                  Wicket Data Analysis
          .
          .
          .          Unit #   Test #1  Test #2  Test #3  Test #4  Test #5  Avg
          .               1    200.00   206.00   210.00   300.00   310.00  245.
          .               2   1200.00  1210.00  1195.00  1182.00  1205.00 1198.
          .               3   9000.00  9050.00  8950.00  9100.00  9200.00 9060.
          .               4    200.00   300.00   150.00   120.00   100.00  174.
          .               5    120.00   121.00   130.00   131.00   150.00  130.
          .               6    119.00   207.00   300.00   500.00   600.00  345.
          .               7    250.00   300.00   495.00   600.00   750.00  479.
          .
          .
          .
          .
          .
          .
          .
          .
          .
          .
          .
          .
          .
          .
          .     >
08:15:11        $!
08:15:11        $!      Lo and behold -- it lives and breaths. Living proof
08:15:12        $!      that you can provide total hands-off Lotus via
08:15:12        $!      batch mode. Now you can provide more services than
08:15:12        $!      your users ever dreamed of. Of course you won't be
08:15:12        $!      any the richer for it, but your newfound smugness
08:15:12        $!      is richly deserved all will agree.
08:15:12        $!
08:15:12        $!
08:15:12        $!      Before we leave, we had better undo some of the
08:15:12        $!      things that we modified to perform this wizardry.
08:15:12        $!      Namely, we had better restore INIT.BAT, or the
08:15:13        $!      next poor jerk that executes Lotus will find
08:15:13        $!      it coming and going automatically, and we wouldn't
08:15:13        $!      want that!
08:15:13        $!
08:15:13        $!      And while we're at it, we should also Purge
08:15:13        $!      ANALDATA.PRN.
08:15:13        $!
```

```
08:15:14        $DFX
        TERM    DOSFLX -- MSDOS <> RSX file transfer utility    V2.02
          .         Copyright (c) 1986 by Logicraft, Inc.
          .
          .
          .       [C]reate            -- Create a new DOS logical disk
          .       [D]irectory         -- Directory of the currently open logical
          .       [E]xit              -- Exit from DOSFLX
          .       [F]ile-transfer     -- File transfer: DOS -> RSX or RSX -> DOS
          .       [I]nitialize        -- Initialize Logical disk (destroys conter
          .       [M]kdir             -- Make a sub-directory
          .       [O]pen              -- Open an existing logical disk
          .       [P]urge             -- Purge a file from the current directory
          .       [R]mdir             -- Remove an empty sub-directory
          .       [S]et-directory     -- Set default directory
          .       [T]ype              -- Type an MSDOS TEXT file
          .
          .     DOSFLX >> o
          .
          .     Enter RSX filename specification: [7,47]lotus20
          .
          .       [C]reate            -- Create a new DOS logical disk
          .       [D]irectory         -- Directory of the currently open logical
          .       [E]xit              -- Exit from DOSFLX
          .       [F]ile-transfer     -- File transfer: DOS -> RSX or RSX -> DOS
          .       [I]nitialize        -- Initialize Logical disk (destroys conter
          .       [M]kdir             -- Make a sub-directory
          .       [O]pen              -- Open an existing logical disk
          .       [P]urge             -- Purge a file from the current directory
          .       [R]mdir             -- Remove an empty sub-directory
          .       [S]et-directory     -- Set default directory
          .       [T]ype              -- Type an MSDOS TEXT file
          .
          .     DOSFLX >> f
          .     Direction of transfer [I]nput to/[O]utput of MSDOS disk: [I]  i
          .     Mode of transfer [T]ext/[B]inary: [T]  t
          .
          .     Enter RSX filename specification: init.bat
          .
          .     Enter DOS filename specification: [INIT    .BAT]
          .
          .       [C]reate            -- Create a new DOS logical disk
          .       [D]irectory         -- Directory of the currently open logical
          .       [E]xit              -- Exit from DOSFLX
          .       [F]ile-transfer     -- File transfer: DOS -> RSX or RSX -> DOS
          .       [I]nitialize        -- Initialize Logical disk (destroys conter
          .       [M]kdir             -- Make a sub-directory
          .       [O]pen              -- Open an existing logical disk
          .       [P]urge             -- Purge a file from the current directory
          .       [R]mdir             -- Remove an empty sub-directory
          .       [S]et-directory     -- Set default directory
          .       [T]ype              -- Type an MSDOS TEXT file
          .
          .     DOSFLX >> s
          .
          .     Enter DOS filename specification: 123
```

```
        TERM
          .       [C]reate            -- Create a new DOS logical disk
          .       [D]irectory         -- Directory of the currently open logical
          .       [E]xit              -- Exit from DOSFLX
          .       [F]ile-transfer     -- File transfer: DOS -> RSX or RSX -> DOS
          .       [I]nitialize        -- Initialize Logical disk (destroys conter
          .       [M]kdir             -- Make a sub-directory
          .       [O]pen              -- Open an existing logical disk
          .       [P]urge             -- Purge a file from the current directory
          .       [R]mdir             -- Remove an empty sub-directory
          .       [S]et-directory     -- Set default directory
          .       [T]ype              -- Type an MSDOS TEXT file
          .
          .     DOSFLX >> p
          .
          .     Enter DOS filename specification: analdata.prn
          .
          .       [C]reate            -- Create a new DOS logical disk
          .       [D]irectory         -- Directory of the currently open logical
          .       [E]xit              -- Exit from DOSFLX
          .       [F]ile-transfer     -- File transfer: DOS -> RSX or RSX -> DOS
          .       [I]nitialize        -- Initialize Logical disk (destroys conter
          .       [M]kdir             -- Make a sub-directory
          .       [O]pen              -- Open an existing logical disk
          .       [P]urge             -- Purge a file from the current directory
          .       [R]mdir             -- Remove an empty sub-directory
          .       [S]et-directory     -- Set default directory
          .       [T]ype              -- Type an MSDOS TEXT file
          .
          .     DOSFLX >> e
08:15:25        $EXIT:
08:15:25        $EOJ
        TERM    Connect time:  1     minutes
          .     CPU time used: 31    seconds
          .     Task total:    20
```

# HOW TO GET THAT UPGRADE

Denny Walthers
2723 Pampas
Orange, California 92665
(714) 974-2486

ABSTRACT

Upgrading systems is a continuous function. The process for obtaining funds to accomplish this can be very frustrating. This paper provides some approaches that have been used successfully by the author. It is intended to provide the reader with a brief understanding of the external environment that can have a dramatic effect obtaining funding. If upgrades are planned at any time in the future, concepts dealt with here should make obtaining the necessary funding easier.

"We need to upgrade ...," ever heard that before? In my own personal experience (22 years worth), I have yet to see a year go by that an upgrade of some kind was not required to keep the business going. The most frustrating part is educating 'management' so they understand the need. Hopefully this article will address some of the techniques for overcoming that dilemma.

One of the leading problems with those of us who are computer techies is we control almost everything in the world; processes, manufacturing, quality, inventory, distribution, payroll, payables and receivables, et cetera. We control everything except the one thing that really matters. We don't control MONEY.

Thus, we are faced with the challenge of always having to ask for money to accomplish the things we were hired to do. Ever find it odd that you were actually hired to do some specific thing and 'after' you got on board learned that you could not possibly do it? It isn't because of your lack of technical skills, but rather the lack of understanding from 'up top' as to what is really required.

Let us examine some of the techniques to obtain funds for upgrades, total new systems or whatever. Some basic things that must be dealt with are:

- Understanding your company
- Understanding your management
- Understanding your need
- Understanding your financial organization

## UNDERSTANDING YOUR COMPANY

Most of us tend to concentrate on the things that we are familiar with and directly affect us. We can easily get lost daily in the trivial things that seem to require our immediate attention. Few of us (myself included) ever take the time or find the time to learn about the rest of our company. What does the company do? What is important to the company? Does it sell a product or services?

One thing is certain about companies in general. You must 'sell' to them internally in order for them to achieve their own business goals. It's all a game. When you realize it is a game and you learn the rules, you might actually enjoy playing it because the challenge is to beat others using their own rules. Learn the rules, play to win, and reap the benefits. Externally, sales people of various types call on the purchasing department and management in order to 'sell' them products. Internally, each department does exactly the same thing but the item being sold is generally an idea, not a product.

What are the things that are important for you to know about your company? I believe that there are four essential items.

### Know Your Enemy

What do I mean 'enemy'? Aren't we all working for the same company and for the same goals? If you believe that, you really are a candidate for buying the Brooklyn Bridge. Every person in every company has at least some self-serving needs that must be met. These are the things that make us achievers. Sometimes that gets in our way. We will tend to look disfavorably on things that are of no direct benefit to us individually. So much for the 'enemies' qualifier.

What kind of enemies? Not physical but rather those who either conflict with your ideas or those who want the credit for them. You can be sure you are competing for the same funds another department is seeking. There may be a decision that has to be made between your upgrade and adding a salesman or two. There may also be the person that you embarrassed months ago who is now in a position to delay your efforts. View anyone who will not support your requests vigorously as an enemy to your success.

Learn who can delay or stop your efforts altogether, and the reasons why, if you do not already know them. Approach the individual and work the problem out. Find some common ground that you both agree on and get them on your side. This is usually possible and often leads to a better working relationship for the future.

Finally, there is the MBA. Recent books and business periodicals have discovered this insidious threat! It doesn't matter whether they are from Harvard or from some university across the street. Their philosophy since the mid-sixties has been the same. Increase the short term profits of the company, even at it's own expense. Never mind long range goals and profits. Look only for the 'short term' and ignore the future (they are seeing to it that you don't have one). Unfortunately these people have successfully attained powerful positions in all facets of companies. They don't want you to upgrade anything or spend any money at all. Their attitude is that you are not working hard enough or you would not need the upgrade. Short term profits are all that they are interested in. These people exist and you must be prepared to do battle with them on their terms. They understand numbers only and I address numbers later in this paper.

Know Your Allies

These are your assets, use them wisely. Don't ever jeopardize your relationship with these people. Keep them informed and keep them on your side. Take the time to solicit support from those who are well thought of in your company. Some may have been there a long time and some may be relatively new but have excellent ideas.

Discuss your thoughs and needs with them, get their ideas. Ask them how they would approach the problem. You will find that most people are willing to provide help of this type because it builds a positive working relationship. They will know that they can count on you when they need assistance.

Poll your user community and get them to solidly support you. One of the things that will work in your favor is the support of your user community. If they are promoting the same thing that you are, when it gets to top management, and all parties agree, bingo, you win and the company does too.

Know Who Makes the Decisions

You can do all of the right things, talk to all of the right people and still fail. Why? Because you forgot or worse, you never knew who it was that was making the decisions. You may work with a vendor for months, sell all of management on them and your need, get the money and then ... FAIL. Why? Perhaps the purchasing department forces the issue of requiring at least three bids. Some other vendor bids less and you lose because you get the wrong equipment and it doesn't work. Many people think that Vice-Presidents, Controllers or CFO's make the decisions. That is usually wrong. There are numerous financial analysts, users et cetera who will shoot you down long before your proposal gets to a Vice-President. If your proposal gets to a Vice-President, chances are that you are going to get what you want.

Find out who makes the decisions and be sure they are always involved. Obtain their support, and the rest is simply formality.

What are the Current Hot Buttons

If there are current things that are going on in your company, you may be able to capitalize on these. Is your upgrade going to do anything toward these items? If it is, even in a minor way, exploit that fact. Hot buttons generally get attention immediately because they usually have a lot of general management support. Avoid those that are risky and controversial. It may cost you your upgrade and that is certainly not what you want.

UNDERSTAND YOUR MANAGEMENT

What is the orientation of your management: sales, engineering, manufacturing, finance, distribution? It is imperative that you know. This knowledge provides you with the ability to tailor the description of your requirements to your audience. Try to keep the narrative simple and perhaps relate some example from their area of expertise supports your position.

Know the strengths and weaknesses of your management. Remember, you are playing a business game in which you want to win. If you can make both you and your management winners no one will turn you down. Exploit weaknesses but do not embarrass anyone who could become an adversary. Exploit strengths as well, involve them and make them look good. It pays off! Especially for future requests that you may have.

Most management focuses on business goals. Does your upgrade support and assist in the achievement of these business goals? If so, management will support your efforts. If not, you are already in trouble and need to develop that type of relationship.

UNDERSTAND YOUR NEED

It is remarkable how many times I have been asked to help support an effort and the people leading it have had no clear, concise understanding of exactly what it is they are trying to achieve. "We are trying to get an upgrade because we need it." That is commendable, but WHY do you need it.

Give reasons and approaches, not emotional appeals. Capacity is a reason. State-of-the-art tools is a reason. 'We need it,' is not a reason. YOU know you need it, THEY don't. The burden of proof is on you the requester. In order to communicate that need you must present the facts in language that the 'decision makers' can understand.

Planning helps too. While it is important to deal with the short term it is equally important to deal with the future. Examine your company's growth patterns. Will the upgrade that you seek be sufficient for five years? Show you have done your homework and long range planning. Provide for contingencies. What happens if you do less of an upgrade now and more later. Address the business goals and support them with your PLANS.

What are the alternatives? There are no alternatives? There are no alternatives you say? WRONG! There are always alternatives. The best argument for that is the alternative of DOING NOTHING. Yes, that is a realistic alternative, doing nothing. How often have you noticed that this alternative is the preferred approach!! You would be amazed how much ground you can get out of that when you are writing up a request for money.

Explore alternatives and explain each. People know that you have thoroughly explored the problem that way.

Is this a requirement from a governmental agency? Many agencies like the Air Quality Management District (AQMD), Food & Drug Administration (FDA), Environmental Protection Agency (EPA) et cetera have requirements that you can use to your advantage. Check it out. If it is there, use it. Be sure however that you do not use it unwisely. There is nothing so degrading as citing some government requirement, that is not a real requirement when investigated.

## KNOW YOUR FINANCIAL ORGANIZATION

The biggest delays occur within the financial area. Finance has the responsibility of protecting the company's money and assets. It is their job to investigate and examine all requests, including yours. Imagine someone who had your checkbook and the authority to write checks. Uncontrolled, they could do you in quickly. Finance is there to do their job. That can work to your advantage or disadvantage depending largely on you.

Finance is interested in the fiscal aspects of your need. Was this upgrade budgeted, was it planned, is there money available? These are some of the questions they are concerned with. Knowing this and knowing the answers up front can save you a lot of time and delays in getting your upgrade. They are interested in knowing exactly when money will be needed and spent. Finance deals a lot with cash flow and none of us knows enough about cash flow.

Who is the individual in the financial organization that does the evaluation of requests for money? Find out. Go to that person and determine what formats they like. Find out what they are looking for. They should be happy to tell you. After all it's their job and they should be willing to give you the information.

Was finance aware of the upgrade early in the planning stage, or did it come as a surprise? If you involve them in the beginning and use them to help plan the financial arrangements, it can lessen the impact and shorten the time to obtain approval.

## REQUEST FOR CAPITAL INVESTMENT

We use a standard format for obtaining funds for all capital projects. It consists almost entirely of narrative and can either make or break your project. Let us examine each section and see how they affect the overall request effort. Note that no matter what your company's format, all of these items must be addressed.

### Background

This is a narrative section which should discuss sufficient background information to explain how we arrived at this point. It is not necessary to cover the entire history. Things such as runaway growth due to division expansion, unknown or new requirements et cetera. Remember, most of the people reading this have no idea how it came about. Treat this setting as if you are in a court of law where all that counts are the facts that you present.

### Proposal

What are you proposing? Equipment, software, change in direction, maintaining service levels, just what is it. This section must be as explicit as possible without placing unrealistic constraints on what you need. Identify the fact that you are upgrading, or adding a new item and establish the credibility that you know what you are talking about.

### Discussion

This is where you provide some narrative about how things are being done currently. If capacity is your problem, discuss it. Not just that you are out of capacity but things such as increased response time, more frequent removal of active data from overloaded disks et cetera. Provide the reader with what you would say to them if you were speaking to them.

### Applications

Define explicitly the initial applications provided by the upgrade. If you are vague in this area, there will be a sense of concern on the part of the reader that you are somehow trying to obtain money to be used for something other than what is being defined. You don't want this at all. It can establish barriers for many years to come when your name appears on requests for money.

### Costs

This is financial information so make sure it is accurate. You should be sure to include costs for equipment, installation, additional monthly maintenance costs, tax, shipping, insurance etc. Many small items are overlooked when preparing this type of request. How many of us order things from someone and unwittingly forget the tax and shipping charges. Then when we get the item and the invoice comes, ZAP!!, additional money that we did not anticipate is required. In most instances it is not necessary to list each item individually or even the model numbers. Simply list the type equipment you plan to buy (i.e., 2 disk drives not 2 RM05 disk drives 256Mbytes each).

### Alternatives

This is the worst area of all. Invariably people will say, 'there are no alternatives'. That statement will cause the longest delay possible. The financial organization has just been given a free reign to investigate your claims on their own. You are now in trouble!

In every instance there is at least one alternative ... do nothing. Yes, that is a realistic alternative. Probably not an acceptable one for you, but acceptable for others (remember the MBA?). Another alternative may be to increase disk capacity over a two or three year period. Write the request to cover the entire period. Spread it out over time and identify when each piece of equipment will be purchased. Once approved, the money is put into your pocket and you add the equipment when necessary even if you elect to bring it in earlier.

In your discussion of alternatives, number each one and discuss it individually. Also discuss the possibility that the item you are requesting may be leased rather than purchased. This allows for upgrading later to newer and more current equipment. Financial organizations prefer to lease rather than purchase whenever possible as the tax benefits are usually better.

Recommendation

This is your final shot. You are the recognized expert, if you have done your work properly. Your recommendations weigh heavily in the decision process and that fact should not be overlooked. If you have not requested money for anything before, be sure from the beginning that you establish your credibility as an expert. Make sure that the money you request, when approved, you spend. Do not ask for significantly more or less than you need. Establish the costs to the best of your knowledge. As time progresses and you write more of these, you will be looked on favorably as an expert who does things well.

## OTHER NOTEWORTHY THINGS

If at all possible, learn something about the way that finance views the expenditure of capital money. One element that is of primary importance is Return On Investment or ROI. Most of us would never provide $100,000 to anyone without knowing up front how we were going to benefit from it.

Remember, the financial organization is the watchdog of your company. It is their responsibility to get the best bang for the buck when spending. Appeal to their emotional needs and get them to walk you through how ROI is calculated for your company. Put that on your system or into a spreadsheet and never lose it. You have now just wiped out their primary objection route. You have begun to learn the game and are now playing it well.

Provide statistics that no one can argue with because a computer generated them. How many people do you know that understand computers, operating systems or the like? You have a wealth of statistics that you can provide and that NO ONE can argue with. You are the expert, not them.

Let me give you an example. I was recently asked to make a presentation to our parent company. The IBM operations folks are always proud of the number of jobs that they run through their gigantic machine (around 200 per day).

I siezed the opportunity and uploaded a copy of RMD and the RMD statistics page to our slide generation PC. Bingo, since my presentation was before theirs, I showed that in 30 hours since the machine had been booted we had executed over 12000 jobs and had over 500 logons. Now, was that really fair? No, I was comparing a Mercedes to a Duesenberg but I got my point across. I was the expert and there was no one present who could argue my numbers because I had taken them in 'real-time' from the system statistics display.

Sometimes it is necessary to spread one request over a period of time. Let us hypothesize that you need terminals and that you cannot purchase them without getting money through formal channels. You investigate and find that you can 'rent' anything with a simple purchase order if it costs less than $150 per month.

First find a terminal rental company, rent one terminal for $125 per month. Then each week have the rental company add two or three more to the invoice. Get the terminals that you need and pay for them on rental. Now for the good part. In three years, write a request for money to replace these terribly expensive terminals that you have on rental with some new spiffy state-of-the-art terminals that cost less. There will be no question as to the justification. You are considered extremely sharp individual because you are actually cutting expenses. Wow, what a great deal. No one will ever question whether or not the original terminal were justified. After all, they have been there for three years or more.

## REQUEST FOR CAPITAL INVESTMENT (RCI)

Outline and Content

Background - This section should provide a brief history of the current situation.

Proposal - This should identify what is actually being proposed for the solution.

Discussion - This should define the current position and methods.

Applications - This should define the applications which will benefit from the upgrade.

Equipment Cost - Although it may be self-explanatory detail the equipment type and costs. If at all possible, DO NOT itemize specific model nubmers or brand names.

Alternatives - What else could be done? Nothing? Something less? Something over time? Examine just what alternatives there really are and discuss.

Recommendation - Select one of the above alternatives and mildly defend your position. Remember, here YOU are the expert. Use that to your advantage.

# OCEANOGRAPHIC DATA QUALITY CONTROL AND DISTRIBUTION SYSTEM

LCDR Lloyd K. Thomas
National Oceanic and Atmospheric Administration
National Ocean Service
Ocean Observations Division
6001 Executive Boulevard, Room 103
Rockville, Maryland 20852

## ABSTRACT

Timely and accurate meteorological and oceanographic data, collected over the vast oceans, are essential for accurate weather forecasts. NOAA's National Ocean Service (NOS) is responsible for implementing, monitoring, quality controlling, and distributing these data. NOS, in cooperation with the National Weather Service (NWS) and the National Environmental Satellite, Data, and Information Service (NESDIS), has developed the capability of delivering both marine meteorological and subsurface temperature data (taken by expendable bathythermographic [XBT] probes) accurately and quickly using a Shipboard Environmental [Data] Acquisition System (SEAS) unit and the Geostationary Operational Environmental Satellite (GOES) satellite system. The quality control and distribution system consists of two MicroPDP 11/23+'s operating under MicroRSX.

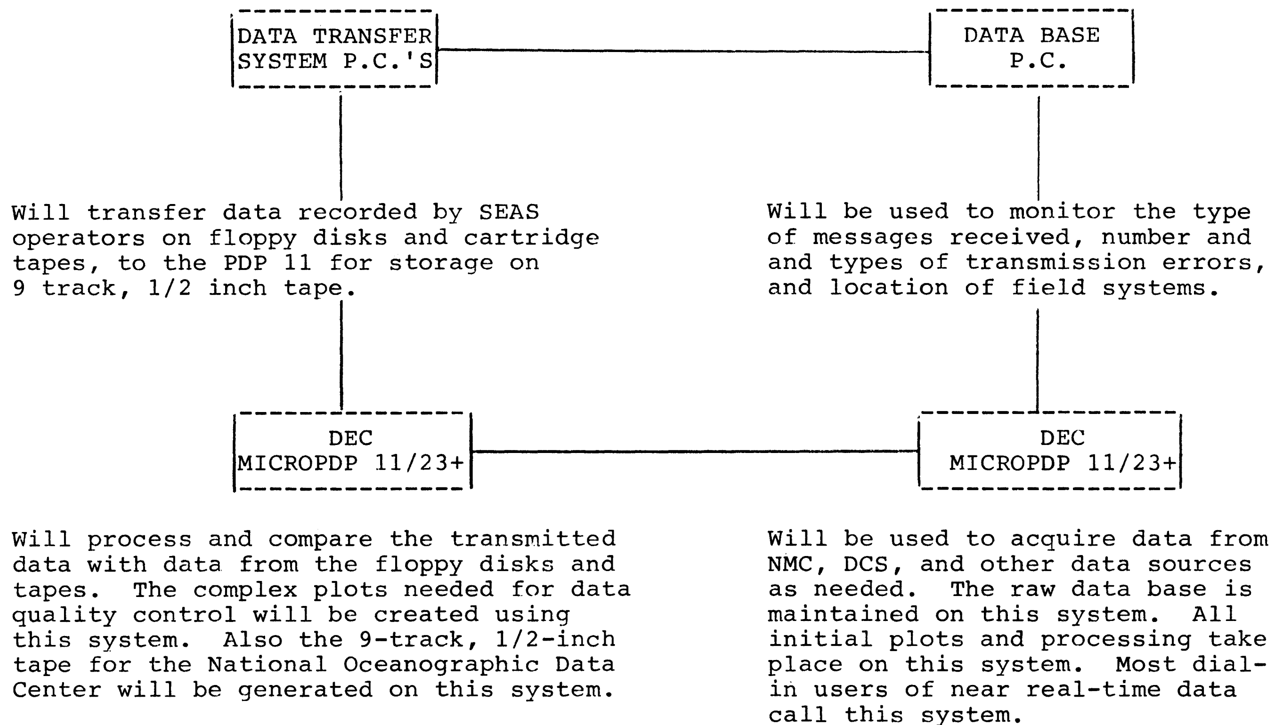
## INTRODUCTION

The SEAS program has been developed in order to deliver data from ships at sea accurately and quickly. The method of data delivery utilizes the GOES satellite system composed of spacecraft in synchronous equatorial earth orbit. SEAS currently delivers standard shipboard meteorological observations, subsurface temperature data (XBT), and subsurface salinity. The data which are entered into the SEAS units, either manually or automatically, are then automatically transmitted via GOES. The data are then received at Wallops Island, Virginia, and passed to the NESDIS headquarters in Suitland, Maryland. The data are used by the National Meteorological Center (NMC) synoptic forecast file for meteorological data and SST analysis file for subsurface (XBT) data. The Ocean Observations Division (OOD) monitors the data from both the NESDIS headquarters in Suitland, Maryland, and from NMC.

## SYSTEM CONFIGURATION

After reception by NESDIS, data from remote ocean areas are acquired over the switched telephone network by the data acquisition MicroPDP 11/23+. The data are also transferred to NMC by NESDIS, processed, and placed in the NMC database. The data acquisition MicroPDP 11/23+ also acquires these processed data over the switched telephone network for comparison with the raw data as received by NESDIS. Both of these data sets are processed by the data acquisition MicroPDP 11/23+, and selected products are transferred to the database P.C. and the second MicroPDP 11/23+. The second MicroPDP 11/23+ will do the non-real -time processing and comparisons with data received from HP-85 tape cartridges and IBM-PC.-format floppy disks from ocean going ships.

OCEAN OBSERVATIONS DIVISION HARDWARE FUNCTIONS
(DATA MONITORING)



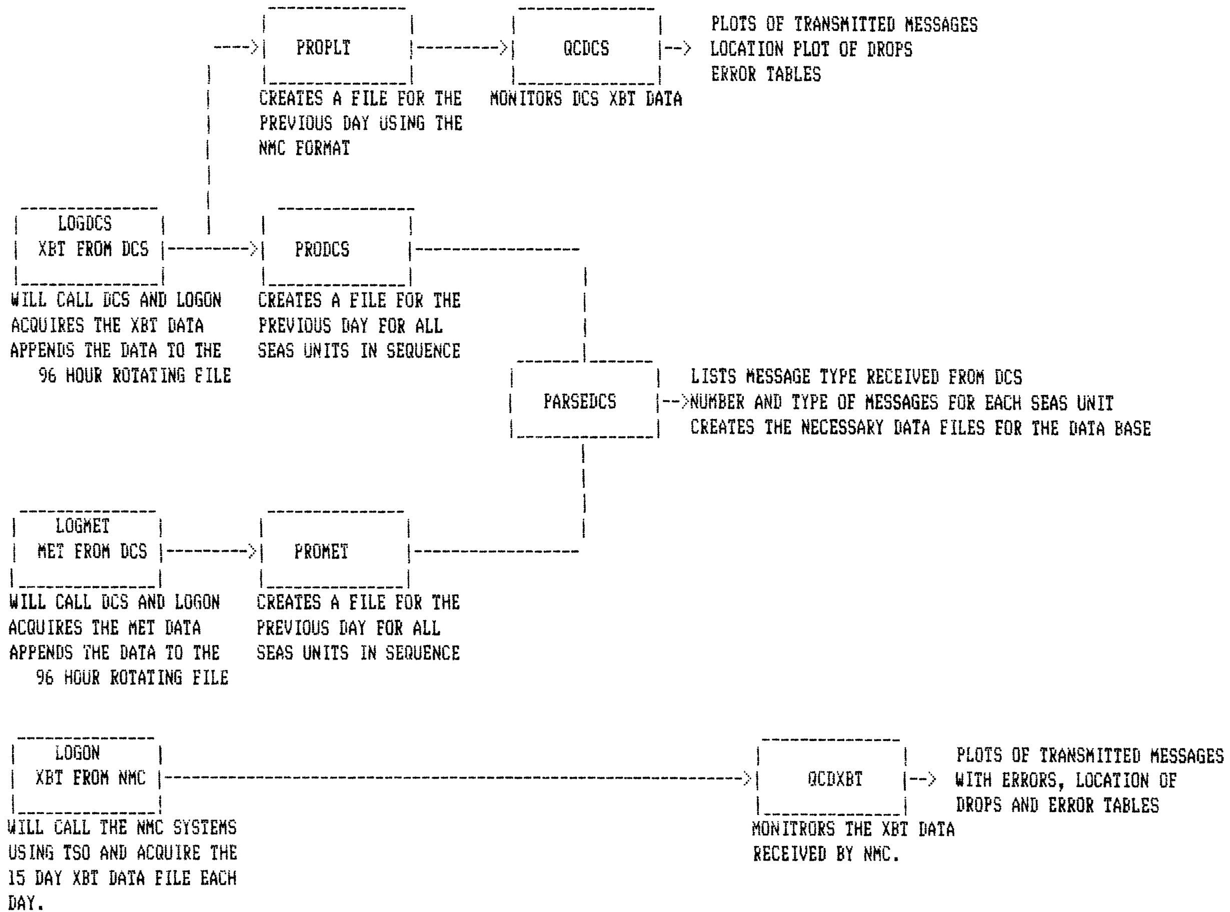

Will process and compare the transmitted data with data from the floppy disks and tapes. The complex plots needed for data quality control will be created using this system. Also the 9-track, 1/2-inch tape for the National Oceanographic Data Center will be generated on this system.

Will be used to acquire data from NMC, DCS, and other data sources as needed. The raw data base is maintained on this system. All initial plots and processing take place on this system. Most dial-in users of near real-time data call this system.

## SYSTEM SOFTWARE

Data are acquired in three separate phases. First the temperature/salinity data are obtained directly from NESDIS in the same format as transmitted by the field. The temperature/salinity records are separated and placed in a 96-hour rotating file for access by our data quality monitoring programs and other interested users via dial-up lines. The second set of data acquired is the meteorological or MET data. These data are also placed in a 96-hour rotating file. The final data set that is acquired and monitored is the bathymetric temperature/depth file from NMC. This file contains observations for 15 days and is monitored for completeness and the presence of any processing errors introduced at NMC. All programming was done using MicroRSX Fortran 77. The initial programming was done on MicroRSX V1.1, with MicroRSX V3.1 currently used.

The LOGDCS, LOGMET, and LOGON tasks utilize the QIO function extensively. Multiple QIO's are queued. This ensures that no data will be lost during reception. The PROMET, PRODCS, and PROPLET programs use the rotating data files and build a single file with selected data records from the raw rotating files. The "PRO" processing tasks use only standard Fortran statements with no MicroRSX executive calls. The final products are created using the QCDCS, PARSEDCS, and QCXBT. Additional work is still needed to create the necessary software for comparison of products for completeness and for ingest of the IBM-PC format floppy disks and the HP-85 tape cartridges.




## EXPERIENCE GAINED

Several unique problems were encountered during the software development. A disturbing problem was that the system at NMC could not accept a continuous data stream. A subroutine that blocked out one character at a time was developed with a delay of several milliseconds between characters. A second and much more serious problem was encountered when the upgrade was finished from MicroRSX V1.1 to MicroRSX V 3.0. The QIO function terminated by a special terminator, as specified in the documentation provided, caused the system to crash. When an additional dummy variable was added following the explicit declaration of the array size and name, the subroutine would run and not crash the system. A final problem was encountered which is common for all real-time and near real-time systems. The time needed to be set when the system went down due to power failures. A Hayes Chronograph was attached to each system to supply time.

## SUMMARY

Other than the above mentioned problems the systems have worked very well and have been extremely reliable. Only one service call was placed in over a year of operation in an office environment. The total cost of this system is less than the cost of a MicroVAX, a very good cost effective system for this project. This system has excellent growth potential within the PDP 11 family including upgrades to the PDP 11/83, and if a VAX is needed an upgrade to a MicroVAX II is also possible. The selection of the MicroPDP 11/23+ has proved a very good choice.

# ARCHIVING SYSTEM FOR RSX

James B. Jackson
Burroughs Wellcome Co.
Research Triangle Park, NC 27709

## ABSTRACT

An archiving system has been developed for RSX-11M-PLUS which produces two magnetic tape copies of user specified files for long term storage. Users may "mark" files for archiving at any time, explicitly or with wildcards. Optionally, files may be marked for automatic deletion after archiving. A batch program is run periodically which copies the marked files to a separate disk and spawns BRU to make the tape copies. It also prepares a directory entry for each archived file, and writes it into an RMS indexed file. A report program prepares printed sorted lists of files archived and error messages for files which were marked for archiving, but which were not archived for any reason. A "user friendly" directory search program locates files for recovery. The batch archiving process is controlled by an Indirect command file designed for use by a naive operator.

## INTRODUCTION

The Wellcome Research Laboratories are served by a custom designed local area network of laboratory microcomputers. The network is controlled by a PDP-11/70 running RSX-11M-PLUS. The major function of the microcomputers is acquisition of data from laboratory instruments, and analysis of the acquired data. The 11/70 acts as a file server for the microcomputers. In order to provide safe long term data storage, and assure compliance with federal regulatory guidelines, an archiving system was implemented for RSX.

The most desirable scheme would provide "instant archiving" of files under user control. The simple way to do this would require a disk drive dedicated to archiving. At the time of system design, this was not available. Consequently, the system was designed so that the actual copying of files from the users' directories is done periodically in a batch process.

## DESIGN GOALS

We wanted the users to be able to specify which of their files were to be archived, so an interactive user interface was needed. In order to provide adequate security, it was decided that two magnetic tape copies of the archives should be produced (for storage in separate buildings). A time stamped audit trail of the batch archiving process was wanted, partly for compliance with Good Laboratory Practice, but also as a reference in case of questions from users about archived files.

An indexed on-line directory of all archived files must be maintained. This was provided so that archived files could be easily retrieved. The directory entries include the date of archiving which in turn identifies the appropriate archive tape (or at most two tapes). We wanted the directory search program to be accessible to naive users, so that the users could do their own searches. This reduces demands on our time, and allows the users to "browse" through their archives.

The archiving process is summarized in Figure 1.

The WRL Archival Storage System consists of the following main parts:

1. The MACRO-11 program ARC, which provides the archiving system's user interface for the 11/70's direct acount users.

2. The user interface for microcomputer users, which is part of a microcomputer utility program called LOSM.

3. The MACRO-11 archiving programs, ARCHIV and ARCREPORT, which are run by the archiving indirect command program, ARCHIVER. These are batch process programs, run weekly after full disk backups. ARCHIV searches the users' directories for the files which the users have marked for archiving (by use of one of the user interface programs). ARCHIV makes one complete pass through all of the lists of files to archive before any file copying takes place. This is needed in order to determine whether the diskpack will be filled during the current batch archiving, and if so, which is the last UFD whose files will fit on the diskpack (see Figure 2). The program creates a 32 byte record for each user file which has been marked for archiving. These records serve as the on-line directory's entries for files which are successfully archived. These entries contain the file specification (ascii), the archiving date and UFD (binary), device and unit number (ascii), creation date (ascii), and a "warning code" byte. The warning code byte can have bits set to indicate that the file was marked for deletion following archiving, that the file had been revised or re-created after being marked for archiving, etc.

PROCESS SUMMARY

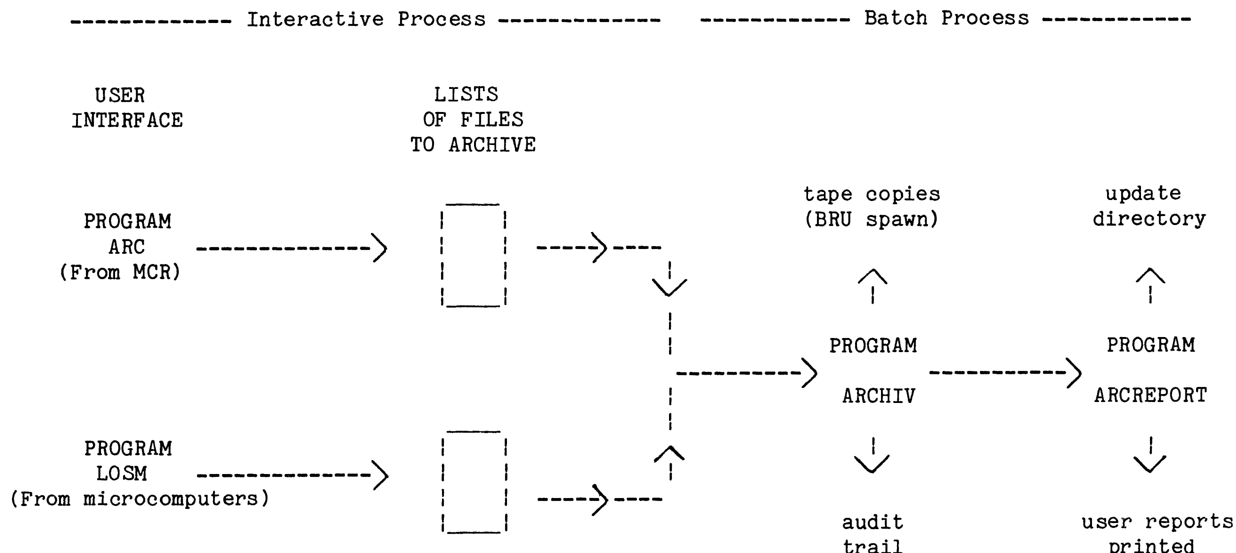



Figure 1

ARCHIV copies the users' files to the archive diskpack (preserving file attributes including creation and revision dates), and spawns BRU in order to make tape copies of the files for permanent storage. ARCHIV writes an archiving audit trail, with time stamps for file transfers and other critical events. The program also updates the file ARCHELPER, which contains a list of all the UFD's which have ever been sources of files which were archived. ARCHELPER also contains the cumulative block count for the current (unfilled) tape. ARCHIV outputs a sequential file of the 32 byte directory entries. The entries for files which could not be archived begin with a -1 (where the archiving date would normally appear).

4. The RMS indexed file ARCHIVDIR is the on-line directory file of all files archived. It is written and read by ARCREPORT, as mentioned above. In addition, a third MACRO-11 program, DIRSRCH, reads ARCHIVDIR. DIRSRCH provides the on-line directory search facility, which is needed to locate and recover archived files. DIRSRCH can also produce at any time a complete or partial printed directory of all files archived, sorted according to any of its four keys of reference. The four keys are (batch) archiving date, file creation date, file name (the nine or fewer characters part of the file specification), and file type (the three or fewer characters part of the file specification).

5. Program ARCREPORT writes the new entries to the directory file, ARCHIVDIR. The program writes the entries for files which could not be archived into a temporary file of "bad" entries. After all the new entries have been written to the directory or to the temporary file of "bad" entries, the report generation begins. ARCREPORT obtains a name and address for each active UFD from an EDT file. The name and address is printed at the beginning of each user report. This is followed by a list of any files which were specified for archiving from the UFD of the current report, but which could not be archived (and the appropriate error message). These file names are obtained by sequentially searching the list of "bad" directory entries for any belonging to the UFD of the current user report. The list of files which were successfully archived from the current UFD is then printed in alphanumerical order by filename. This is accomplished by reading the directory file sequentially on the filename key.

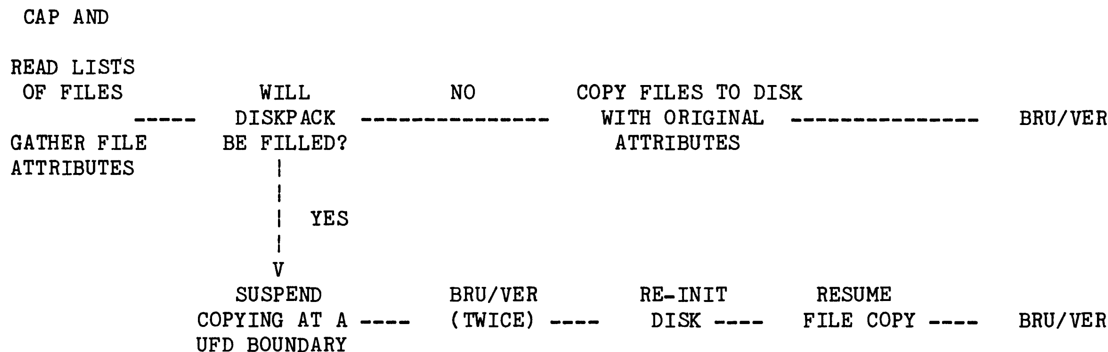
6. The MACRO-11 program LATREPORT can produce (or reproduce) user reports of archiving activity after a batch archiving is completed. LATREPORT is useful in case of printer failure, or a system crash which occurred during the printing of user reports.

7. Indirect command program ARCHIVER is responsible for running the batch archiving process. ARCHIVER performs the following functions:

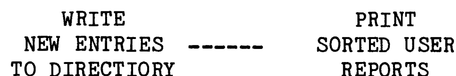
- o Checks that the terminal is logged in to the appropriate account.
- o Checks that the last batch archiving completed satisfactorily.
- o Checks that the archive disk drive and tape drive are dismounted.
- o The archive diskpack is mounted by its label to insure that it is the correct diskpack. The disk drive containing the archive diskpack is mounted unlocked (UNL) so that the index file can be modified by program ARCHIV in order to preserve the original file attributes with the copies.
- o The tape drive is mounted foreign (for BRU).
- o A one line test file is sent to the printer to test the printer. The operator is asked to confirm that the line was printed before ARCHIVER continues.
- o The operator is asked to confirm that the first tape is loaded and that the tape drive is on-line.
- o The BRU utility (which makes the tape copies) is installed.
- o The disk drive is set for writechecking to insure integrity of file copies.
- o ARCHIVER invokes PIP to delete the lists of files to archive from the last running of the batch archiving process. PIP also deletes the temporary file containing names of files which could not be

BATCH PROCESS - Indirect Command File ARCHIVER.CMD

PROGRAM ARCHIV




PROGRAM ARCREPORT




PREPARE DIRECTORY ENTRIES

WRITE AUDIT TRAIL FILE

BACKUP
UPDATED DIRECTORY
(RMSBCK)

Figure 2

archived at the last running of ARCHIVER.
o All temporary files from the last running of ARCHIVER are deleted. These files are used to provide "expandable buffer space" in program ARCHIV.
o RMS-11 utility RMSCNV is invoked to make a sequential backup file from the directory file ARCHIVDIR.
o Program ARCHIV is executed.
o The exit status from ARCHIV is checked. If a serious error occurred during execution of ARCHIV, an error message is displayed, and the indirect command program terminates.
o If ARCHIV completed normally, then the audit trail file is queued for the line printer.
o When RMSCNV has completed, Program ARCREPORT is executed.
o The exit status of program ARCREPORT is checked. If a serious error occurred, an error message is displayed, and the indirect commmand program terminates.
o When the final BRU spawned by program ARCHIV has completed, the newly revised directory file is backed-up to tape by the RMS-11 utility RMSBCK.
o The BRU utility is removed.
o The disk drive is set for no writechecking. and the disk drive containing the archive diskpack are dismounted.

**USER INTERFACE**

Program ARC provides the user interface which allows users to specify which of their files are to be archived. It can also display names of files awaiting the batch archiving process, and it allows users to delete names from this list. ARC is a privileged task (level 0) because it must be able to open files on a non-default UFD. It must be installed because non-privileged users cannot install priveleged tasks. ARC is an I and D space task, which enables it to read a whole directory in, up to 30 512 byte blocks. Larger directories are read in pieces. A large buffer is provided for the list of files to archive, which is maintained by ARC.

The ARC commands are:

o ARCH - Adds names to the list of files to archive, allows selection of automatic deletion of files from the user's directory during the batch archiving process.
o ARCDEL - Removes names from the list of files awaiting archiving.
o LIST - List names of files awaiting archiving, with date and time of selection for each file.
o HLIST - Send the list of files to a printable file.
o EX - Exit from the program.

PIP style wildcards (*,%) may be used in file specifications. Help files are displayed whenever "?" is entered as input.

**PROGRAM DIRSRCH**

The directory search program can search the directory on any of its 4 keys - ARCHIVING DATE, FILENAME, FILETYPE, and FILE CREATION DATE. The program is designed to search for entries belonging to a specified User File Directory. The user then

selects a key and a range of that key's values within which to search. Then the program requests a target filespecification, which may include PIP style wildcards. This scheme offers great flexibility in searching for particular files. DIRSRCH performs sequential gets on the search key, comparing each retrieved record to the desired filespecification. Matching records are counted and put into a buffer. If a large number of matching records is found (i.e. the buffer is filled), then matching records are written to a temporary file. When the search is completed, DIRSRCH displays the number of matching records, and inquires if the matching records are to be displayed on the screen or sent to a printer. The records include the archiving date, full file specification, file creation date, and any associated messages or warnings (such as "file marked delete-on-copy", etc.). DIRSRCH displays an appropriate help file whenever "?" is entered at a prompt.

## RECOVERY OF FILES FROM THE ARCHIVES

Full file specification(s) and corresponding archiving date(s) are obtained by use of the directory search program DIRSRCH. The archiving date specifies which archive tape will contain a file. Either of the two tape copies is located and mounted. BRU is then used to restore the file(s) from tape to the user's directory.

## SYSTEM RESOURCES NEEDED

Because programs ARCHIV and ARC are I/D space programs, and because supervisor mode libraries are used, the system needs RSX-11M-PLUS, and hardware to support these features. RMS-11 is used for the directory file, which was created by use of the RMSDES utility. One dedicated disk volume (fixed or removable) is required. We use a diskpack for an RM03. Obviously, a tape drive is needed, preferably one which can operate at 6250 bpi. In its current configuration, the system consumes a maximum of about 140 Kbytes of memory. At least 50K additional blocks of disk space are needed, mainly for the indexed directory file.

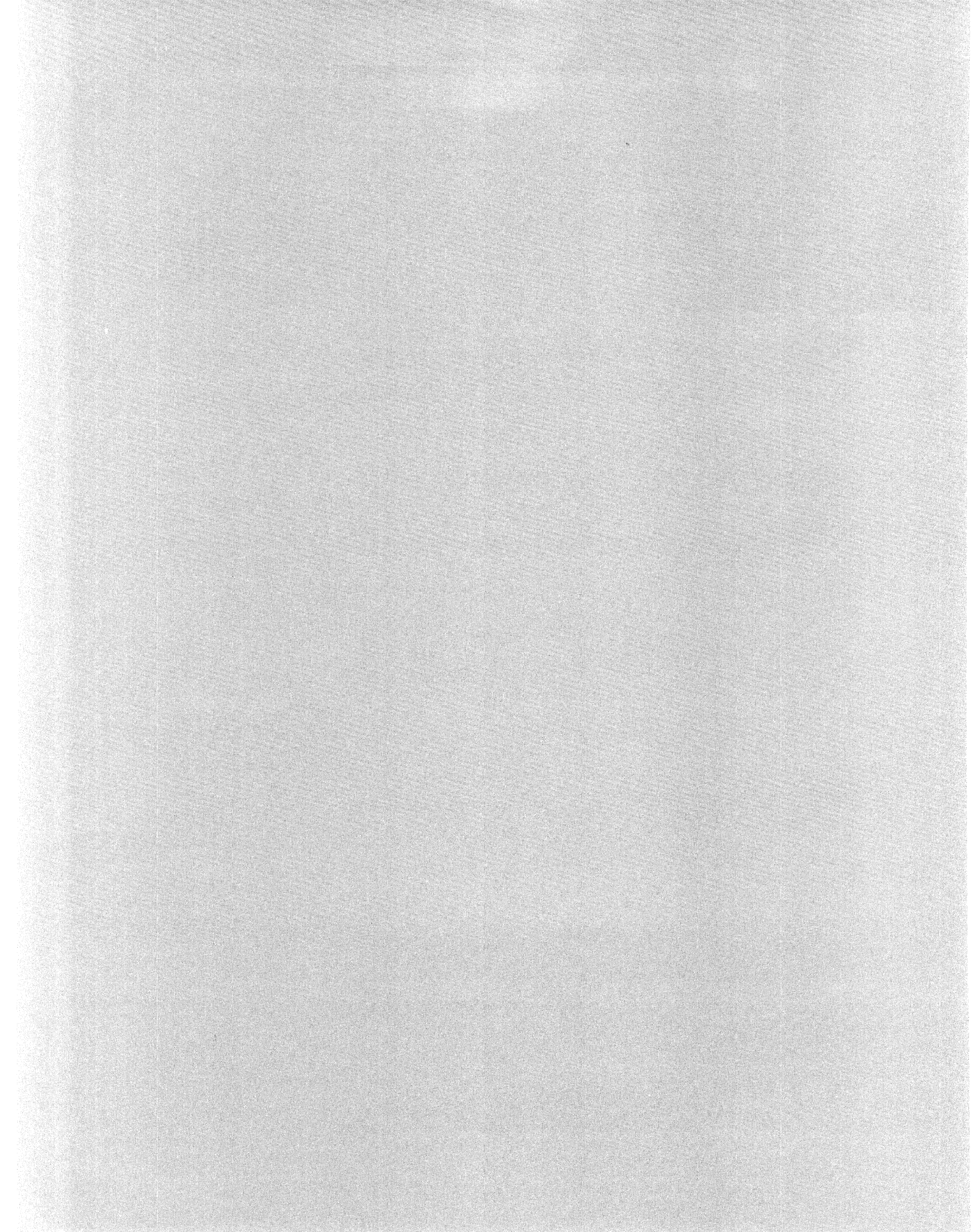
## SYSTEM UTILIZATION

Since the introduction of the system, over 137,000 files have been archived. We average about 1600 files per week, although more than 4000 files have been archived in one day.

## ACKNOWLEDGEMENTS

The author thanks Jerry P. Koontz for numerous helpful suggestions during the design and implementation of the system. Judy Hinderliter-Smith wrote the microcomputer user interface.

# SITE MANAGEMENT AND TRAINING SIG



# Organizing, Maintaining, and Distributing Software Products

**Peter Heinicke, Tom Nicinski,**
Penelope Constanta-Fanourakis, Donald Petravick,
Ruth Pordes, David Ritchie, Vicky White
Fermi National Accelerator Laboratory
Batavia, IL

### Abstract

The Computing Department at Fermilab develops and maintains software used at more than 30 different sites. A general methodology has been devised to keep track of and distribute the software at these different sites. Experience over the past year has proven the usefulness and efficacy of the method.

## Introduction

The Fermi National Accelerator Laboratory (Fermilab) is a facility dedicated to basic research in the field of high energy physics. This research takes the form of "experiments", which are conducted by groups of physicists. The experiments are highly computerized; there are usually one or more minicomputers devoted to the tasks of data acquisition and analysis of the experimental data.

Most experiments have at least one VAX or MicroVax computer, as well as one or more PDP-11 computers, and possibly various programmable microprocessors. There are many different experiments either actively taking data or preparing to do so at any one time.

The Data Acquisition Software Group of the Fermilab Computing Department provides software support for the experiments. The Data Acquisition Software Group's role is to develop a wide range of useful software for data acquisition and analysis. Experimenters use the software to perform the required online data acquisition and analysis for their experiment. In some cases, the software is used in a turnkey manner; more often, however, it is used as the basis for more elaborate and experiment-specific software. In the latter case, the experimenters obtain the basic package and then do their own software development to customize it to their particular needs.

Software is targetted for PDP-11 or VAX computers; the target operating system environment is RT-11/RSX-11M or VMS. Other targets are microprocessors, such as 68020's, etc. These target computers are located at approximately 30 different sites scattered over the 6800 acres of Fermilab. The VAX's and MicroVAX's at these sites are connected to one another via DECnet. These VAX's (or the Central Facility VAX Cluster) are used by the experimenters for software development in enhancing the supplied software as well as for online data acquisition and analysis. Software is transfered to these machines via DECnet from the Data Acquisition Software Group's Development VAX. It is also transfered via magnetic media to the computers not connected via DECnet. These include the PDP-11's (not connected mainly due to memory limitations) and the microprocessors.

Additionally the software may be transfered to the collaborating universities and research institutions which participate in Fermilab experiments. This transfer occurs so that the experimenter may continue software development activities for a Fermilab experiment while residing at the home institution or so that the experimenter may test apparatus under construction with components of the software intended for the experiment.

With so many sites and so much software in use at these sites, we quickly realized that some systemization of the task of organizing, maintaining, and distributing the software was mandatory. Keeping track of the software at the various sites, although a formidable job, is nevertheless a necessary one–we must be able to offer assistance with the current version of the software at hand.

A requirement on the systemization was that it must support having different versions of the same software at different sites or even at the same site.

While it might be possible in principle to arrange the same version of the software at all sites, in practice it does not occur. One of the most important reasons is that an ongoing experiment does not necessarily want to avail itself of the latest enhanced version of a piece of software; bugs or side effects may be introduced which might complicate the primary task of monitoring the experiment. Even when an experiment decides that the new features outweigh any risks of complication, it is extremely important that the experiment be able to switch back to the previous version as quickly as possible. The motivation may be to retreat from a software enhancement because it itself was found to have problems or because one wishes to rule out software changes as a cause of changes in the data being monitored. When the latter occurs, one then wishes to go forward again to the latest version.

This paper describes how we have organized our software development and support efforts to satisfy these considerations. In what follows, we describe the organization of our software into **Products**, how these Products are created, maintained and versioned, and how this Product organization is used in the distribution of software to the target VAX computers, and from there to other target computers when necessary.

## What is a Product?

A Product is an arbitrary group of logically connected directories and files (stored on a VAX/VMS system) and referred to by a Product name and optionally by qualifying names, such as the Version number, target operating system, or hardware interface. The Product name is a printable ASCII string describing the group in a mnemonic way. For each Product name there is a single development version of the product and/or one or more distribution versions. It is not necessary that a Product be developed by the Computing Department to participate in this scheme. However, the Product (the directories and files which comprise it) must be organized in a prescribed way. The constraints are relatively minor because we wanted the ability to include all kinds of software as products–not just those developed at Fermilab.

An example of a non-Fermi Product is KERMIT, a communications package. KERMIT_VMS is the Product name for the VMS version of KERMIT.

When the source code contained in the development version of a Product is updated, either for maintenance or enhancement reasons, a new **Version** of the Product is generated. This may occur even if the source code of the Product is unchanged. For example, if a Product is rebuilt using new "versions" of code on which it depends (such as an object library), but which is not a part of the Product itself, a new version of the Product is still generated. A Product version is used to inform the user, developer, and Product maintainer of not only which level of source code of the product it contains but also the entire state of the Product, its dependencies on other software Products, etc.

As a simple example of a Product with different Product versions, consider the KERMIT product for RT-11, where each version reflects the update level and the language.

Version V1.0 of the KERMIT_RT Product refers to the first Pascal version of RT-11 KERMIT. When the MACRO-11 version became available, the users needed to decide whether to keep supporting the Pascal version. If they had, they could have renamed it to be the **KERMIT_PASCAL** product (and call the other version the KERMIT_MACRO_RT product V1.0). Otherwise, they could have chosen to supersede it with V1.1 of KERMIT_RT.

Products come in two flavors: **simple** and **compound**. A simple Product consists of a collection of software which is expected to be used, upgraded to a new version, and distributed to target sites independent of the state of other software Products. The decision to organize a product as a **simple** one is basically that of the developer; it is a statement that this Product is somehow basic and not further made up of Products.

This does not necessarily mean that the Product was not dependent upon other software external to the Product when it was "built" (compiled, linked, etc.). Nor does it necessarily mean that the Product requires no other software Product in order to function.

For example, many of our Products are written in FORTRAN. These are definitely dependent upon the FORTRAN compiler and the FORTRAN Run Time Library–both of which are external to the product and which (in the case of the Run Time Library, at least) are required in order for the Product to function.

A compound Product is a collection of different "component" Products (either simple or compound), frequently used together. These Products do not necessarily have to be dependent upon each other although in many cases they are. They may be grouped together only for ease of distribution of many small Products which change infrequently. Alternatively, they may be grouped together because of dependencies on each other; hence, a change in a component Product would indicate that a new version of one or more of the other components is either necessary or desirable. DEC's ALL-IN-1 system is an example of something that is structurally similar to a compound Product.

## Goals

The Data Acquisition Group is primarily responsible for designing, developing, and maintaining software as well as supporting the end users of the software. The distribution and installation of the software is only a peripheral activity. To permit us to spend more time on software development, we have devised a Product Specification and specialised procedures, whose goals are:

- **Provide a Uniform Product Specification**

  The Product specification is meant to provide system management tools and the user with a uniform interface to the software we are responsible for. The specification includes

  - the directory structure of the files in a Product (defined to be a tree structure),
  - a list of required and optional files,
  - the naming conventions for these files and directories,
  - how logical names should be used.

- **Keeping Track of Product Versions on a System.**

  Different sites use different versions of a Product creating a need to maintain a database of which Products and versions reside on a particular system. This functionality is provided by a system management tool we call SITE_PRODUCTS.

- **Simplification of Product Distribution**

  We need to automate the distribution of versions of Products to remote sites (making use of DECnet) and the installation of the Products on the target site. Such automated procedures are needed both for efficient use of our time and to minimise the risk of errors or omissions.

- **Transportability to External Sites**

  Although restrictions are placed on a Products structure and interaction with users (how the Product is distributed and how the system manager treats it), it is still necessary to permit the Product to be easily installed and used on systems which do not follow our methodology.

- **Permit Switching Between Product Versions**

  In order to maintain and improve existing Products, and have the new releases accepted by experimenters, there is a need to allow the use of the latest version of a Product, but also to instantly and transparently "switch" to using a previous version residing on the same system.

  The ability to switch between versions on the same system is also important for Product developers and maintainers. A user may discover a bug at a previous release of the Product - and the Product maintainer is then able to check for the bug in that release just by switching to it. This capability is provided by PRODUCT_SETUP and the database of Products and their versions (maintained by SITE_PRODUCTS).

- **Permit the Composition of a Product to be Known Precisely**

  We make extensive use of DEC CMS (Code Management System) and MMS (Module Management System) to control the source code release level of a Product and to automate the construction of that product from its sources and any other libraries etc. it may be dependent on. However in situations where a Product may be dependent on libraries in other Products - the specific version of the library-related Products used must be both controllable and forever known. The time-stamps of the individual files as used by MMS are not sufficient to control such inter-dependencies.

  The procedures which we call BUILD permit the dependencies of one Product on another, either as a part of a compound Product, or just as a required but separate piece of software, which must be present in order to build the Product, to be expressed in a formal way. From this formal specification the order of creation of the component parts can be determined and the business of creating a very large software Product can be automated in a foolproof way.

**The Results**

All the management tools we have developed are written as DCL command procedures. DCL command procedures were chosen because of speed of implementation, and because we underestimated the full extent of the project we were undertaking.

The remainder of this paper will discuss the concepts and management tools introduced above which together allow us to achieve the goals outlined in the previous section. These include: Specification of a Product, use of the BUILD procedures, the SITE_PRODUCTS, DISTRIBUTE and PRODUCT_SETUP procedures.

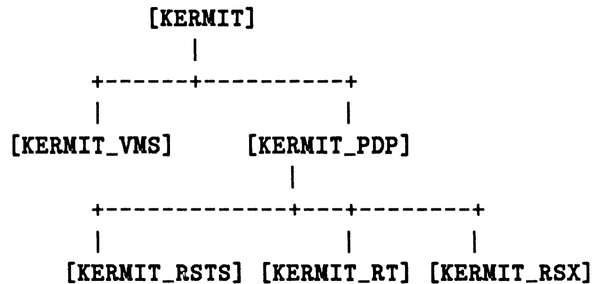
## Specification of a Product

The Product Specification provides system management tools and the user with a uniform interface to the software. We have written a 50-page specification of a Product including the mandatory and recommended requirements thereon. The Product Specification addresses three areas:

- Directory tree structure and the files in a Product.
- Logical names to be defined (associated with the Product).
- Required and optional command procedures and how they are used. (definition of parameters).

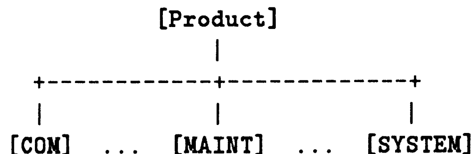
### Directory Tree Structures

Products reside under rooted directories. Actually, two rooted directories are associated with a particular Product. The **Version** Root is the rooted directory for a particular version of a Product. This is the rooted directory that a user will see when using a Product. Version Rooted directories reside under an **Umbrella** Rooted directory. The Umbrella Rooted directory contains all the versions of a Product. However, more than one Product and its versions can reside under the same Umbrella Root. For example:




The leaves are products, while [KERMIT] is the Umbrella Root. The Version Roots for the Products are [KERMIT_VMS], [KERMIT_RSTS], [KERMIT_RT], and [KERMIT_RSX]. The directory [KERMIT_PDP] is an intermediate directory, which could also represent a Product, or an Umbrella Root (the interpretation is up to the Product developer).

For each Product version, there is a set of required and optional directories:



```
              [Product]
                  |
  +---------------+---------------+
  |               |               |
[COM]  ...     [MAINT]  ...   [SYSTEM]
```

The [PRODUCT] directory is the Version Rooted directory, while [COM] and [SYSTEM] are required, and [MAINT] is an optional directory. Beyond these directories, the developer can use any tree structure (under the Product's version rooted directory).

### Logical Names

To keep Products site-independent, logical names are used to point to different files. All logical names should be defined in terms of one logical name:

```
'product'$ROOT
```

which is the rooted logical name pointing to the PRODUCT's Version Root. By changing 'product'$ROOT's definition (with PRODUCT_SETUP), a user can easily "switch" between different versions of a Product. In the example given above, the rooted logical name for HORSE is

```
$ SHOW LOGICAL KERMIT_RT$ROOT
   "KERMIT_RT$ROOT" =
   "disk:[KERMIT.KERMIT_PDP.KERMIT_RT]"
```

*Logical Name Tables*

Some Products require a large number of logical names to be defined. Ideally, users should only see logical names that they require, which implies that they should be process logical names. But, defining many logicals can be quite time consuming. The solution to make the logical names system wide was rejected for aesthetic and performance reasons, in that the system logical name table (LNM$SYSTEM_TABLE) would become cluttered as the number of Products grew.

Instead, shareable (system wide) logical name tables are created for each Product. When a Product is started up, it defines its logical names in the table created for it. This makes the logical name tables (and the logical names) invisible unless they are required.

Keeping a Product's logical names within its own logical name table keeps the system clean and allows for easy switching between logical names defined for different versions of a Product. It also helps when looking for all the logicals associated with a particular Product when you are on a large system with many logicals defined.

To use a Product, a user invokes PRODUCT_SETUP (described later) to "link" the logical name table into his/her logical name table search list.

### Required Files

The Product specification requires that each product provide two command files, of defined names, to be implicitly invoked at system bootstrap time and when a user wants to use the Product. All products must provide these files in a particular directory for the Product version. The specification also recommends a Help file to be provided with each Product; this is automatically included in the general Product Help library when the Product is entered into the SITE_PRODUCTS database.

COM SETUP.COM is used to define logical names and symbols on a per process basis. That is, the user invokes SETUP.COM (normally at login time) if there is a need to use the Product.

SYSTEM PRSTARTUP.COM is used during system boot time (Product startup) to define shareable logical names in the logical name table generated for the Product, and to perform any other operations which affect the Product system wide (such as INSTALLing files, loading device drivers, starting a queue, etc.) and other privileged initialization functions.

## BUILD and Developing the Products

The BUILD procedure is used to construct a Product based upon its dependencies on other Products. BUILD takes into account that a Product may:

- Depend on other Products.
- Depend on specific versions of other Products.
- Incorporate other Products totally within it.

The construction of a Product consists of compiling and linking the software comprising the Product.

A Product developer uses a Product Maintenance Language (PML) file to describe how a product is dependent upon other products. Only the immediate dependencies need to be described, since BUILD recursively uses the dependent Product's PML files to generate a final list (a Product Maintenance Output (PMO) file) which sequentially describes the order in which Products should be built (to satisfy all dependencies).

For example, the product KERMIT_VMS is to be **built**:

- KERMIT_VMS is dependent upon an another product called GET_PORT
- KERMIT_PDP is dependent upon KERMIT_RT, KERMIT_RSX, and KERMIT_RSTS.

BUILD would determine that the Products would need to be built in the following order:

```
GET_PORT
KERMIT_VMS
KERMIT_RT
KERMIT_RSX
KERMIT_RSTS
KERMIT_PDP
```

BUILD then will construct the Products in the appropriate order to generate the final Product. To save time, BUILD will not construct a Product if the required version already exists.

The actual details of construction of each of the component pieces are left up to the component piece of software. We normally use DEC CMS and MMS wherever possible. This is especially useful in conjunction with our methodology of one development version of a Product and multiple distribution versions. By having a single CMS library in the development version of each Product and creating classes for each source release level we avoid the need to keep the sources with or for each version of the Product. We can always recreate any version at any time. This saves disk space and also provides a centralized record of who changed the software and when.

## SITE_PRODUCTS - System Management of Products

SITE_PRODUCTS was developed to keep track of which versions of which Products reside on a system. It not only maintains a database of Products and their versions, but it schedules the starting up of Products at system boot time (or any other time) and the shutting down of Products. SITE_PRODUCTS avoids the need for the system manager to change the system specific startup command procedure (SYSTARTUP) every time a Product or a version of a Product is added, modified, or removed.

Products are made "known" to SITE_PRODUCTS (this should not be confused with the known files of the VAX/VMS INSTALL Utility). The **Known Product List** file, maintains this information.

For each known product, SITE_PRODUCTS maintains a **Product Version List** file which resides under the product Umbrella directory. The Product developer is able to add, modify, and remove Product versions without requiring privileges (only access to the particular Product's area is required).

The SITE_PRODUCTS procedures point to the Known Product List using a logical name. Users can use SITE_PRODUCTS to maintain their own Known Product List, and Product Version Lists. This can be extended for use on a VAX Cluster system, where a common Known Product List is used to startup (shutdown) all Products common to all nodes in the Cluster. Then, by redefining the logical name, a node-specific Known Product List can be used to manipulate software Products licensed (or useable) only for that particular machine.

SITE_PRODUCTS allows the addition, modification, and removal of Products and Versions. These operations only modify the Known Product List and Product Version Lists, not the actual files of the Products. When a Product version is declared to be the default version on a system, its Help file is included in a general Product Help library (if one exists) and also a Bulletin is posted on the system (if the Bulletin Product is available).

For each Product, the Known Product List maintains the Product's name, the specification of the Umbrella Root, and other miscellaneous information. Associated with each Product version in the Product Version List is a directory path from the Umbrella Root to the rooted directory for the Product version.

When a Product is started up by SITE_PRODUCTS, a shareable (system wide) logical name table is created to contain logical names defined by the Product. Then the Product specific startup command procedure is invoked. This procedure usually defines logical names, device drivers, starts up queues, installs privileged images, etc.

The final stage of any Product is its use. PRODUCT_SETUP is used to "setup" a product for use by a user. It also allows a user to choose which version of a product to setup. Setting up a Product involves the definition of logical names and symbols required for using the Product.

A symbol by the name of SETUP is used on all systems to invoke PRODUCT_SETUP. Users of a software Product such as our example KERMIT_VMS simply type

```
SETUP KERMIT_VMS
```

to use the default version of the Product and all its component sub-Products.

The ability to switch transparently between Product versions is provided by the logical name tables created for the Product. When switching between Product versions, PRODUCT_SETUP creates a new logical name table (which overrides the old table) and defines the logical names for that particular version. Therefore, different Product versions are not required to use the same logical names.

## DISTRIBUTE - Distributing the Products

DISTRIBUTE provides a system manager on a remote machine the ability to copy Products, from an "Archive machine", and install them. Most of the time, DISTRIBUTE is used over DECnet, but it also provides a tape mode, which permits Products to be distributed and installed at external sites using Magnetic tape as a transfer medium.

DISTRIBUTE interactively queries the user for the information it needs. The questions are self explanatory, so that no documentation is normally required in order to obtain a Product. Besides the Product name and version, DISTRIBUTE asks where the Product should be placed (the disk and Umbrella Root), and whether the Product and its version should be declared to SITE_PRODUCTS.

When a Product is selected by the user, DISTRIBUTE uses that Product's Product Maintenance Output (PMO) file (generated during a BUILD) to determine which Component Products need to be copied over as part of the chosen Product. This provides all sites with a complete and consistent view of a Product. Products which are not constructed with BUILD and therefore have no PMO file can also be distributed - all files in the directory tree stemming from the Product version rooted directory will be taken to comprise the Product version.

DISTRIBUTE uses BACKUP save sets compatible with the VMSINSTAL utility (part of VAX/VMS). Because the Product conforms to the Product Specification, only one KITINSTAL file (used by VMSINSTAL) needs to be written for all Products. This frees the Product developer from writing code used strictly for the purpose of installing a product.

A complete log of software distributed, date, version and to where is maintained on the Archive machine.

## Conclusions

The organisation of products and the procedures described in this paper have been in use for more than a year now. Hundreds of Products have been distributed to target sites. The sacrosanct nature of a Product version once built has enforced a strict discipline on program development and aided immensely in tracking down complicated problems where any one of a number of hardware and software variables could have been at the root of the problem. The procedures described were first developed for software to be executed on a VAX(VMS). We have found them such a useful aid for distribution, maintenance and archiving that we extended the concepts to cover software for other operating systems in use.

We have found the Standard Product specification to be extremely useful. Not only has it enabled us to write the management tools described but it has also helped enormously in the ease of understanding, maintaining and supporting our software. New members of the group and new users new to Fermilab can very quickly produce software to conform to the general specifications and obtain and use software that is available. It is much easier for any member of the group, regardless of particular area of expertise to be able to distribute, demonstrate, find bugs in, create a new version of any Product. New software Products produced elsewhere at Fermilab or at other institutions or vendors can be quickly added to the set of available software and made available in the same uniform way to all the users on site (via the same SETUP command). We package all software according to our minimum standards - give it a Product name, a version, keep all versions under a single Umbrella directory and define all logicals relative to a single root logical name pointing to the specific Product version. Following software Product "standards" has saved manpower also in enabling us to write general procedures. For example, the arrival of Microvaxes with lim ited disc space created a need to trim Products. A general procedure which omitted all list, map and documentation files from a distribution version could be written because of the standards imposed, thus solving the problem in general for all software which we maintain or distribute.

This entire program of work was undertaken without a proper realization of the size of it - really as a non-serious sideline, which people did a little work on when the need arose. If we were doing it again we would better understand the benefits and scope of the project and would take it further than we have today. The database maintained by SITE_PRODUCTS would be made extensible and easily accessible as a database. Some of the system management procedures would have been written in a high level language instead of DCL, thus increasing both their speed and extensibility.

## Acknowledgements

Contributions to the ideas, definitions and procedures have been made at various times by all members of the Data Acquisition Software and DEC Systems Group in the Computing Department at Fermilab - which consist of the authors, David Berg, Eileen Berman, Andy Cohen, Terry Dorries, Arkady Lubinsky, Carmenita Moore, Liz Quigg, Dave Ritchie, Chip Kaliher, Nancy Hughart and Steve Kalisz. We also acknowledge helpful feedback from various users of the system (DISTRIBUTE in particular) ranging from on-site local system managers to experiment participants distributing software over DECnet from Italy.

## References

[1] Aurbach, Richard, *Using VMSINSTAL with User-written Applications*, Fermilab Programming Note 262.

[2] Constanta-Fanourakis, Penelope, *BUILD Procedure for Product Distribution*, Fermilab Internal Note.

[3] Heinicke, Peter, *Backup / Distribute Procedure for Product Distribution*, Fermilab Programming Note 261.

[4] Nicinski, Tom, *SITE_PRODUCTS / Maintaining Known Products*, Fermilab Internal Note 140.

[5] Nicinski, Tom, *BULLETIN / Maintaining an Electronic Bulletin Board*, Fermilab Internal Note 141.

[6] Nicinski, Tom, *PRODUCT SETUP User's Guide / Setting Up Products*, Fermilab Programming Note 269.

[7] Pordes, Ruth (ed.), *Data Acquisition Software Group Product Specifications*, Fermilab Internal Note 157.

# Developing a Computer Training Program for a DEC/IBM Environment at Dupont

**Marlys Denison**
E. I. Dupont de Nemours, Inc.

## Introduction

When I was given the task of developing a training program for the VAX 8600 I was afraid that training issues would get about the same priority as the annual employee picnic. Generally this is no longer true at Dupont. At our plant a certain number of hours of training are suggested for various types of employees and the training choice is left up to the employee and his immediate management. Therefore we began by interviewing several potential students to get an understanding of the need.

I noticed that the IBM-PC users thought that the people with DEC terminals were at great disadvantage and the people with DEC terminals felt the IBM-PC was somewhat unprofessional. The 3270 users didn't seem to acknowledge that any other terminal existed. These biases were almost religious in nature.

When I asked a young DEC programmer what he thought of the IBM equipment he said, "The best thing to do with an IBM mini or mainframe is to sell it. I don't like the way they work. Menu driven software is too slow, it gets in the way of a programmer."

When I asked an older IBM-PC "power" user why he liked his computer better than a terminal, he said, "The software is so much better and less expensive, the effect on the economy is the same as that of the Ford company when they started mass producing cars. There are many more people writing programs for the IBM-PC than any other computer. For the same reason that a larger University can field a better football team, the software for the IBM-PC is easier to use, and less expensive for the function than any other computer."

A young IBM systems programmer said, "I don't even consider the VAX to be a mainframe in the sense of the IBM mainframes."

One of the men whom I work with feels that the fact that, most of the end-user equipment on our plant is of these three types, didn't just happen. "At the time that we were changing from tabulating equipment to an IBM mainframe, IBM seemed to offer the best business environment. At the time that we were beginning to monitor the processes with computers, IBM had discontinued their 1800 equipment in that area and DEC was the best equipment for this use. Both were good choices and have benefited the company."

The bottom-line is that we have made the best decision at any point in time. We have millions of dollars invested in each kind of computer hardware and more than that invested in training to use the computers.

## Purposes of Training

Just plugging in the equipment is not enough. It takes years to learn how to use it most effectively. The cost of computing is high but the cost of not computing is even higher and time is a factor.

Gilbert/Commonwealth conducted studies to determine productivity gains among its engineers. The company found 50 to 70 percent reduction in the time needed for certain tasks. They found that the machines assigned to support and non-engineering areas are less profitable but still make back the investment before the equipment is fully depreciated. (*PC World*, August, 1986).

Allied Stores installed a computer system and saw no improvement in productivity for three months but after three years the productivity had doubled.

At the Federal Kemper Life Assurance Co., a large insurance company, productivity increased by five times. This required a change in procedures to best use the equipment.

According to Raymond E. Cairns, Jr., head of the Information Systems Department at Dupont, the computer can increase the span of control and enable elimination of layers of management. (*Fortune*, May 26, 1986).

Besides the productivity improvements, the employee will recognize the training as evidence that the company is serious about its commitment to individual development. This can result in well-motivated employees rather than semi-conscious dead wood that couldn't get a job anywhere else.

## Purpose of the Center

The purpose of this center is to develop the computing skills of technical employees and their support group so that they can use the computer to think, compose, control and communicate. The minds of these people are the company's most valuable resource.

Competence depends on training and education. Training equips someone to do a job, it is task oriented. Education, however, strives to improve our ability to understand, discover and relate our tasks to our long term goals. We try to both train and educate in our training center.

If you have only a hammer, everything looks like a nail. Likewise, could it be that if you have a personal

computer, everything looks like a personal computer application and if you have a terminal to the VAX everything looks like a VAX application? To some extent this is true, therefore by merging the training for these two types of equipment in the same facility, we are able to better determine the relative merits of each piece of equipment in an educated way.

We are also more likely to develop communication facilities and procedures between the types of equipment and software. This leads to standardization

We constantly complain of lack of standardization. Large national and international committees work to bring standards to the world of computing.

Only months ago the DEC and IBM equipment on our plant could not communicate. Now they can all communicate at some level, some better than others. What is true today will not be true tomorrow.

I applaud the effort to standardize but complete standardization stagnates progress.

If we had only IBM equipment or only DEC equipment over the past 20 years we would not have made nearly the progress technically that we have. One current example of this is the sudden technical advances made at the end of the breakup of the telephone companies. Now we have many more communication and end user choices at the expense of what sometimes seems to be total confusion.

## Facility Design

Few of us are given unlimited budgets and freedom to design a training facility. In our case the facility was to be designed within a large room with one fixed pole in the center of the long way of the room and about four feet from one side. The pole had to be considered a challenge. We decided to face the students towards the side of the room without the pole. This gave the shortest maximum line of sight and avoided the pole.

First we visited four training sites in the company. Each one had different ideas. This created some valuable contacts which I used later when I needed help with finding instructors or documentation. From all the information that they gave us we decided what would be right for us within our budget.

Lighting is an important consideration. To have light low enough for projection while also bright enough to see notes is nearly impossible. Egg crate diffusers on the lights help point the light down and not on the projection screen.

A raised floor would have been nice but could not be included in the budget, so we settled for flat electrical wire and carpet squares. This has the added advantage of looking nice and damping the noise level in the room. Wires go directly down to plugs in the floor. The excess is held up under the edge of the table with special holders which affix to the side of the table.

The classroom has board writing area which is glare free and two paper tablet writing areas. Projections are visible from all parts of the room.

A printer need not be provided with each terminal or PC if the machines can be networked. A laser printer for printing course material was available.

Storage space is needed both in the classroom for materials to be used and in a library for the materials used in the programs. This needs to be secure but accessible by the users. We have two oak credenzas in the classroom which match the oak top tables and oak visual board. Next to my office there are more shelves for storage space. We could use much more storage space but space is always at a premium.

Usually we have classes of ten so we have ten chairs with adjustable height,back and arms. We also have ten matching spring type stacking chairs for the times when we have classes two to a computer or for the student who likes a wider chair. A sloping-front edge which does not interfere with circulation is necessary and a five-star base with casters makes the chairs stable.

The work tables in our center are sixty inches wide which allows an IBM-PC and a DEC terminal per table. This saves any moving of terminals in and out for different classes. Often students studying the same software sit side by side in the same class using different types of equipment. There are ten tables with two pieces of equipment on each table. There is room to spread a work book out on each side of the computer. Since the chairs are adjustable in height, the height of the tables are one standard height. This was designed to teach either ten IBM-PC users or ten DEC terminal users per class but, as mentioned, we found that in some classes we have both types of equipment being used at the same time. A few students are familiar with both types of equipment and will use whichever one is convenient.

## Providing Different Types of Training

We are continually trying to provide the best classroom training available for our employees.

There is the argument that people who are always in training never contribute any real work. Not everyone wants or needs classroom training. One possibility is self training. Most people with initiative are continuously learning on their own. This requires references or materials. While self study is important it is not efficient in providing a complete background. Programs to help the student uncover what is relevant and use the experience of others is useful. There is not time enough to invent every wheel yourself.

Some students find the convenience of working on computer-based instruction when they find time is the best way for them to learn. Others like to check out Video-tapes to watch at work or at home. The more people who become trained in an area the more one-on-one networking type instruction takes place.

## Quality of Training

In-house training can provide the most cost effective method of training. Quality of materials and teachers are perhaps the most important part of the training effort. Whether an instructor costs $3,000 per day or $500 is of little importance. The most important question is how effective is the instructor. The student's time used for the class represents more than $4,000 per day in the average class. If each day is not effective the cost of wasted student time is added to the cost of the instructor. There is the concern of the interruptions on site, but the ability to address emergency needs, answer electronic messages during breaks, and generally keep the onslaught of piled up work from engulfing you while in training outweighs the problems. Many times we offer a series of half day classes for this purpose.

At our facility, we don't have a teaching staff. This is a problem in some ways but a blessing in others. Any one person can only be proficient in a limited number of software packages. We have the choice of the very best teachers of any software package.

Finding the best is mostly a matter of trial and error but there are some very good teachers in every area. I've found the best teachers are also users and the cost per day does not measure their worth as teachers. Talking to former students, looking at class materials, and talking to the teacher about what you expect are helpful in locating good teachers.

Some of our teachers come from other training organizations within the company. They are usually of very high quality and have the added benefit of the company perspective.

A few teachers are users from our site. These people usually do not have time to take from their regular job to either teach or prepare to teach. They work out best for short courses on site specific communication or areas in which they are especially knowledgeable.

My worst experiences have been in hiring the relatives of co-workers or from companies where the trainers only train and don't use the software, or the first class when developing my own course.

## Applications

Word processing is probably the most used application anywhere. To many of us, memos, reports and personal notes have taken on a whole new dimension with the use of the computer. Mistakes can be corrected instantaneously, ideas can be rearranged in more logical order with a few strokes. Formatting can be done automatically and changed automatically. We have MASS11 on the VAX 8600 and on many of the secretaries PC's. The idea is to make it easier to pass information from the technical person to the secretary. We have not insisted on the MASS11 software, and in fact would be willing to train on Displaywrite or Word Perfect if requested. One secretary learned four word processors in four years but it became easier each time because of the similarities. Standards will occur as the need develops.

Electronic mail is the best thing since bottled beer. Not only is communication speeded up but it is improved. The receiver does not need to be there when the message is sent. He can receive the message at his own convenience. It is more accurate than speech and leaves an audit trail to help you remember what you need to do. It has the added benefit that you can pass on some of your work as quick as a wink and keep a record of that as well. The training for this is included in a three half day introductory course covering also the editor and some simple VMS commands and utilities.

Database is used more than most people realize. There are the large corporate databases which follow the cost of manufacturing etc. There are also personnel data bases on the corporate level. Production build a large databases which the Engineers use to improve the product. These are in large custom applications. The very small databases on the other hand usually reside in Lotus or Dbase on personal computers. There are new easier to use database application programs now on the market and these old standbys are making improvements regularly. DEC of course has several types of database programs. We are currently training on Lotus, Dbase, Datatrieve, 20/20, RS1 and the custom programs.

Spreadsheets are really a special kind of database. Lotus and 20/20 are the favorites with some people using DECalc and RS1. Lotus and 20/20 have a much shorter learning curve than the more sophisticated RS1.

Statistical packages are used in many areas. This function is found in Minitab and RS1 on our plant. Minitab is taught in a class on Data Analysis which covers the subject and uses the package only as a tool.

Expert Systems are the new interest at our plant. There are two personal computer packages which are so easy to use that the only instruction takes place in a two day Artificial Intelligence class that also covers the use of RS1 as an engine.

Chemical Modeling is taught by an expert from our corporate office. He uses several packages such as CPES6, PROCESS, UPPS, and ASPen.

We currently have an Integraph System for drafting. It is centralized but it can now be accessed from the field. Robotics in the production line is an area that will be developing in the near future.

## Documentation

Documentation is no good unless it is used. It is not used unless it is easily referenced. Finding the right documentation to meet the student's needs difficult. Better documentation can be produced than is available the cost is usually prohibitive. Some documentation needs to be produced for those things that are site or corporation specific.

A newsletter or bulletin board on the computer is helpful for distributing new items concerning training classes or tips about using the various pieces of software.

The schedule should be published both in hard copy for those who are just beginning to use computers and on the bulletin board.

Even though good emulators are available for personal computers, good documentation to translate from the keys of the VAX to the keys of the IBM PC is not available. In the beginning VMS/Mail/EDT classes it is necessary to produce documentation to make the transition easier for the student. Overlays can be helpful to remember what the function keys will do but they are not generally available for the IBM-PC.

## Conclusion

We have found that by combining the training for the two types of equipment in one facility has helped to standardize the plant on the equipment that is best for the job.

Competition between the two types of users helps us determine the best way to do a job rather than forcing us to do the job with the hammer at hand.

We can consider in a rather dispassionate way the relative merits of having backups done on the VAX versus always having a standalone capability and complete security of information.

We can determine if a person has the ability and interest to use a personal computer's special advantages or if that person only wants the easiest possible way to communicate.

We can offer the capability of large capacity on the VAX or the ease of use of the application programs on the IBM-PC.

Every day the picture changes. When I was learning to program, many hours were spent writing a program that would take as little memory as possible. Today usually only systems programmers are concerned about space in memory.

What is true today is not true tomorrow.

Today the PC user is responsible for backups, tomorrow it may become very automatic.

White collar workers account for about three-fourths of total business payroll costs in the U.S. Only about 10% of business computers are used in manufacturing.

We still have many problems to solve. The IBM mainframe is accessed fairly easily by the DEC terminal users on our plant but we are only beginning to develop methods to access the VAX equipment with the IBM 3278 terminals.

When the terminal setting on the desk is best for ninety percent of a person's work and is adequate for the other ten percent we feel he or she has the best piece of equipment.

A maintenance worker may need a 3270 for most work order or personnel type work but he needs to be able to access the communication system in the VAX and perhaps the drawings system in the VAX.

The production worker needs the ability to monitor and control the process which the smaller Digital systems offer but also needs the communication on the VAX and portions of the business computer.

The Engineer needs the technical software available on the VAX but also needs the data from the process computers, the ability to change drawings on the VAX, communication on the VAX and the ability to write work orders on the business computers.

A new use for computers is the development of expert systems. The expert needs to be able to develop the expert system on a VAX or on a PC and the worker needs to be able to access the expert system.

People are different and different kinds of equipment suit different people. Therefore Viva La Difference!

# COMPUTER ROOM DESIGN AND CONSTRUCTION: A CASE HISTORY

Brent Teeter, P. E.
Naval Weapons Center
China Lake, CA. 93555

ABSTRACT

Once a computer system has been purchased, the anxious optimism of waiting for it to arrive will fade into the realization of where to put it when it arrives. By considering the computer system site early in the procurement process, headaches can be reduced for system startup, system operation, maintenance, and expansion.

## INTRODUCTION

Good computer room design practice entails addressing six concerns: setting up the design team, physical requirements, electrical requirements, cooling requirements, security, and contractor interactions. With proper attention to these needs, a well designed computer room will provide a reliable environment for the computer and will improve work performance of the people that use it.

## GETTING STARTED

There are a number of concerns that should be addressed before the design gets underway. The first concern is setting up the design team. People who have a vested interest in the results should be used to assist in the design and review. An example of such a person is the system manager. He will have to live with the computer room that results from the design. Other good sources of people are those who have shown an interest in the project, who have good memories, and who have the time to pay attention to the construction. A third source of help is DEC Field Service. Field Service can provide specifications and requirements for much of the computer room.

Once the design team is established, team members should keep a historical record of all interactions with contractors and consultants. This record may be useful in the future if performance problems occur.

When the computer room design is finished, it should be reviewed by at least two knowledgeable people outside of the design team. This review is necessary because people who are intimately involved in construction designs sometimes miss details. Finally, due to the high time demands placed on design team members, it may be advantageous to hire a consultant. However, consultants do not have the vested interest that employees have.

## PHYSICAL REQUIREMENTS

The first task in designing a computer room is to determine the size and weight of each cabinet and peripheral that will be placed in the room. Adequate room for growth should be allowed as well as clearance to allow rear doors of cabinets to be opened.

The easiest method of determining equipment placement is to make a floor plan using a convenient scale (ie. 1/4 inch equals 1 foot), cut out each peripheral and cabinet floor footprint to the same scale, and place them in the floor plan until all specifications have been met. While laying out equipment locations, it is important to consider workflow. Workflow considerations make users of the computer room more efficient. An example of workflow is to group console terminals of multiple machines together so that the system manager can use them with a minimum of movement and effort.

An important benefit of using a floor plan is that when the computer room construction is completed, the plan can be used to accurately locate equipment in the room. Each cabinet can be located by taping out the location of it.

If the amount of equipment in the computer room justifies it, a raised floor should be considered. Raised floors have several benefits, among them being:

* The floor acts as an air conditioning plenum.

* The floor aids in the natural flow of convective cooling air.

* It protects data and power cables from damage.

* Raised floors are cleaner than non-raised floors.

However, raised floors have concerns that must be addressed:

* How do heavy cabinets/peripherals enter and exit the room?

* Air distribution can be blocked by having too many pipes and cables under the floor

* The locations of cables that exit and enter the floor in relation to other equipment must be determined so that they will not be blocked.

* All concrete and drywall must be sealed with concrete sealer to prevent blistering.

Another physical computer room requirement is that the room be treated as an environmental entity. The walls of the computer room should go from the floor to the roof of the building, effectively separating the computer room from the building.

Since computer room noise control is an important consideration, effective steps to reduce noise are necessary. These include using static free carpets, sound absorbing materials on the walls and separating the computer room into noise zones. Equipment that is noisy is grouped in one zone and quiet equipment is grouped in another zone.

## ELECTRICAL REQUIREMENTS

In order to determine the electrical requirements for the computer room, these specifications for each cabinet and peripheral are needed as well as their tolerances:

* Volts

* Current

* Phase

* Plug type

* Peak Power (Peak Current)

The final specifications should also allow for growth since once the wiring is in place it can be expensive to increase capacity.

All power receptacles for computer equipment should be isolated ground type sockets with grounding occurring at a central point. This central point grounding minimizes ground loops which can induce noise into the system. The local electrical code should however be examined about regulations concerning isolated ground sockets - some municipalities will not allow them.

All power lines feeding the computer should be dedicated to the computer. There should be no other electrical equipment on the line. The main feed line for the computer room should be checked completely from the distribution transformer of the building to the computer room for other noise producing equipment that might affect computer operations. If there is any doubt about the quality of the power, a power line monitor can be used (rented or purchased) to check for disturbances. This monitor should be allowed to run for as long as possible since some power line disturbances are season dependant. A good example is the summer thunderstorms that occur in some areas of the country.

If a power line monitor reveals noise and power problems on the electrical system, the following solutions can be tried in order of increasing severity:

* Filters - low cost and easy

* Constant Voltage transformers

* Motor - Generator Sets

* Uninterruptable Power Supplies - High cost and difficult

In any computer room electrical system, it is very helpful to use a Power Distribution System (PDS). These systems provide some filtering but mostly provide isolation. Newer PDS systems, called Power Conditioning Systems (PCS), provide substantial filtering. PDS/PCS systems are useful because they modularize the electrical distribution and installation process. There is only one connection that a licensed electrician must make, thus speeding up the installation. When purchasing a PDS it is best to pick the one with the highest input voltage available. This provides greater noise reduction than using lower input voltage PDS units.

In addition to receptacles for computer cabinets and peripherals, convenience outlets should be included in the design. These are the electrical outlets that will be used for vacuum cleaners and other noise (electrical) producing equipment. Because of this noise, these receptacles should be placed on a different feeder line than the PDS.

Closely related to electrical requirements are lighting requirements. Generally, it is very desirable to use light dimmers in the computer room. These can decrease the heat load placed on the air conditioners. However, some dimmers are Radio Frequency Interference (RFI) sources. For this reason, the particular brand of dimmer should be carefully examined for RFI before it is installed.

## ENVIRONMENTAL REQUIREMENTS

In order to determine the environmental requirements for the room, the BTUs of heat for each peripheral must be determined. The total heat load produced by all electrical equipment including lights must be capable of being cooled by the air conditioning equipment. As always, the environmental specifications should allow for growth.

The temperature limits for each peripheral must also be known. All peripherals have two types of limits: static and dynamic. Static limits establish the overall range in which the equipment can operate. Dynamic limits specify how fast the temperature can change per unit time (usually in degrees per hour). Generally disk drives have the most critical dynamic limits because read/write head alignment depends upon uniform temperature throughout the drive.

The air flow direction for each peripheral and cabinet should also be known. This information will determine where to place raised floor vents (if used) and determine whether certain peripherals and cabinets are compatible. The usual flow direction is front to back and bottom to top. The raised floor vents can then be located in order to assist this natural flow of air.

When specifying the air conditioning units, it is usually better to specify two small units rather than one large unit. Thus, if one unit fails, the computer facility can still operate in a degraded mode. Likewise, the larger the computer room is in volume, the more time there is to shut down the system when the air conditioners fail.

In order to minimize contamination, there should be a source of air that will maintain a positive pressure in the computer room. This positive pressure will tend to push dirt and dust out of the room. If a raised floor is used, the concrete slab and drywall underneath it should be sealed with concrete sealer to reduce the number of particles that are produced as the concrete ages.

## SECURITY REQUIREMENTS

In choosing the location of the computer room, careful attention must be paid to physical security. Security involves room location, fire suppression, electrical noise, and protection instrumentation. If possible, pick an interior room. Interior rooms are more temperature stable than exterior rooms. They are also less susceptible to external electromagnetic interference (EMI). However, if self contained air conditioners are planned for the room, a room with an external wall(s) becomes necessary.

Due to the high value equipment in computer rooms, all computer rooms should be protected from a potential fire. Smoke detectors should be installed in the room-generally under the raised floor. However if a raised floor is not used then they can be installed on the wall.

Handheld fire extinguishers should be placed near the computer room exits. Thus, if a fire occurs, people looking for fire extinguishers will already be near an exit should they change their mind about fighting a fire. These extinguishers should be filled with Halon 1211 or 1301. Halon 1301 is less toxic to humans than 1211 but both halons are excellent fire suppression agents. For large computer rooms, under floor self contained halon systems are available.

Ceiling sprinklers are another method of fire suppression. However, since most damage in a fire occurs from water damage and since there is a high electrical shock hazard in a computer room, ceiling sprinklers should be used as a backup to Halon systems. Also, sprinklers should be used that can turn themselves off when the computer room temperature decreases to a set value so that flooding does not occur. Except for ceiling sprinkler pipes, water lines in the ceiling should be avoided. At minimum, they should be kept away from equipment.

Another security concern is electromagnetic interference (EMI). EMI can occur from many sources including welders, motors, heavy industrial equipment and even other computers. The solution to EMI is usually to tie all equipment to a common ground, move equipment away from the source (since EMI strength is proportional to the square of the distance), and surround the computer with copper screen.

Once the security issues have been addressed, different types of detectors can then be interfaced to the computer room Power Distribution System (PDS). These sensors connect to the PDS through the Building Interface Alarm box (BIA). Some of the detectors that can be used are: smoke, fire, water, over/under temperature sensors and over/under voltage sensors. If any of these sensors detects an out of bounds condition, it will trigger a power shutdown of the PDS.

A last concern for computer room security is environmental data gathering. Instruments such as temperature and humidity recorders provide a record of the stability of the environment. Other instruments can provide data on other desired data such as voltage levels.

## CONTRACTOR INTERACTIONS

In order to protect the company and the computer room design team, all interactions with the contractor should be conducted through one specific contact. There should also be an alternate contact to serve as a backup should the primary contact be unavailable. Both contacts should be aware of what they legally can and cannot do regarding the construction contract. Other people in the company, while not designated as contacts, can serve very usefully as eyes and ears during the construction process. In this manner, they can keep the construction contacts appraised of information they might not normally know.

In the construction contract, there should be penalties for late completion of

the work. If there are not penalties, construction may drag on for an excessive time.

Finally, during construction of the computer room, disruptions to normal business can be minimized by scheduling the contractor to work at times convenient to the company.

## SUMMARY

Computer room construction requires attention to a very large number of details. If motivated people are used on the design team, and these people have access to the proper information, then a successful computer room design will result.

## REFERENCES

1. Digital Equipment Corporation, Power Distribution System Technical Guide, 1982.

2. Digital Equipment Corporation, The Power Distribution System Configuration and Ordering Guide, 1981.

# VAX SYSTEMS SIG

# Effective use of VAX/VMS Autogen

**Dennis L.W. Thury**
Texas Instruments
McKinney, Texas

## Abstract

The functions of AUTOGEN, and the inter-relationships of the major SYSGEN parameters are presented. The resulting suggestions for the structure of MODPARAMS.DAT will maintain consistent VMS performance as system memory and peripheral devices are modified.

## Introduction

AUTOGEN is a VAX/VMS utility which attempts to maintain a well tuned VMS system. The default criteria for determining necessary parameters is the current hardware inventory (i.e. physical memory, terminal ports, disks/tapes and their servers, and communication devices).

When actual system behavior differs from the DEC expected "norm", the system manager can "tune" the system parameters to improve system performance. DEC recommends and encourages that this tuning be done with the AUTOGEN utility.

This paper will briefly review the functions and operations of AUTOGEN, discuss why and when to use AUTOGEN, and provide specific recommendations on how to effectively use the AUTOGEN utility. This paper will NOT discuss system tuning or the system parameters.

## What is AUTOGEN

AUTOGEN is a DIGITAL (VMS) supplied command procedure found in SYS$UPDATE. It is first used during the initial VMS installation, and during major VMS updates, to establish default system (SYSGEN) parameters. AUTOGEN will also set the size of the primary pagefile, swapfile, and dump file. AUTOGEN also provides a mechanism for the system manager to affect the resultant calculations of SYSGEN parameters, via SYS$SYSTEM:MODPARAMS.DAT.

### AUTOGEN functions

AUTOGEN performs several functions, depending upon how it is invoked. These functions include:

1. Evaluates the current hardware configuration. This includes the amount of physical memory, number of terminal ports, disk drives and controllers, bus devices (i.e. CI, NI, and UDA), and communication devices (e.g. Ethernet, DMR, DUP, etc.). This information is included in determining minimum and default parameter values.

2. Evaluates current SYSGEN parameters. The values of a select set of SYSGEN parameters are read and saved, forming a "baseline". This baseline is used in subsequent calls to AUTOGEN, providing minimum or default values.

3. Calculates new values for significant SYSGEN parameters.

   The data collected from the hardware configuration, previous SYSGEN parameter values, and input from the system manager (via the SYS$SYSTEM file MODPARAMS.DAT) is used to compute new values for significant SYSGEN parameters (see Table 1 for the list of these significant parameters).

4. Calculates new sizes for the system PAGE, SWAP & DUMP files. A value for the size of the Page and Swap files are calculated based on:

   - the amount of physical memory
   - the size of the system disk, and
   - the values of MAXPROCESSCNT & VIRTUALPAGECNT

   The size of the DUMP file is based on the amount of physical memory.

5. Resets system parameters and files sizes.

   The computed values described in the previous functions are saved in a file, which you are able to preview. This file is used as the input to SYSGEN to make the actual changes, which will be effective at the next system reboot.

6. Optionally shutdown and reboot the system.

## How does AUTOGEN work

To ensure that you have the required privileges, AUTOGEN should be invoked ONLY from the System Manager's account. The format is:

```
@SYS$UPDATE:AUTOGEN -
  [start-phase] [end-phase] [exec-type]
```

You can enter up to 3 parameters to designate the AUTOGEN function you desire. All parameters are optional; however, any missing or leading parameters must be replaced with null arguments, according to DCL parameter passing standards. The end-phase defaults to the same value as the start-phase; GENPARAMS is the default start-phase.

As of VMS 4.4, there are seven phases of AUTOGEN. These phases have a definite sequence; the start-phase MUST either precede or be equal to the end-phase according to the sequence in Table 2.

### AUTOGEN data files

The phases of AUTOGEN pass information among each other through data files. Each phase uses a set of files for input, and a set of files for output. Table 4 associates the various phases with their corresponding input/output file sets. Of the list of files shown, ONLY MODPARAMS.DAT should be modified, all the remaining are considered read-only to the System Manager, and their modification could interfere with AUTOGEN operations.

The OLDSITE*.DAT files contain the "baseline" values for select parameters. These baseline values define the default and/or minimum value for that parameter.

PARAMS.DAT is simply the concatenation of the physical configuration, OLDSITE*.DAT, and MODPARAMS.DAT.

SETPARAMS.DAT is a SYSGEN input file, containing a collection of commands setting the calculated parameter values.

## When to use AUTOGEN

There basically two times to run AUTOGEN.

- Whenever ANY hardware modification is made (i.e. addition or deletion of memory, controllers, or devices).
- Whenever you need to modify certain system parameters.

## Why use AUTOGEN

The VMS system parameters effecting system performance have very complex inter-relationships among themselves AND system hardware configuration. Parameters "out-of-balance" with each other, and with system configuration, can and will have an adverse impact on performance that can be very difficult to isolate or identify.

The most important benefit of AUTOGEN is that it understands this complexity and is therefore able to provide a consistent set of parameters. This consistency is maintained as you adjust the system parameters during tuning activities and/or software installations AND when you alter your hardware configuration. Table 5 demonstrates some of the inter-dependencies of the major system parameters.

Another benefit is that any/all changes you make to your system are documented in MODPARAMS.DAT. This documentation is very important when system management responsibility is transferred to another person, when sufficient time elapses that you've forgotten what/why you've made some alteration, and when you remove an optional software product (i.e. GBLPAGES)

### Relationships among major SYSGEN parameters

Table 5 defines, in a very general manner, some of the inter-relationships among some of the major system parameters. The format of the table is:

```
PARAM = f (param_1,param_2,...param_n)
```

which indicates that PARAM is a function of param_1 thru param_n.

old_xxx refers to parameter xxx from one of the OLDSITE*.DAT files.

add_xxx refers to an increase in parameter xxx as specified in MODPARAMS.DAT.

## How to use AUTOGEN effectively

Now to the meat of this discussion. How does one effectively use AUTOGEN?

- The MOST important rule is: All input to SYSGEN is made through the file MODPARAMS.DAT with your favorite editor. The format of MODPARAMS.DAT will be discussed a little later. SYSGEN should not be run to modify a system parameter, without a corresponding change to MODPARAMS.DAT. This will provide the necessary documentation of system parameters.
- Invoke AUTOGEN with the DCL command

```
@SYS$UPDATE:AUTOGEN GETDATA GENPARAMS
```

This command will cause AUTOGEN to read the OLDSITE*.DAT files, your MODPARAMS.DAT, and gather the system hardware configuration data, placing all this data in PARAMS.DAT (GETDATA). It will then use this data to calculate new system parameters values (GENPARAMS).

| | | |
|---|---|---|
| ACP_DIRCACHE | ACP_HDRCACHE | ACP_MAPCACHE |
| ACP_MULTIPLE | ACP_QUOCACHE | ACP_SWAPFLAGS |
| ACP_SYSACC | BALSETCNT | BORROWLIM |
| FREEGOAL | FREELIM | GBLPAGES |
| GBLPAGFIL | GBLSECTIONS | GROWLIM |
| IRPCOUNT | IRPCOUNTV | LOCKIDTBL |
| LOCKIDTBL_MAX | LRPCOUNT | LRPCOUNTV |
| MAXPROCESSCNT | MPW_HILIMIT | MPW_LOLIMIT |
| MPW_WAITLIMIT | NPAGEDYN | NPAGEVIR |
| PAGEDYN | PFCDEFAULT | PFRATL |
| RESHASHTBL | SCSCONNCNT | SPTREQ |
| SRPCOUNT | SRPCOUNTV | SYSMWCNT |
| VAXCLUSTER | VIRTUALPAGECNT | WSMAX |
| WS_OPA0 | | |

**Table 1 – System Parameters Affected By Autogen**

| PHASE | FUNCTION |
|---|---|
| SAVPARAMS | Save significant "old" parameters for propagation and update (baseline). |
| GETDATA | Collects all data, including configuration, old data & site-specific items. |
| GENPARAMS | Computes new system parameters. |
| TESTFILES | Displays computed page, swap & dump files. cannot be specified as a start phase. |
| GENFILES | Creates new page, swap & dump files, if required. cannot be specified as a start phase. |
| SETPARAMS | Runs SYSGEN to set new parameters. |
| SHUTDOWN | Shuts down the system. |
| REBOOT | Automatically reboots the system. |

**Table 2 – AUTOGEN Phase Parameters**

| TYPE | Meaning |
| --- | --- |
| INITIAL | AUTOGEN is being run as part of an initial install. SAVPARAMS is never executed. |
| V4UPGRADE | AUTOGEN is being run as part of an upgrade from a Version 4.n system, or that interactive tuning is being performed. This is the DEFAULT execution type. |
| V3UPGRADE | AUTOGEN is being run as part of an upgrade from Version 3.n to Version 4.n |

**Figure 3:** Table 3 – Execution Types

| Phase | Input Files | Output Files |
| --- | --- | --- |
| SAVPARAMS | None | OLDSITE1.DAT<br>OLDSITE2.DAT<br>OLDSITE3.DAT<br>OLDSITE4.DAT |
| GETDATA | OLDSITE1.DAT<br>OLDSITE2.DAT<br>OLDSITE3.DAT<br>OLDSITE4.DAT<br>MODPARAMS.DAT | PARAMS.DAT |
| GENPARAMS | PARAMS.DAT | SETPARAMS.DAT<br>VMSIMAGES.DAT |
| TESTFILES | PARAMS.DAT | SYS$OUTPUT |
| GENFILES | PARAMS.DAT | PAGEFILE.SYS<br>SWAPFILE.SYS<br>SYSDUMP.DMP |
| SETPARAMS | SETPARAMS.DAT | VAXVMSSYS.PAR<br>AUTOGEN.PAR |
| SHUTDOWN | None | None |
| REBOOT | None | None |

**Figure 4:** Table 4 – AUTOGEN Data Files

```
------------------------------------------------------------
GBLPAGFIL   = f (old__gblpagfil,add__gblpagfil)

GBLPAGES    = f (old__gblpages,add__gblpages,gblpagfil,
                 VMS__installed__images (from VMSIMAGES.DAT))

GBLSECTIONS = f (old__gblsections,add__gblsections,
                 VMS__installed__images (from VMSIMAGES.DAT))

SRPCOUNT    = f (maxprocesscnt,number__of__devices,add__srpcount)

IRPCOUNT    = f (maxprocesscnt,num__mscp__servers,add__irpcount)

LRPCOUNT    = f (number__comm__devices,num__mscp__servers,num__ci,
                 memory__size,add__lrpcount)

NPAGEDYN    = f (device__drivers__loaded,num__devices,maxprocesscnt,
                 memory__size,add__npagedyn)

PAGEDYN     = f (npagedyn,balsetcnt,acp__parameters,add__pagedyn)

SYSMWCNT    = f (gblpages,sptreq,pagedyn,memory__size,add__sysmwcnt)

LOCKIDTBL   = f (maxprocesscnt,acp__parameters,cluster,add__lockidtbl)

RESHASHTBL  = f (lockidtbl, add__reshashtbl)
------------------------------------------------------------
```

**Figure 5:** Table 5 - Relationships Among Major SYSGEN Parameters

| | | |
|---|---|---|
| ACP_DINDXCACHE | ACP_DIRCACHE | ACP_HDRCACHE |
| ACP_MAPCACHE | ACP_MULTIPLE | ACP_QUOCACHE |
| ACP_SWAPFLAGS | ACP_SYSACC | BALSETCNT |
| BORROWLIM | CTLPAGES | FREEGOAL |
| FREELIM | GBLPAGES | GBLPAGFIL |
| GBLSECTIONS | GROWLIM | IRPCOUNT |
| IRPCOUNTV | IRPSIZE | KFILSTCNT |
| LOCKIDTBL | LOCKIDTBL_MAX | LRPCOUNT |
| LRPCOUNTV | LRPMIN | LRPSIZE |
| MAXBUFF | MAXPROCESSCNT | MPW_HILIMIT |
| MPW_LOLIMIT | MPW_WAITLIMIT | NPAGEDYN |
| NPAGEVIR | PAGEDYN | PFCDEFAULT |
| PFRATL | PFRATL | PHYSICALPAGES |
| PIXSCAN | RESHASHTBL | SCSCONNCNT |
| SPTREQ | SRPCOUNT | SRPCOUNTV |
| SRPSIZE | SYSMWCNT | VAXCLUSTER |
| VIRTUALPAGECNT | WSMAX | WS_OPA0 |

**Figure 6:** Table 6 - Parameters Adjustable with ADD_prefix

- The VMS System Manager's Reference Manual suggests that at this point you should read PARAMS.DAT. I find the information in this file generally useless. It simply gives back to me what I already know! I'm usually most interested in what NEW values have been calculated for certain parameters. This information can be found in SETPARAMS.DAT. This file should be examined.

  **Never edit SETPARAMS.DAT.** If you examine it with an editor, use the /READONLY switch with the editor.

  If you don't like some of the computed values, based on your experience, edit MODPARAMS.DAT again to make the appropriate adjustment.

- If you ARE satisfied with AUTOGEN's calculated values, then you have to cause these new values to take effect. This is done with the DCL command

```
@SYS$UPDATE:AUTOGEN SETPARAMS [reboot|shutdown]
```

  The single parameter of SETPARAMS will cause AUTOGEN to invoke SYSGEN, setting the parameters as defined in SETPARAMS.DAT. If only the single parameter is specified, the new parameter values will be loaded into the system, ready to take effect at the next system reboot.

  If you want/need the new parameter values to take place immediately, add the REBOOT phase, which will cause a system shutdown and an immediate reboot. The SHUTDOWN phase will simply cause AUTOGEN to shutdown your system, and wait for you to manually reboot the system.

  When AUTOGEN initiates a shutdown, it uses the system shutdown procedure in SYS$SYSTEM, and specifies a time of 0 minutes before shutdown (i.e. NO warning to your users). You can specify the shutdown time for AUTOGEN to use by assigning the logical name AGEN$SHUTDOWN_TIME to your desired time delay for shutdown or reboot.

SYS$SYSTEM:MODPARAMS.DAT is the system manager's input to AUTOGEN. VMS V4 introduced a new syntax to MODPARAMS.DAT, the ADD_prefix, to increment the values of certain SYSGEN parameters (see Table 6) during the GENPARAMS phase. The ADD_parameter value can be negative to decrement AUTOGEN's calculated value. This new syntax makes the VMS 4 AUTOGEN a useful tool for system tuning.

You, as the system manager, should develop a format for MODPARAMS.DAT and stick to it! The format should document "WHO made WHAT changes WHEN and WHY!". Some specific recommendations are:

- Never hard-code MAJOR SYSGEN parameters (use the ADD_xxx syntax).
- The ADD_xxx is valid ONLY for affected parameters (Table 6). Note that rumor has it that VMS 4.5 will

allow the ADD_ prefix for ALL numeric sysgen parameters. An ADD_xxx entry can appear ONLY once for a given parameter. The "last" entry will prevail if multiple entries occur.

- Define a value for SWAPFILE and PAGEFILE.
  - A value of 0 tells AUTOGEN to leave the current files alone.
  - An explicit value for each, equal to the desired size.
- You can do simple symbolic (DCL, integer) arithmetic. However, any parameter used in the right-hand side of the equation MUST appear earlier in the file.

Appendix A is an example of a bad MODPARAMS.DAT, containing NO documentation; you do not know when or, more importantly, why these parameters are set that way. Also major SYSGEN parameters are hard-coded, preventing AUTOGEN from adjusting them according to their inter-dependencies and/or hardware configuration changes. This was about all that could be done before VMS V4, and is part of AUTOGEN's low regard among many users.

Appendix B contains a good MODPARAMS.DAT, incorporating some documentation, You get an idea of why some of the parameters defined, and in some cases who changed them (via initials) and when. The major SYSGEN parameters are modified with the ADD_xxx syntax. This allows AUTOGEN to include site-specific requirements in its calculations.

The EXCELLENT example (Appendix C) demonstrates improved documentation. The step-wise refinement is demonstrated (either as a correlated sum of numbers, see ADD_GBLPAGES, or each contribution is itemized, see ADD_GBLSECTIONS and ADD_IRPCOUNT). The PAGE and SWAP files are explicitly set, and BALSETCNT is a function of MAXPROCESSCNT.

## Migrating to an "EXCELLENT" MODPARAMS.DAT

If your system's MODPARAMS.DAT is anything like mine was before I started, it looks like the BAD example in appendix A. If you would like to upgrade your "BAD" MODPARAMS.DAT to the "EXCELLENT" category, follow the these simple steps:

- This procedure assumes that you are basically happy with the current performance of your system, and you ONLY wish to alter the format of MODPARAMS.DAT.
- Obtain a listing of your current SYSGEN parameters. This can be done by entering the SYSGEN utility and executing the following commands:

```
SYSGEN> SET/OUTPUT=<some-file>
SYSGEN> USE CURRENT
SYSGEN> SHOW/ALL
SYSGEN> EXIT
```

- Rename your current MODPARAMS.DAT (to something like MODPARAMS.OLD). This is important! You need to disable reading your "defective" MODPARAMS.DAT.
- Run AUTOGEN using:

```
@SYS$UPDATE:AUTOGEN GETDATA GENPARAMS INITIAL
```

- Compare the calculated/initial SYSGEN parameters with your current values recorded in step 1.
- If there are any differences, which there most likely will be, create a new MODPARAMS.DAT to compensate for these differences. Here's where you migrate to the "excellent" format. Go back to the third step, to make new calculations.

## Conclusion

The VAX/VMS AUTOGEN Utility can be a very powerful tool to maintain a well tuned computer system. The effectiveness of this tool, as with all tools, is how well it is used. If you use AUTOGEN in the manner just described, and with the recommended frequency, you too should be able to run a well tuned VAX system.

## References

[1] "VAX/VMS System Manager's Reference Manual," V4.4, Chapter 11, April 1986, Order Number AA-Y507B-TE, Digital Equipment Corporation

[2] VAX/VMS V4.4 Micro-fiche, Sheet 285, Page J14, Digital Equipment Corporation

## APPENDIX A
## A "BAD" MODPARAMS.DAT

```
VAXCLUSTER=2
QUORUM=2
DISK__QUORUM="_$1_$DUA8"
SCSSYSTEMID = 8
SCSNODE= "MYNODE"
VIRTUALPAGECNT=40000
MAXPROCESSCNT=60
BALSETCNT=50
GBLSECTIONS=345
GBLPAGES=4570
LOCKIDTBL=3500
SRPCOUNT=4106
NPAGEDYN=400000
IRPCOUNT=546
SYSMWCNT=670
```

APPENDIX B
A "GOOD" MODPARAMS.DAT

```
!
!   NODE "customizations"
!       CLUSTER STUFF
!
VAXCLUSTER=2                        ! Always join cluster
QUORUM=2                            !   with NODEA.
DISK__QUORUM="_$1_$DUA8"            ! use NODEA's system disk as a quorum disk
SCSSYSTEMID = 8
SCSNODE = "MYNODE"
VIRTUALPAGECNT=40000                ! Added to support crash dump analysis.
MAXPROCESSCNT=60
BALSETCNT=50
SWAPFILE = 0                        ! Leave Swap _& Page files alone
PAGEFILE = 0
ADD__GBLSECTIONS=45                 ! Added for RDB,CMS,FORTRAN,SD
ADD__GBLPAGES=2570                  ! Added for RDB,CMS,FORTRAN,SD
ADD__LOCKIDTBL=1500                 ! Added for RDB
ADD__SRPCOUNT=1206                  ! Added for RDB
ADD__NPAGEDYN=40000                 ! Adjust for observed usage + PRODUCTA (dlc)
ADD__IRPCOUNT=556                   ! TNG's tuning suggestion (RAP).
ADD__SYSMWCNT=70                    ! Added to reduce system page faults 1/7/86
```

APPENDIX C
AN "EXCELLENT" MODPARAMS.DAT

```
!
!   NODE "customizations"
!       CLUSTER STUFF
!
VAXCLUSTER=2                    ! Always join cluster
QUORUM=2                        !   with NODEA.
DISK__QUORUM="_$1_$DUA8"        ! use NODEA's system disk as a quorum disk
SCSSYSTEMID = 8
SCSNODE = "MYNODE"
!
VIRTUALPAGECNT=40000            ! Added to support crash dump analysis.
MAXPROCESSCNT=60
BALSETCNT=MAXPROCESSCNT-5
!
ADD__GBLSECTIONS=18             ! Added for RDB(10),CMS(5),FORTRAN(2),SD(1)
ADD__GBLPAGES=854+123+341+45    ! Added for RDB,CMS,FORTRAN,SD
ADD__LOCKIDTBL=1500             ! Added for RDB
ADD__SRPCOUNT=1000+206          ! Added for RDB,observed requirements
ADD__NPAGEDYN=35000+4000        ! Adjust PRODUCTA (dlc)
!                               !  & obs'd reqmt, up from 3000 JRM  05/20/86
ADD__IRPCOUNT=500+56            ! TNG's tuning suggestion (RAP).
!                               ! (+43) Observed requirement. JRM  05/15/86
!                               ! (+13) Observed requirement. JRM  05/19/86
!
ADD__SYSMWCNT=50+(BALSETCNT/5)  ! Demonstrate computational capability
SWAPFILE=30000                  ! Leave Page and Swapfiles
PAGEFILE=40000                  !   alone!
```

# Heterogeneous VAXClusters

**Frank J. Nagy**
Fermi National Accelerator Laboratory[1]
Batavia, IL, 60510

### Abstract

The vast majority of VAXClusters are homogeneous, so just what is a heterogeneous VAXCluster? The definition and uses of a heterogeneous cluster and the concept of an unbalanced cluster are discussed. The development of an unbalanced, heterogeneous VAXCluster in the Fermilab Accelerator Control System is described.

The introduction by Digital of the VAXCluster has had an enormous impact on VAX users and managers. Several thousand clusters have been installed and are in operation today. More clusters are being installed daily and existing clusters are growing larger and more complex. Existing VAXClusters are beginning to rival mainframe computers in aggregate compute power and disk resources. Clusters are also beginning to mutate into different forms to fill specialized niches. This paper defines some of these mutations and their uses.

### Definitions

What constitutes a VAXCluster? The minimal cluster is two or more VAX systems interconnected by a high-speed communications channel (currently the CI bus). A cluster is usually typified as having a single, shared file system. This file system can be spread across disks connected to the cluster directly via Hierarchical Storage Controllers (HSC) or via local disks connected to VAX processors and made available to the cluster with the MSCP Server. The vast majority of clusters in use today are homogeneous clusters. A homogeneous cluster can be characterized as having a single system image for its users by virtue of there being a single User Authorization File (UAF) for all the nodes of the cluster. On the other hand, a homogeneous cluster may have different VAX models as nodes or may have multiple system disks (usually for performance reasons) and still be considered a homogeneous cluster if it has a single UAF.

A VAXCluster might be termed a semi-homogeneous cluster when it contains unique hardware or software nodes but still supports a single UAF. By having a single UAF, any user can use the cluster from any node of the cluster and still access the cluster-wide resources such as the file system and print and batch queues. In the semi-homogeneous cluster, the users sometimes need to be aware that certain resources are available only from specific nodes of the cluster and not from other nodes. This most commonly manifests itself in nodes with non-shareable hardware resources such as tape drives, non-served disk drives or high performance graphics or CAD stations connected to a single cluster node. Note that printers are often connected to the cluster via a single node but are considered as cluster-wide resources since the queues which feed the printers are available to all nodes of the cluster.

Going from a single UAF to multiple UAF files, with different UAF files used by the various nodes in the cluster, results in a heterogeneous VAXCluster. Such a system will almost surely have unique nodes with specialized hardware or software. In a sense, a heterogeneous cluster is very much like independent systems connected in a network. However, the heterogeneous cluster does provide some advantages over networked systems:

- The heterogeneous cluster provides a common file system which can be simulated to some degree with networked systems by use of proxy logins and logical names but does not provide the easy and rapid file access available on the cluster. A heterogeneous cluster common file system is most easily realized and managed if the separate UAFs of the cluster constitute a master UAF with subset UAFs for particular nodes.
- The heterogeneous cluster permits the sharing of resources that networked systems do not. Such resources include:
  - Disk and tape drives
  - Printers and print queues
  - CPU time (batch jobs)

The heterogeneous cluster can provide a significant cost savings using HSCs to support either a common

[1]Fermilab is operated under contract to the US Department of Energy.

file system or a segregated file system but on cluster-wide disk and tape hardware.

- Experience seems to indicate that a heterogeneous cluster is somewhat easier to manage than independent networked systems. This may primarily be an attribute of a common file system.

### A Site History

The Fermilab Accelerator Division VAXCluster began as two independent VAX-11/780s using RM03 and RM80 Massbus disk drives. At about the time a major disk storage capacity upgrade was required, Digital announced the VAXCluster program with the CI and HSC hardware. A decision was made to expand the system in just that direction. Accordingly, a complete CI system with dual HSC50s and several RA81 and RA60 disk drives was installed and brought into operation under VMS V3.5. By the time VMS V3.7 was installed, a third VAX-11/785 was added (the other two VAXes were now also 785s) along with additional RA81 disk drives. While running under VMS V3.7, the disk farm was partitioned into multiple, independent file systems. The DCL command

```
SET DEVICE /NOAVAILABLE
```

was used to prevent users on one system from mounting another system's disk (of course a disk was only mounted on a single system at any time).

Two of the three VAXes in this system are used for software development for the ACNET accelerator control system and for accelerator physics calculations used in the design and understanding of accelerators. The third VAX was an active part of the accelerator control system; the only interactive use of this system, other than by system managers, was by users working on the control system database via limited, captive accounts. One of the two development VAXes constituted a hot backup for the operational system. When VMS V4.0 became available, the plans were made for forming the two development systems into a single homogeneous cluster and running the operational VAX as an independent system much as was then being done under VMS V3.7. This configuration was chosen to integrate the two development systems into a single cluster and yet have minimal impact on the operational VAX while providing a test bed to gain experience in operating a cluster and evaluate the effects of integrating the operational VAX into the cluster.

The systems were cutover to VMS V4.2 in just this mode of a two-node homogeneous cluster and an independent VAX system sharing the CI and disk farm but with segregated file systems. The techniques used under VMS V3.7 to prevent direct access to the disks of the other systems were carried over and used successfully under VMS V4.x. However, a more flexible scheme was needed to manage access to the shared peripherals, a "spare" RA60 disk drive and the TA78 tape drives. A program called ALLOCWATCH[2] was written to use DECnet to make the allocation state of these shared peripherals carry across the boundary between the independent VAX and the cluster sharing the CI system. This system functioned satisfactorily until needs forced the planning for a second operational VAX node.

Two possibilities existed for integrating the fourth VAX. This could be another independent VAX system like the existing operational VAX with its own system disk and file system separate from the existing operational VAX and development VAXCluster file systems. Alternatively, the existing systems could be merged into a single cluster in which case the fourth VAX would just be an additional node for this cluster. By the time this decision had to be made, just after VMS V4.4 was installed, sufficient experience with the operation of the cluster had been accumulated to sway the decision toward the four-node heterogeneous cluster configuration.

Thus, in the Spring of 1986, the existing homogeneous cluster was migrated into a three-node heterogeneouscluster. The original cluster used a common system disk based on an RA81 which contained the additional layered products (compilers, libraries, etc.) and third party software (TeX, laser printer fonts, etc.) used by the software development users. The original system disk of the independent operational VAX on an RA60 was converted into a second common system disk to be used for the active control system nodes of the cluster. Each system disk had its own UAF; the accounts in the UAFs had already been scanned to insure that those common to both UAFs were intentional and had identical User Identification Codes (UICs). The file systems were "merged" by mounting all the volumes cluster-wide. File usage was still segregated in the sense that the operational control system files were on RA60 disks and the development files on RA81 disks.

Experience has demonstrated that managing the resulting heterogeneous cluster is somewhat easier than managing independent systems. DECnet is still used frequently for interactive access via SET HOST, but most DECnet file transfers have been replaced by direct access to the disks. Plans to merge the separate queue managers (one for the operational system and one for the development systems) into a single cluster-wide system of queues have not yet been implemented. The plans had called for the batch queues on the operational systems to not be fed by the cluster-wide generic batch queues but to remain separate due to the separate UAF. The primary result of merging the queue management will be to permit the operational systems to have direct access to the printers connected to the development nodes and to be able to queue print jobs even when the development nodes are down.

[2]ALLOCWATCH appears on the Fall 1986 VAX SIG tape in the [.FERMILAB] directory tree.

## Unbalanced VAXClusters

The effort to merge the Fermilab Accelerator Division VAXes into a single cluster was dominated by the problem of maintaining an operating cluster with only the single operational VAX node. The accelerator control system is active around the clock; it can tolerate occasional outages of a few minutes to a few tens of minutes but longer periods cause severe problems. Thus the cluster had to be configured in such a manner that the operational node could remain alive while either or both of the development nodes were down. This requirement was solved by creating an "unbalanced" VAXCluster in which the distribution of votes is not uniform across the nodes of the cluster.

A VAXCluster maintains system integrity with the concepts of votes and quorum. Each active member of the cluster contributes one or more votes and a target value for the quorum (the SYSGEN parameters VOTES and QUORUM). As long as the sum of all the votes in the cluster exceeds the current quorum value, then the nodes of the cluster will operate normally. When a cluster state transition results in the sum of the votes dropping below the current quorum value, then all the nodes will hang awaiting a new cluster state transition. This scheme is used to prevent the cluster from partitioning into two which could then proceed to modify the file system without the necessary synchronization required to maintain the file system integrity. The new quorum value ($Q_{new}$) of a state transition is calculated as the maximum of current quorum value ($Q_{current}$), the quorum values contributed by the individual nodes ($q_1, \ldots, q_n$) and the value $\frac{V+2}{2}$ where $V$ is just the sum of the votes from the individual nodes:

$$V = \sum_{i=1}^{n} v_i$$

$$Q_{new} = \max(Q_{current}, q_1, \ldots, q_n, V)$$

where $v_i$ is the value of the SYSGEN parameter VOTES and $q_i$ is the value of the QUORUM parameter for the $i$th system.

For the Accelerator VAXCluster, the goal was to keep node OPER (central server for the control system) operating when either or both of the development nodes (DEVL and ADCALC) were down. Table 1 shows the values of the VOTES and QUORUM SYSGEN parameters used by the Accelerator cluster nodes to achieve stable operation of an unbalanced cluster. Table 2 shows how these SYSGEN parameters interact to insure stable operation of the unbalanced cluster without allowing partitioning. As seen in Table 2, $V$ is always greater than or equal to $Q$ as long as OPER is a member of the cluster. Without OPER, the cluster will either not form or the DEVL and ADCALC nodes will hang (if OPER has dropped out of an operating cluster). Similarly, if all three nodes are in operation as cluster members, the loss of either or both of the DEVL and ADCALC nodes will not cause $V$ to drop below $Q$ so that OPER can always operate as a single-node VAXCluster. In fact, a bit of study of Table 2 results in the general configuration rules for a $N$-way cluster in which all nodes are "equal" but for a single special node (call it node $A$):

| Node | $v_i$ | $q_i$ |
|---|---|---|
| OPER | 4 | 3 |
| DEVL | 1 | 3 |
| ADCALC | 1 | 3 |

**Table 1:** Parameters for 3 nodes

| *Nodes...* | $v_i$ | V | (V+2)/2 | Q |
|---|---|---|---|---|
| OPER alone | 4 | 4 | 3 | 3 |
| OPER + DEVL | 4<br>1 | 5 | 3 | 3 |
| DEVL + ADCALC | 1<br>1 | 2 | 2 | 3 > V |
| OPER + DEVL + ADCALC | 4<br>1<br>1 | 6 | 4 | 4 |

**Table 2:** A 3-node unbalanced cluster

- The $N-1$ *unimportant* nodes have VOTES set to 1.
- The QUORUM parameter of all the nodes is set to $N$.
- The VOTES parameter of the *important* node $A$ is set by
$$v_A = 2N - 2$$

which, for three nodes, results in Table 1.

Tables 1 and 2 illustrate the case of a three node unbalanced VAXCluster in which a single node is all-important. Tables 3 and 4 illustrate a four node unbalanced cluster in which two nodes (OPER and CDBS) form the *important* core of the cluster. These tables come from the planning for the near-term upgrade of Accelerator VAXCluster to go from a single operational control system node (OPER) to two such nodes (OPER and CDBS) while still retaining the two development nodes DEVL and ADCALC. In this four-way cluster configuration, nodes OPER and CDBS are the *important* nodes which must always be operating. As Table 4 shows, the

| Node | $v_i$ | $q_i$ |
|---|---|---|
| OPER | 3 | 5 |
| CDBS | 3 | 5 |
| DEVL | 1 | 5 |
| ADCALC | 1 | 5 |

**Table 3:** Parameters for 4 nodes

| *Nodes...* | $v_i$ | V | (V+2)/2 | Q |
|---|---|---|---|---|
| OPER alone | 3 | 3 | 2 | 5 > V |
| OPER +<br>DEVL | 3<br>1 | 4 | 3 | 5 > V |
| OPER +<br>CDBS | 3<br>3 | 6 | 4 | 5 |
| OPER +<br>CDBS +<br>DEVL | 3<br>3<br>1 | 7 | 4 | 5 |
| OPER +<br>CDBS +<br>DEVL +<br>ADCALC | 4<br>3<br>1<br>1 | 8 | 5 | 5 |

**Table 4:** A 4-node unbalanced cluster

cluster will not operate unless both are up and running. Either of the nodes, OPER or CDBS, alone or in concert with DEVL and ADCALC cannot form a cluster. However, both OPER and CDBS together form a cluster without needing to have either DEVL or ADCALC online.

## Summary

The effort to derive a general set of configuration rules for an unbalanced VAXCluster has not been undertaken since the approach illustrated in Tables 1 and 2 for a three-node cluster and then again in Tables 3 and 4 for a four-node cluster can be used to solve other specific cases. This scheme has worked well for configuring the existing Fermilab Accelerator VAXCluster and for planning for its near-term expansion as an unbalanced cluster serving the dual missions of operating the Fermilab accelerator complex and hosting software development for the control system and accelerator design calculations.

## References


Digital Equipment Corporation. *Guide to VAXClusters*, Version 4.0, September 1984. Order number AA-Y513A-TE.

Digital Equipment Corporation. *VAX/VMS System Generation Utility Reference Manual*, Version 4.0, September 1984. Order number AA-Z433A-TE.

# Primarily Ultrix and a Little VMS on MicroVaxes

**Wendy Rannenberg**
Sanders Associates
Nashua, NH 03061-2034

## Abstract

What began as an effort to evaluate Ultrix on a MicroVax II quickly turned into a review of Sanders use of VMS based MicroVaxes as well. The intent of this paper is to describe Sanders computing environment and how an Ultrix based MicroVax fits in as well as to report the results of benchmarking efforts.

Sanders, being an engineering firm, requires a large amount of flexibility in its computing environment. Performance of systems, both throughput and plain old cpu horse power, is critical to many programs. Benchmark results for an Ultrix 1.1 based MicroVax and other Unix systems are presented in an effort to address this area. Although many of the benchmarking programs used for the evaluation are widely distributed throughout the Unix user community many signal processing applications, developed at Sanders, were also used.

Included in the evaluation is a brief review of the Ultrix documentation and installation procedures.

## Computing Environment

The intent of this section is to provide readers with an overview of the computing environment into which Ultrix based MicroVaxes would be placed.

### Physical Description

Sanders computing facilities have undergone major changes over the past three years. Originally a few PDP11's running RSX and Version 7 Unix facilities now include a VMS/DECnet based ***engineering network*** of at least one hundred systems complemented by a TCP/IP network of forty Unix based systems and Lisp machines as well as an IBM/SNA network. These networks, providing support for software development, program management, office automation, CAD and internal research programs, are interconnected by gateway systems as shown in Figure 1.

Access to external networks, uucp and usenet, are provided through a single Ultrix system so utilization can be monitored. Long range plans for these networks include adhering to ISO standards while providing for a more tightly integrated computing environment.

MicroVax computers account for most of the cpu's available to engineers. They are primarily used for CAD applications but several host small project development activities. Project based MicroVaxes tend to be VMS based systems stripped of utilities, such as mail and MMS, and tuned for optimal compiler performance. Tools have been eliminated as a result of limited disk space on most systems.

### Software Development

As previously mentioned, Sanders computing facilities support a broad range of engineering applications. Software development environments and activities fall into three broad categories:

- VMS based development - Work in this environment includes development of high performance systems using Fortran such as real time training systems, database design, and graphics system design.
- VAXELN based development - Real time signal acquisition systems with development done in both Pascal and C.
- Unix based development - Air traffic control systems, research and development, signal processing applications, and prototyping work in f77, C, Pascal , and Modula 2.

Typical program development needs are no different from those found in other organizations. For example there is a need for production quality development tools (e.g. compilers, editors, debuggers, etc.), high performance networking, high performance graphics, and prototyping capabilities. The latter is being driven by DoD requirements to provide prototype systems prior to system development contract award.

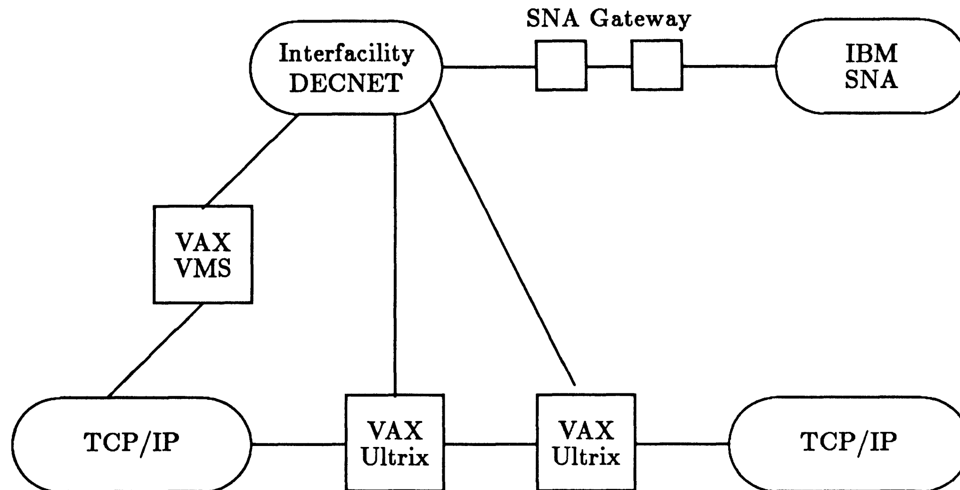



**Figure 1:** Sanders Internal Computer Network Structure

### Evaluation Activities

Technical evaluation of the MicroVax II/Ultrix combination focused on use as a one to three user system rather than one that would be expected to support a large complement of engineers. Originally the system was to be evaluated for potential use by at least eight software developers. The selected range was based on experiences with VMS and VAXELN software development on MicroVaxes although it is possible to support more users depending on the application. Of most interest was the potential use of the system for signal processing and other cpu intense applications such as AI development and text formatting. Since so many benchmark results have been published on MicroVaxes the processor speed was not critical to the evaluation and price performance issues were considered beyond the scope.

Microvax/Ultrix benchmarks were completed in January of 1986 while others described herein were completed prior to that date. Since that time new releases of software and hardware have been announced by all vendors mentioned in this report.

### Evaluation System Configuration

The MicroVax II used for this evaluation was loaned to Sanders by DEC for six weeks. The system was configured as shown below and a hardwired uucp connection was made to a 4.2bsd Vax 11/750 for file transfer purposes.

- 3 MegaBytes of physical memory
- 1 DHV-11 eight line serial multiplexor
- Floating point hardware (standard)
- Q22 bus
- 2 RD52 (71 MegaByte) disk drives
- 1 TK50 (95 MegaByte) cartridge tape drive
- Ultrix 32m V1.1 operating system

### Benchmark Results

The following systems were used during this evaluation to provide for comparative analysis. The latter two are Masscomp systems, one of which was on loan to Sanders.

The reader is cautioned in making conclusions based on the following results. Many factors influence the results, particularly hardware and software configurations of the systems. Vax 11/780 and MC 5600 results were obtained from systems in multiuser mode. It is always best to run benchmarks in single user mode so that the exact state of the system is known at any given point in time and to have more accurate timing information.

The following sections provide analysis of both synthetic and application specific benchmarks. Unless otherwise noted reported times are given in seconds.

#### *Whetstone Benchmark Results*

The whetstone benchmark, used extensively throughout Sanders as well as industry, provides a measure of floating point arithmetic performance of a cpu. Developed in 1970[CURN70] as a "compromise between requirements for simplicity and complexity", this benchmark is possibly becoming obsolete as compiler technology advances

| Processor | Floating Point | Memory | Disks | O.S. |
|---|---|---|---|---|
| MicroVax II | Yes | 3Mb | 2 RD52 | Ultrix 32m V1.1 |
| Vax 11/750 | No | 3Mb | 3 RM05 | 4.2bsd, System V, Ultrix 32 V1.2 |
| Vax 11/780 | Yes | 8Mb | 1 RP07/2 RM05 | Ultrix 32 V1.1 |
| MC 500 (68010) | No | 2Mb | 1 Fujitzu(80Mb) | RTU 2.2A |
| MC 5600 (68020) | Yes | 8Mb | 1 Eagle(380Mb) | RTU 3.0A |
| MicroVax II | Yes | 3Mb | 2 Rd51 | MicroVMS V4.1 |
| VAX 11/780 | Yes | 8Mb | 3 RA81 | VMS V4.0 |

**Table 1:** System Configurations

| | | Whetstone IPS | |
|---|---|---|---|
| Processor | O.S | Single Precision | Double Precision |
| MicroVax II/fpp | Ultrix | 383k | 354k |
| Vax 11/750/nofpp | 4.2bsd | 126 | 105 |
| Vax 11/750/nofpp | System V | 138 | 114 |
| Vax 11/750/fpp | Ultrix | 304 | 266 |
| Vax 11/780/fpp | Ultrix | 480 | 415 |
| MC500/nofpp | RTU | 43 | 17 |
| MC500/fpp | RTU | 549 | 296 |
| MC5600/fpp | RTU | 978 | 850 |
| MicroVax II/fpp | VMS | 895 | 666 |
| Vax 11/780/fpp | VMS | 1311 | 838 |

**Table 2:** Whetstone Benchmark Results

and languages that use record and pointer data types (e.g. Ada[1]) become more common.

These results show dramatic differences between compilers and machine architectures. Results were obtained without specifically invoking optimizers, although some systems have significant optimization embedded in the compiler. Most Unix systems, and all those tested, convert all single precision calculations to double precision and back. VMS results are provided to show just how much of an impact this conversion can have. Note the Unix results are not that different between single and double precision while the VMS results vary by close to 300k whetstone instructions per second. The latest release of Berkeley Unix, 4.3bsd, has compilers and libraries that support single precision arithmetic. Several vendors specialize in development of optimizing compilers for Unix systems. Masscomp's MC 5600 performance reflects this trend. DEC now provides the Vax/VMS Fortran compiler as an option for Ultrix. This has created a flurry of excitement within the Unix community at large as it is considered one of the better compilers available, although it was not available on the test system. Whetstone performance for particular machines is improving with advances in compiler technology as well as hardware architecture.

*Dhrystone Results*

The dhrystone[WEIC84] benchmark is designed to more closely represent the performance of a typical application program that is not numerically based. In addition to computing the number of dhrystone instructions per second that these systems run, the number of lines of code per minute (LCPM) for the C and Pascal compilers was determined. As shown in table 3 the lower number of dhrystones represents better performance of the given compiler/o.s/processor combination.

*I/O and Other Assorted Results*

The following results represent a subset of those typically run by systems engineers within Sanders who are faced with the task of system specification. In addition to these more application specific benchmarks are also used, an example of one such benchmark is presented later.

Although the individual disks are shown in table 4 reality is that the controller exerts a large influence on the I/O performance. The controllers used were those distributed by each vendor with that particular configuration. Many controllers for drives, like the eagle and fujitzu, are available. At the time of the evaluation DEC had announced a new controller providing a 20% increase in performance. Although such a configuration was not available to us at the time of the initial evaluation some tests have since been run on an Ultrix V1.2 Microvax with RD53 disks and the new controller. Indeed results indicate a 42% increase in performance for sequential and random read/writes access. Read and write measurements are based on 512 byte blocks. This can be somewhat misleading as many Unix systems now support 1024 byte block sizes for file systems. 512 byte blocks correspond to a single page on the Vax systems. The increased block size for file systems leads to an overall increase in system throughput as testing on the Ultrix V1.2 MicroVax, previously mentioned, shows. Increasing the buffer size in the benchmark software to 1024 bytes resulted in a 14% increase showing a high increase in system throughput.

[1]Ada is a trademark of the DoD

| | | 1Mil. Function Calls | | |
|---|---|---|---|---|
| Processor | O.S | Real | User | System |
| MicroVax II | Ultrix | 23.9 | 23.7 | 0.0 |
| Vax 11/750 | 4.2bsd | 42.3 | 41.5 | 0.6 |
| Vax 11/780 | Ultrix | 34.8 | 26.0 | 1.0 |
| MC500 | RTU | 24.4 | 23.4 | 0.1 |
| MC5600 | RTU | 8.5 | 8.4 | 0.0 |

**Table 5:** Function Call Performance

Random and sequential read test results shown for the Masscomps were not expected and remain unexplained at this time.

Remaining benchmark results are given with three times, real, user, and system. Real time is not the most important of these results as it tends to reflect delays beyond control, such as writing termination notices to the terminal. In the case of systems tested in multiuser mode the real time results were impacted the most. User and system time are reflective of actual processing time. It is recommended that any comparisons made use these as a basis.

Results shown in table 5 were obtained by executing one million calls to a null function. It is indicative of function call overhead rates.

Pipes are a unique concept originating in the Unix environment. There are many uses for pipes, from networking to filter connections. *Piping* is a way to connect the output from one program to the input of another. Any program that can read from standard input, usually the terminal can read from a pipe and likewise for writing to pipes. Such programs are called filters. Consider the following example:

```
$ who | sort
```

In this example the vertical bar implies the use of a pipe. The output of the who program becomes the input for the sort utility. This concept eliminates the need to create temporary data files and is very useful in prototyping applications. Read and write performance of pipelines has a large impact on all users of the system.

As with any system, users are going to have an interest in the overhead encountered when using system calls. The getuid system call reads the system password file and

| Processor | O.S | Compiler | LCPM | Dhrystones |
|---|---|---|---|---|
| MicroVax II | Ultrix | C | 1834 | 720 |
| MicroVax II | Ultrix | Pascal | ? | 860 |
| Vax 11/750 | 4.2bsd | C | 2017 | 1356 |
| Vax 11/750 | 4.2bsd | Pascal | 1110 | 1539 |
| Vax 11/780 | Ultrix | C | 2478 | 695 |
| Vax 11/780 | Ultrix | Pascal | 1967 | 777 |
| MC500 | RTU | C | 1266 | 896 |
| MC500 | RTU | Pascal | Not Available | |
| MC5600 | RTU | C | 3300 | 306 |
| MC5600 | RTU | Pascal | Not Available | |
| Vax 11/780 | VMS | C | 3400 | ? |
| Vax 11/780 | VMS | Pascal | 3180 | 946 |

**Table 3:** Dhrystone Benchmark Results

| Processor | O.S | Disk | Writes | Reads | Random Reads |
|---|---|---|---|---|---|
| MicroVax II | Ultrix | Rd52 | 5.9 | 4.1 | 2.0 |
| Vax 11/750 | 4.2bsd | RM05 | 5.5 | 2.8 | 2.1 |
| VAX 11780 | Ultrix | RP07 | 3.9 | 1.6 | 1.1 |
| MC500 | RTU | Fujitzu | 4.0 | 3.2 | 4.1 |
| MC5600 | RTU | Eagle | 1.7 | 1.3 | 3.7 |

**Table 4:** 1000 Write/Reads to Disk

| | | 5000 Read/Writes | | |
|---|---|---|---|---|
| Processor | O.S. | Real | User | System |
| MicroVax II | Ultrix | 19.8 | 0.3 | 9.2 |
| Vax 11/750 | 4.2bsd | 41.4 | 1.1 | 20.5 |
| Vax 11/780 | Ultrix | 26.4 | 1.0 | 10.7 |
| MC500 | RTU | 16.0 | 0.0 | 6.8 |
| MC5600 | RTU | 8.9 | 0.1 | 7.7 |

**Table 6:** Pipe Performance

| | | 50,000 System Calls | | |
|---|---|---|---|---|
| Processor | O.S. | Real | User | System |
| Microvax II | Ultrix | 9.8 | 0.3 | 9.2 |
| Vax 11/750 | 4.2bsd | 16.4 | 2.2 | 14.0 |
| Vax 11/780 | Ultrix | 9.0 | 0.4 | 5.2 |
| MC500 | RTU | 10.8 | 0.4 | 10.1 |
| MC5600 | RTU | 6.4 | 0.2 | 4.1 |

**Table 7:** System Call Performance

retrieves the user identification number for a particular user. The associated call overhead does not necessarily reflect the overhead rates for all other system calls, it is a call used by many programs. Test results for the system call benchmark are shown in table 7. The Vax 11/750, with 11 users, executes between 40 and 300 system calls per second. This was determined by running several 30 second samples using a system utility that computes the system call rate. Such use reflects what can be considered typical use at Sanders, not a heavy system load (although not everyone would agree to that!) and is therefore representative of loading to be expected on an Ultrix based MicroVax II. There is a significant increase in performance of some utilities that can be made by reducing the number of system calls they make. This is an area DEC has been concerned with and results of their activities can be seen in more recent versions of Ultrix than the one reported on herein.

Sorting is indicative of general file manipulation capabilities of a system. Results shown in table 8 are based on alphabetic sorting of 5000 words, each word on a line by itself in the input file.

Timing information for edit sessions and nroff document processing are not included in the above tables for several reasons. Benchmarks intended to reflect editor performance are based on the programmable line editors available with all Unix systems. The results are in no way reflective of the screen oriented editors, emacs and vi, that are used by many Unix users throughout Sanders. Full

| Processor | O.S. | Sort 5000 Words Real | User | System |
|---|---|---|---|---|
| MicroVax II | Ultrix | 7.3 | 6.2 | 0.3 |
| Vax 11/750 | 4.2bsd | 10.2 | 8.8 | 0.8 |
| Vax 11/780 | Ultrix | 6.3 | 5.2 | 0.4 |
| MC500 | RTU | 10.3 | 9.0 | 0.6 |
| MC5600 | RTU | 4.0 | 3.1 | 0.3 |

**Table 8:** Sort Performance

screen editors have memory requirement and cpu utilization rates many times that of the line editor. Experience on the Ultrix based Vax 11/750 shows that these editors use as much as 35% of available memory and 25% of the cpu. The Vax 11/750 supports up to 10 users depending on specific applications being run.

*Signal Processing Results*

Signal processing applications, including radar, speech, and image processing, tend to be the most cpu intense programs run and are true hogs. They also tend to be representative of the work done on Sanders Unix based systems. A large spreadsheet type application, developed for internal use at Sanders, that provides access to several hundred algorithms has also become a useful benchmarking tool. Called SPPEED,[2] this tool is used for both design of signal processing systems and algorithms as well as for signal analysis. A subset of this application, used for radar simulations, was used during the MicroVax II evaluation with results from two small data sets shown in tables 9 and 10. This subset program has used as much as 14 hours of cpu time collected over a period of days on the Vax 11/750. Estimates for use with larger, but not atypical, data sets have reached two weeks of real time processing on the Vax 11/750.

Two cases are represented below; the first is a small set of data using fast fourier transforms (FFTs) while the second includes more complex signal to noise ratio (SNR) sampling. These results, shown only for Ultrix based systems, provide insight into the downfall of synthetic benchmarks. Whetstone results for the MicroVax and Vax 11/780 show a more significant difference in performance than that found when executing a true application.

## Command Set and Networking

Oft criticized for lack of a complete command set it was found that V1.1 of Ultrix 32m was satisfactory for most potential applications. Franz Lisp was not included with the distribution and just how to get it was not clear although many DEC support specialists seemed to think it was pos-

[2]Signal Processing Package for Engineering Design

| Processor | Case 1 Real | User | System |
|---|---|---|---|
| MicroVAX | 23 | 21.1 | 1.0 |
| VAX 11/750 | 30 | 26.7 | 2.3 |
| VAX 11/780 | 27 | 22.6 | 3.1 |

**Table 9:** FFT Performance

| Processor | Case 2 Real | User | System |
|---|---|---|---|
| MicroVAX | 1:04:39 | 3868.2 | 4.7 |
| VAX 11/750 | 1:41:02 | 5097.8 | 191.8 |
| VAX 11/780 | 1:04:36 | 3774.1 | 55.2 |

**Table 10:** Signal to Noise Ratio Performance

sible. This made evaluation for use with AI development a bit difficult since Common Lisp was not available to us for the evaluation period either. Lack of commands such as finger was not considered to be a major catastrophe, just an inconvenience.

Neither TCP/IP nor DECnet were tested due to limitations of the hardware configuration of the MicroVax used for the evaluation.

## Documentation and Installation Procedures

The documentation provided with the evaluation system was for Ultrix 32, not 32m, and it was found to be satisfactory. Additional examples of how to use particular tools and concise installation procedures were of particular benefit to some of the evaluation staff.

## Conclusions

There are many conclusions that can be made regarding the MicroVax II, the primary one being that it makes no real difference what operating system is runing. The relative performance is the same as that of a Vax 11/780 and in many cases better. The following list highlights those areas that were important to this evaluation:

- The MicroVax II/Ultrix combination provides a quality, production use environment for software development
- Ultrix, and other Unix based systems, are no longer *too researchy*
- It is not the best combination for use in signal acquisition/processing environment, VAXELN is a bit quicker!

- The minimum recommended system configuration for any MicroVax based system should be 3Mbytes of memory, 2 RD53 disks and a DEQNA for networking.

## References

[CURN70] H.J Curnow and B.A. Wickman. *A Synthetic Benchmark* Computer Journal, Vol. 19, No. 1, 1970

[WEIC84] R.P. Weicker. *Dhrystone: A Synthetic Systems Programming Benchmark* Communications of ACM, Oct. 1984, Vol. 27, No. 10, pp 1013-1030

# XDELTA/DELTA command strings

**B.C. Leahy**
Magic One
1971 Mt Pleasant Rd
San Jose, CA

Command Strings in XDELTA/DELTA are useful tools for displaying information that would be otherwise hard to obtain. What the "Command String" commands allow you to do is to display register and location contents, plus ASCII text, and then continue excution if desired. In a sense, they allow you to tailor the DELTA/XDELTA debugger to your needs. Two commands in XDELTA/DELTA allow you this capability, they are: `;E` and the Complex Breakpoint option of `;B`.

The directions for how to use the `;E` command are somewhat misleading in the manual. The key point that the VMS 4.4, and previous manuals, fails to make is that you must close the location before you begin to execute the commands you have stored. The XDELTA/DELTA command that is used to close a location is carriage return. Below is an example of a command string that prints out the contents of R0.

```
7FFE1600/00000000  '<CR>R0/'
./00000000
```

Note, ¡CR¿ stands for carriage return. The location 7FFE1600 was chosen because it contains all zeros, the name of this location is CTL$A_COMMON. The stored command must be followed by a zero byte as specified in the manual, (See VMS Device Driver Manual chapter 15, or Manual on XDELTA/DELTA). Once the command has been stored in the memory location, you can execute the commands stored in that location by typing the address followed by `;E`. For example:

```
7ffe1600;E
```

The above examples are very simplistic in their content but they show a necessary fact and provide the foundation for further development. Command strings can be written so that they print out the necessary information and continue executing, just by tacking on the `;P` command. For example, the command below will print out R0 followed by R1, R2, and then continue execution from that point on.

```
7ffe1600/00000000 '<CR>R0/<LF><LF>;P'
```

Each linefeed command, denoted by ¡LF¿, prints out the contents of the next location, in this case R1 followed by R2. By putting this command string in a complex breakpoint, the user has the ability to view the contents of memory locations and registers every time the code reaches this breakpoint. The syntax for the complex breakpoint command is:

```
bkpnt loc,bkpnt #,mem loc,strng addr;B
```

An example:

```
234,2,r0,7ffe1600;B
```

Although the above examples are useful to display memory contents, they lack the textual information provided by the english language. English text can be built into these examples by making some parsing strings. The XDELTA/DELTA commands typically used for parsing strings are: ", [L, [W, ., +, Q. These commands control the setting of various modes and incrementing, or decrementing of current memory location.

Let's see how the parsing works by starting with a simple string in a very simple macro program in Program 1.

Program 1 will print out the word HI when the address of the parser string is typed in followed by a `;E`. The parser string starts at address 204 for this program. How the parser works is as follows: first a carriage return is issued to close out the location. Then the Q defines the current address. Four is subtracted from the current address to set the location to CRLF. Next the output mode is set to longword, and the double quote command is used to output ASCII. The ASCII longword output consists of a carriage return, linefeed, and the two letters HI. Try it out.

Note, that in order to use XDELTA/DELTA, the user must have either booted the system with XDELTA, or used a logical variable name assignment for DELTA. Below is an example of the program SIMPLE run with DELTA.

```
$ DEFINE LIB$DEBUG SYS$LIBRARY:DELTA
$ MAC/DEBUG/LIS SIMPLE
$ LINK/DEBUG SIMPLE
$ RUN/DEBUG SIMPLE
DELTA Version X2.2

0000020D/CLRL   R6       204;E
00000204
HI
```

Again, the example above is rather simple. See Program 2 for an example of more complexity. Notice in the STRING that all of the words appear run together. When the words are output they will not be run together because XDELTA/DELTA inserts a space between the output commands. This fact created a problem for outputting

words longer than four letter. To overcome this problem a back space was inserted in the ascii text string.

Program 1

```
        CR = ^O15                       ; CARRIAGE RETURN
        LF = ^O12                       ; LINEFEED

CRLF:   .BYTE    CR,LF                  ; OUTPUT CR,LF
STRING: .ASCII   ^HI^                   ;SIMPLE TEXT

PARSER: .BYTE    CR                     ; MUST CLOSE LOCATION BEFORE COMMAND STRING
        .ASCII   ^Q-4[L"^               ; OUTPUT A CR,LF, THEN THE TEXT.
                                        ; Set Q to Parser address, subtract 4 to
                                        ; obtain the address of CRLF.  Set output
                                        ; to longword mode, and display in ASCII
                                        ; a carriage return, linefeed, and HI
        .BYTE    0                      ; Terminate string with a zero byte

        .TITLE SIMPLE
        .ENTRY SIMPLE, ^M<>

        CLRL     R6                     ; DO ONLY ONE COMMAND SO THAT THE DEBBUGER
                                        ;STOPS
        RET
        .END SIMPLE
```

Program 2

```
        CR = ^O15
        LF = ^O12
        BS = ^O10

        .BYTE CR,LF                 ; carriage return, line feed
STRING: .ASCII ^ Hi,MynameisBarb^<BS>^ara ^      ; ascii string to be printed

PARSER:  .BYTE CR                   ; carriage return to close location
trial:  .ASCII ^Q-17[W".+2[L".+4[W".+2[L".+4[W".+2[L".+4"^

        .BYTE 0                     ; terminate with a zero

        .entry barb,^m<>

        clrl r6                     ;dummy command in order to get into debugger

        ret

        .end barb
```

# Trojan Horses, Worms, Viruses, and Robin

**Steven Szep**
Chase Manhattan Bank
New York, New York

## Abstract

This paper describes ideas and techniques which have been found to be useful in designing a software-based risk assessment module for enhanced computer security. We present background material on the current state of computer security as well as preliminary results.

## Introduction

Are we – the designers and builders of systems, the managers of installations – at fault for not providing adequate security?

It is our responsibility to protect the right of individuals and corporations to have private thoughts, restricted domains, confidential information, personal property, and reliable operation.

Security is a "hot" topic because of DP's ongoing evolution... Rapid proliferation of computing power occurs without addressing security concerns. Methods of perpetrating fraud keep pace with each technological advance in their prevention. Connecting internal systems with the outside world increases their vulnerability. Total dependence on corporate systems offer little chance for survival after extended downtime or disruption of service.

The fundamental objectives of a security management project must include:

- data integrity;
- confidentiality of information;
- continuity of services.

The case of the 414's brought to national attention the sorry state of computer security in the U.S.A. At last, breaking into someone else's computer was no longer categorized as a harmless "prank".

Communications networks and information systems can no longer be considered safe from criminals. Technological safeguards and law enforcement must now come together to keep them out of our systems.

## General security

To be trusted, your system must reliably enforce a precise policy for accessing the data it possesses, while it accomplishes the functions for which it was actually built. We must investigate:

- the user interfaces;
- operations which cause information to flow into, or out of, the system;
- places where the classification of information could be changed.

Quickly, we realize that "sensitive applications" should reside on stand-alone computers in sealed rooms.

To watch for abnormal usage patterns, historical data become a necessity.

Some control must be established over the types of information retained and their classification. Also, rules for destroying information must be developed.

Common sense says that backups are vital.

Each site should be responsible for developing its own system manager.

Every user should read and sign a form which defines computer abuse and lists its possible penalties. Each individual should then be allowed to hold ultimate control over his own personal information.

The major drawback of using passwords is that people do not protect their own. Password generation is seldom used. Password compromise is hardly ever investigated. And, systems cannot differentiate between a legitimate user and an impostor who has logged on with someone else's password.

Hardcopy is the wrong way to disseminate confidential information in your organization.

There must be procedures for keeping information secret. Such procedures include:

- "sectioning off" departments where secrets might be exposed;
- making these areas "off-limits" to all but qualified personnel.

Finally, improperly trained people can be the cause of a disaster in a computer center.

Criticality of information processing demands that we index the impact of the several types of risk/exposure and match our security measures against them:

- necessary to maintain daily business (major annoyance, minor loss);
- necessary to maintain statutory requirements (major disruption);
- necessary to maintain the business (severe disruption, disaster);
- not primary to business (minor annoyance).

Encryption has a way of instilling confidence in otherwise clever persons that files and programs are secure from intruders. Hopefully, the recent Lotus 1-2-3 episode will do for these people what the 414 case did for others.

Encryption overhead, if software-managed, is quite intolerable. Sharing data means sharing keys: overexposure of both is a result. Lost and destroyed keys lead to inconvenience, if not lost files.

Finally, since encrypted data must first be converted into plaintext before normal processing can occur, sensitive data becomes vulnerable to compromise.

### Access controls

The intruder who impersonates a valid user must be stopped from altering data or data flows within the system.

The insider who attempts abuse or sabotage must be caught in the act. Containment of the damage is our primary concern.

Which parts of your system demand protection from intruders? Employ secondary passwords, access control lists (ACL's), and alarms.

Finally, maintain useful audit trails. Determine the effectiveness of your security measures and review frequently.

The maintenance of appropriate audit trails, particularly for online transaction processing, is highly recommended as the "baseline" requirement for applications security and control.

Also of high priority are:

- monitoring authorization and access-control compliance;
- designing/maintaining a disaster recovery plan;
- continually testing your systems' security.

Your computer system will probably operate in several "modes" with respect to file security.

Common categories include:

- Convenience mode (User-friendly/shared)
- Confidential mode (Personal files)
- Restricted mode (Need-to-know)
- Proprietary mode (Technical/strategic)
- Secret mode (Compromise seriously damages the firm) .

Security concerns demand a two-pronged examination of your network:

- evaluate the basic "rules";
- consider the user-visible behavior.

Access controls must not block current operational efficiency nor adaptability to changing needs. We must isolate the several components from each other. The "trick" is to balance sufficient security against cost, functionality, and performance.

Password checking and automatic disconnect are minimal requirements. Disallowing the further use of accounts which are the targets of attacks is a prime example of a security measure which applies to both standalone and networked computer systems.

Hopefully, the network log files will play an improtant role in Robin's evolution.

Access control must provide the following features:

- verification of usernames/passwords
- restriction of access to data and resources (levels)
- protection of data and resources (domains, levels)
- monitoring the usage of all defined resources (audits, logs)
- immediately reporting security violations (alarms, reports).

Not to be forgotten are these management concerns:

- ease of installation
- administrative controls
- maintainability
- documentation
- flexibility .

Two common "holes" in many security plans are:

- lapses in auditing ongoing software development;
- neglectful treatment of departing personnel.

Fraud involving ongoing software projects necessitates numerous controls:

- library control of code and documentation;
- authorization of program changes;

- separation of duties between development and production areas.

Trapdoors and Trojan horses have been planted during the development phase of the software life-cycle.

Code bombs are a real problem. Deal only with reputable software vendors and consultants. Review all sources before committing your firm in critical applications. And, of course, retain an "escrow" copy in case the authors go bankrupt.

Before he leaves, get the departing individual to train his replacement. Later, be sure to remove all traces of his daily activities – command procedures, personal directories, and so forth – from your system.

### User profiles

Robin will be capable of using pre-recorded, stored characteristics to authenticate the identity of an individual who has logged onto the computer system using a valid username/password combination. Also, Robin will verify a user's capabilities to perform tasks on the system.

A "real world" problem is that we may reject legitimate users – if our confidence level is quite low.

Fundamentally, Robin is responsible for

- adding objects to a user's space;
- deleting objects from the user's space;
- preventing objects from one user's space from being re-used, as is, in another user's space;
- accounting/auditing;
- alarms.

The VAX Accounting Utility is useful as a system management tool for learning more about how your system is used, how it performs, and how your users use it. Its primary area of concern is environment manage- ment. Routinely-collected audit trails must be carefully scrutinized.

Accounting records are generated by various events:

- process or image termination
- system initialization
- login failures
- batch and print jobs.

An accounting record consists of a header and a number of information packets – depending on the type of information being recorded.

An identification packet contains: PID, owner PID, UIC, privileges, priority, username, account name, node name, terminal ID, job name, job ID, queue name, node address, and remote ID.

A resource packet contains: process/image start time, final status, image count, total CPU time, volume count, and working set, paging, and I/O information.

An image packet only contains the image name.

A print packet contains: job status, queue time, begin time, symbiont CPU time, printed page count, and QIO and GET counts.

A filename packet contains only a filename.

User data packets contain user information, in ASCII format, sent to the accounting manager via the $SNDACC system service.

### Capability scripts

We wish to create "files" which contain operations performed, as per the user's profile.

Obviously, this requires a system-wide, common user interface. Those of us who have experience with the latest micros and workstations would probably opt for a graphics-oriented, mouse-driven solution.

The VAX Authorize Utility is the system management tool for controlling access to your system and allocating resources to your users.

We must consider

- what kind(s) of data processing the individual user will perform;
- what level(s) of authority apply to these activities.

VMS currently permits us to narrowly define a user's potential activities via captive accounts:

- automatic-login accounts;
- user-specific LOGIN.COM files;
- network proxy logins.

VAX SPM is more than a performance monitor. Its statistics- gathering capabilities make it an ideal source of information for a resource controller program.

The data collected by SPM is of several types:

- processor usage per process
- processor usage by IPL
- processor usage by IPL for one process
- processor usage by IPL for interrupt stack
- system module usage.

### Resource controller

Our fundamental objectives again are:

- data integrity;
- confidentiality of information;
- continuity of services.

We must identify a (small) set of operations on a set of data objects upon which all conceivable security can be based. These "primitive operations" need to know the actual (implementation) structure of our data objects. (Naturally, handling error conditions is extremely important.)

We require the (complete) specification of all the relationships among the various data objects and whatever security functions we wish to perform.

We shall attempt to follow a typical project scenario... We shall extract the central ideas (data types + data structures). We will focus on methods and algorithms amenable to computer solution: namely, frames, scripts, value lists, etc. Finally, we will combine these components (objects and procedures) together into an efficient and robust whole.

### Object-oriented programming

A STRUCTURE gathers together many pieces of information into a fixed "pattern". If we think of a STRUCTURE as a box with many "pigeon holes", then each hole (called a SLOT) may contain information written on pieces of paper.

Each SLOT is labelled according to the type of information it contains. These labels are called ATTRIBUTES. The pieces of information placed inside the SLOTs are VALUEs. When VALUEs are placed into the SLOTs, an actual OBJECT comes into being.

Thus, a STRUCTURE defines ATTRIBUTES important in doing something, while an OBJECT is a specific INSTANCE of that STRUCTURE. We can think of a STRUCTURE as a PLAN for constructing an OBJECT.

The following objects are under consideration:

- User profile object (UPO)
- User daily history object (UDHO)
- Periodic action object (PAO)
- Alarm object (AO)
- User summary object (USO)
- Action daily history object (ADHO)
- User biography object (UBO) .

Once we understand what objects are and how to use them in programming, we must adapt them to our current needs: a rule-based, decision-making system. Basically, we must design the structure RESOURCE_ALLOCATION: it will define those things which must be considered in order to decide whether to grant "security clearance" to a user or whether to transmit an AO.

Our knowledge base (KB) consists of all objects relevant to the user and the system – that is, UPO, UDHO, PAO, USO, UBO, and ADHO. This KB can be thought of as a decision tree to be operated upon by our program.

Although some of our rules will be have "yes-no" responses, experience has taught us that we may not always be certain that a particular "IF..., THEN..." rule is correct. Neither can we be certain that the values provided when instantiating a variable are 100% correct. These complications lead us to employ certainty factors and confidence levels.

When a certain slot is filled, this will cause a procedure to be performed. This procedure will be responsible for generating alarms. Or, depending upon how we wish to escalate the risk manager's response, even shutdown the computer system. The level of response could be configurable by site management.

### Status Report

We have identified our problem domain: risk management. And, we have formulated specifications for the three basic software modules:

- BP: biography prober
- RE: risk evaluator
- RA: resource allocator

There is now a prototype of BP available. Most of the current work is being done in the area of RE. Once RA is also completed, our risk manager will be ready for testing.

Our short-term goal is to have a working program for a single CPU. This version of Robin is called "Morisot".

The network, and final, version of Robin will come later.

### Glossary

Here is a glossary of some frequently used terms in the area of system security and access control.

**Abuse** - Misuse of a system's resources.

**Access control** - Oversight and management of the ability to use system resources.

**Action** - Performance of a particular task.

**Agent** - The user who gains access to this resource on the system. (The user requested the allocation of a specific resource, and our program fulfilled this request.) For example, a programmer typing at his terminal.

**Alarm** - A warning notice.

**Allocation** - Distribution of one or more system resources to a user.

**Attribute** - A property of an object, stored in slots inside structures.

**Authentication** - Establishing the trustworthiness of a user, via the "prober" module.

**Audit** - Formal examination and review of a log file.

**Browsing** - Obtaining information left in some part of the computer system after execution of a legitimate job.

**Capability** - The capacity for the appropriate use of a (requested) resource.

**Certainty factor** - Numerical weight indicating the degree of certainty that a rule or a fact is valid.

**Classification** - Categorization of tasks or goals.

**Confidence level** - See certainty factor.

**Confidential** - Containing information whose unauthorized disclosure would be prejudicial to its owner's interest.

**Containment** - Preventing the expansion of a hostile user's power over a system.

**Domain** - A topical area of knowledge.

**Encryption** - The encoding of data to help prevent theft.

**Exposure** - Risk.

**Frame** - Knowledge representation of an object's strcuture.

**Fraud** - Using system resources for one's personal gain.

**History** - Chronological record of significant events.

**Impersonation** - Using another person's authorization codes to achieve unauthorized access.

**Inheritance** - Process by which characteristics of one object are assumed to be characteristics of another.

**Instantiation** - Specification of particular values.

**Integrity** - Accuracy and consistency of the information in the system.

**Knowledge base** - The rules, facts, and strategies pertinent to a particular domain.

**Location** - The source of the trigger. For example, a terminal port.

**Log** - The record of all actions which have occurred in the system since the latest reboot.

**Logic bomb** - Leaving a set of instructions in an otherwise innocuous program which when set off do damage to your system.

**Monitor** - Software which oversees risk management.

**Object** - A conceptual entity with multiple attributes.

**Password** - The weakest link in the security chain.

**Profile** - A concise historical sketch of a user.

**Resource** - A specific component of the computer system requested by a user. For example, a CRT.

**Saboteur** - An insider who destroys his employer's property: files, data, etc.

**Secret** - Information meant to be shared by a select few.

**Script** - A strategy based on pre-defined, stereotyped situations.

**Slot** - Storage area in a structure associated with an object's attributes.

**Spoof** - Misleading the system into performing an operation which appears normal, but actually results in unauthorized access.

**Structure** - The knowledge representation of an object.

**Summary** - Historical abstract of a user's activity, extracted from the system's log.

**Time** - The occurrence of a trigger marked by system time.

**Trapdoor** - Allowing a user program to perform a normally privileged system function.

**Trigger** - An activity which causes the "security handler" to be invoked. For example, the use of the DCL command $SUBMIT.

**Trojan horse** - Tricking programs with legitimate access into doing things they ordinarily would not do.

**User** - Human person who performs some activity on the computer system.

**Value** - Information placed within slots inside frames.

**Verification** - Establishing the ability of a user to perform a specific task, via the "resource-allocator" module.

**Virus** - Internally taking control of a computer system via latent zapping.

**Wiretap** - Intercepting communications with the intention of obtaining authorization or other confidential data.

**Worm** - Achieving free-ranging access and the consumption of resources by one illicit program across a network.

**Zap** - Violating established access controls in order to modify, destroy, or obtain protected data.

## References


[1] Foiling the System Breakers: Computer Security and Access Control, J. Lobel (McGraw-Hill, 1986).

[2] Technocrimes, A. Bequai (Lexington, 1987).

[3] How Secure is Your Ethernet LAN?, G.B. Williamson, in *Pageswapper* Vol. 8 No. 1 (August, 1986).

[4] Desperately Seeking Access: Identifiers, ACL's, and Alarms, S. Szep, in *DECUS Proceedings U.S.A.* (Fall, 1985).

[5] Security Considerations for Network Access, S. Szep, in *VAX SIG Session Notes U.S.A.* (Spring, 1986).

# A VMS RESPONSE LOGGER - WHAT THE USERS THINK OF RESPONSE TIME

Robert B. Goldstein, Daniel P.B. Smith, Rivkah Stabiner
Eye Research Institute of Retina Foundation
20 Staniford St
Boston, MA 02114

## ABSTRACT

A program has been written that queries the user, upon logout, for his judgment of the quality of the system response. The results of the responses are analyzed to provide an objective measure of these subjective impressions.

The user is asked by the program to give the system a letter grade of A,B, C, D or F. A record is then written to a log file containing the grade and other information such as number of users, time of day, and user category.

Often users will grade the system based on overall satisfaction, and not solely upon system response time. Therefore, by periodic scanning of the response file for low scores, we can determine if any users are having trouble with the software or procedures.

The program has been used on three systems at two different sites. The results generated by the program have proven extremely useful for judging the effect of tuning efforts, evaluating the impact of adding new applications, helping to decide if additional resources are necessary, and determining the system load versus time of day.

## INTRODUCTION

The Response Logger is a program that, at logout time, queries a user regarding his opinion of the response of the system during the session. The results are collected, analyzed and plotted, thus giving an objective measure of users' subjective impressions.

One reason the program was written was to assist in tuning efforts. The overall average response as determined by this method gives a long-term measure by which to judge tuning effectiveness. Using an overall average of this type is appropriate since it is impossible to reproduce identical workloads in a variable timesharing environment.

Also, when tuning many statistics are obtained from SPM and MONITOR, but these statistics tell you nothing about how your changes effect USER perceptions.

## WHAT THE USER AND SYSTEM MANAGER SEES

Figure 1 shows the dialogue that occurs when a user types 'logout'. If he does not answer the query, the system gives him two more tries before proceeding with the rest of the logout process. If he types a '?', the explanation of the meaning of the grades is presented. These meanings are defined in reference to response time; however, a user would occasionally give an 'F' grade due to problems with data or programs.

Periodically (typically once/month) the system manager can generate a report as shown in Figure 2. This report gives the overall grade for the month as well as the average grade given by different categories of users.

```
Response Scores:

 Overall :       3.54 (SEM = .02, 1647 sessions)
 Progammers :    3.57 (SEM = .04, 293 sessions)
 ORACLE users: 3.72 (SEM = .03, 545 sessions)
 Wordll users: 3.35 (SEM = .04, 470 sessions)


Response Scores:

 Overall :       3.81 (SEM = .03, 339 sessions)
 RS/1 users :    3.84 (SEM = .03, 167 sessions)
```

Figure 2. Monthly Report of System Response

Figure 3 shows response scores as a function of number of users. This plot is the only one showing error bars. On other plots the error bars are within the symbol. Note that a 4.0 grade is not obtained even with only 1 user on the system.

Figure 4 shows the response as a function of hour of day. This plot illustrates that the program is sensitive enough to show the 'lunch time' bump.

Figure 5 shows results over a long period of time. Various events are labeled on the plot, and their effect on overall system response can easily be seen. Figure 6 is the same type of plot but for several other categories of users.

```
$ lo
How good was system response? Type question mark or A, B, C, D, F:


How good was system response? Type question mark or A, B, C, D, F:
?


During this session, how fast did the system respond to your commands?
Please give a letter grade:

     A -- Good
     B -- Satisfactory, but slower than usual
     C -- Slow enough to be a problem at times
     D -- Interfered with getting my work done
     F -- Logging off because system is unusable

How good was system response? Type question mark or A, B, C, D, F:
A
00:02:06 connect  0.11 units   00:00:12 CPU  0.14 units  0.25 total units
  GOLDSTEIN    logged out at  2-AUG-1986 11:36:10.59
```

Figure 1. Logout Dialogue

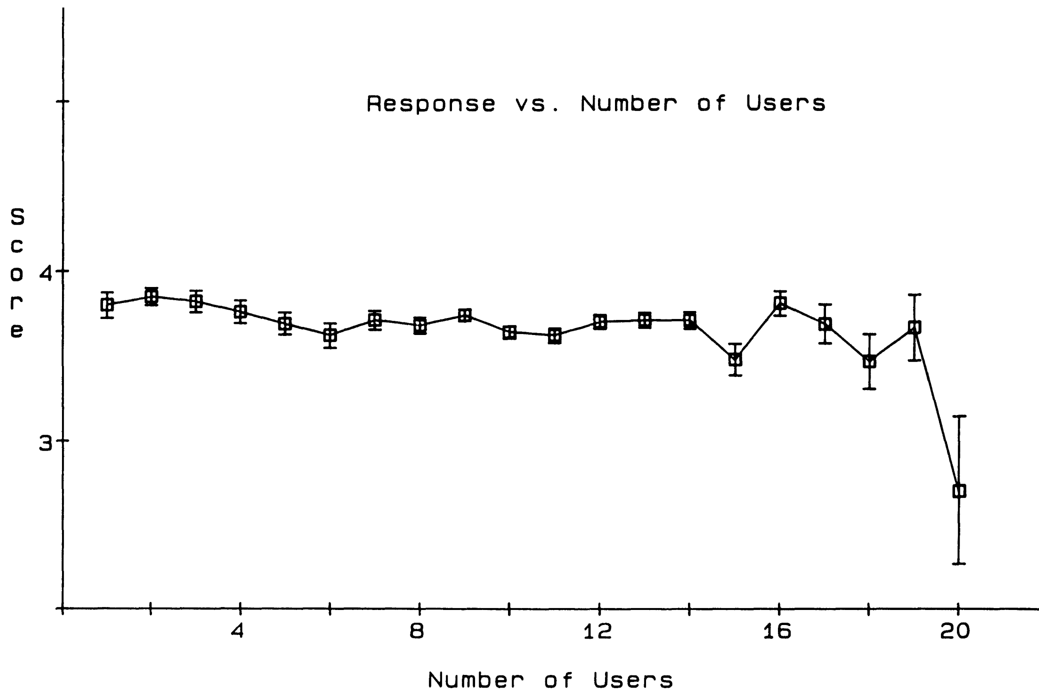



Figure 3. System Response by Number of Users

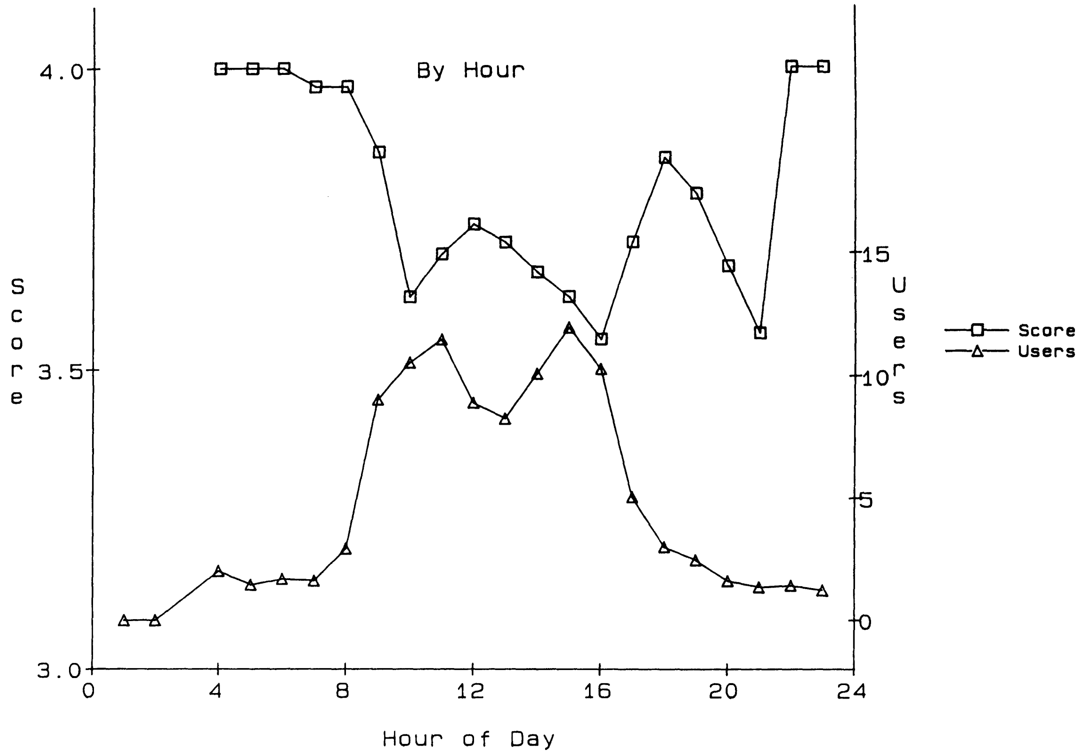



Figure 4. System Response by Hour of Day

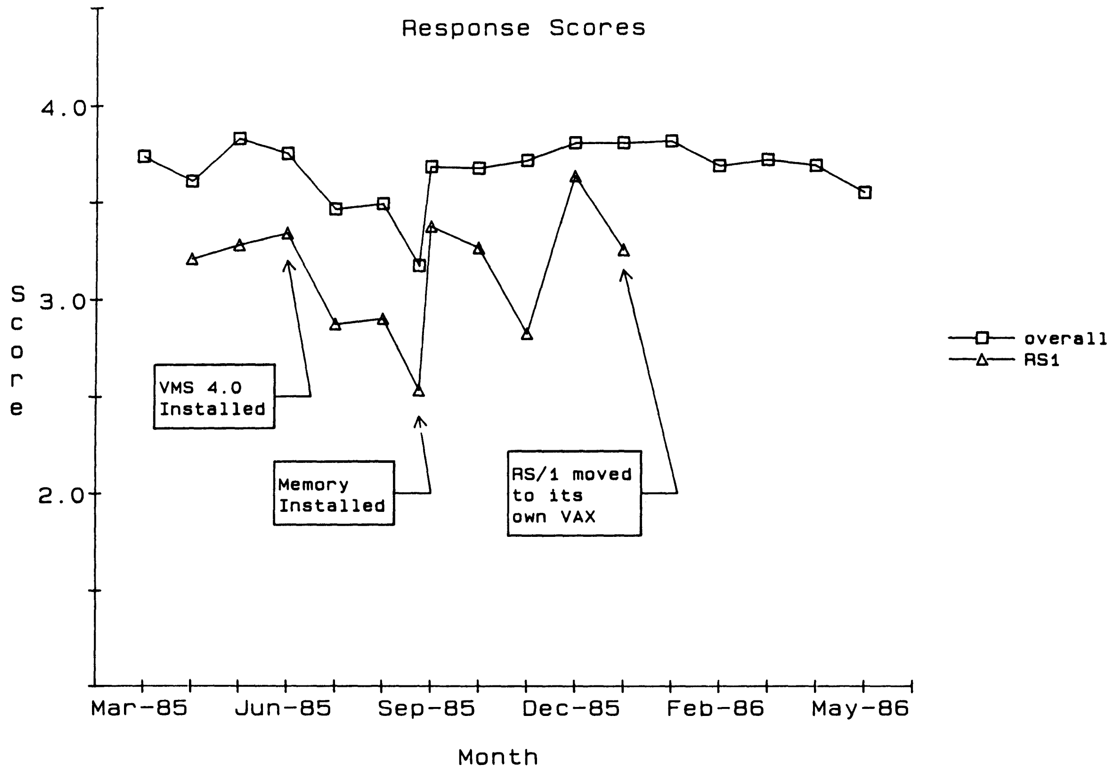



Figure 5. 18 Month Log of System Response

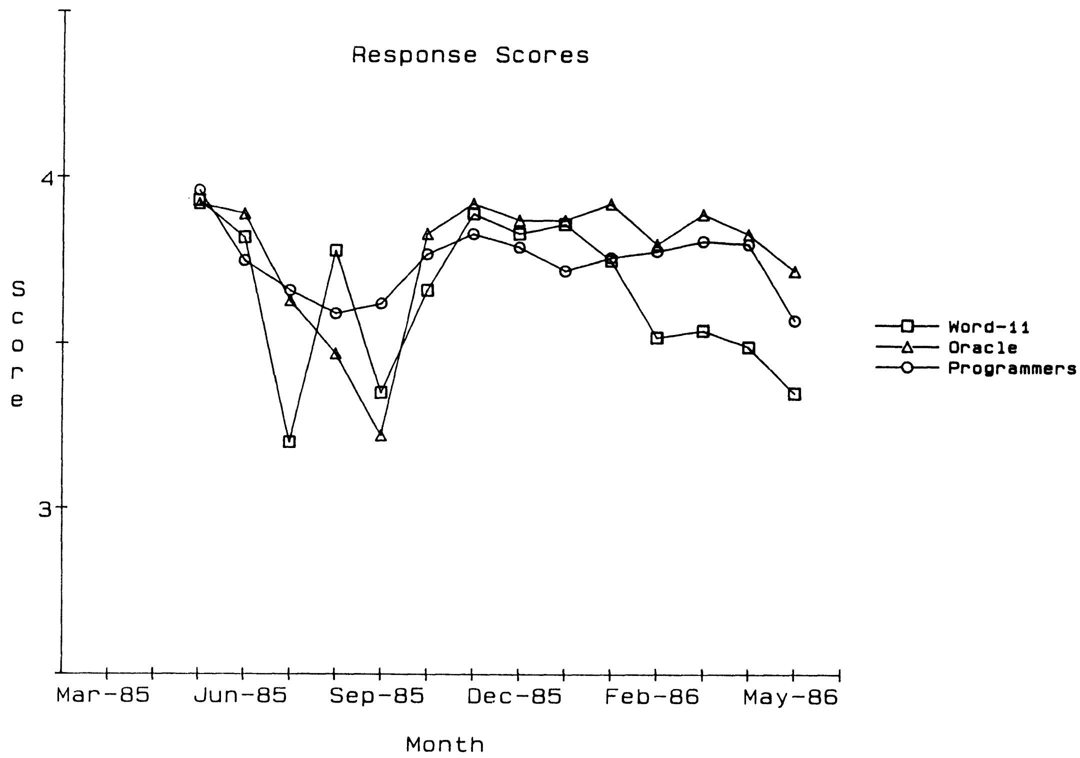



Figure 6. System Response for Various Categories of Users

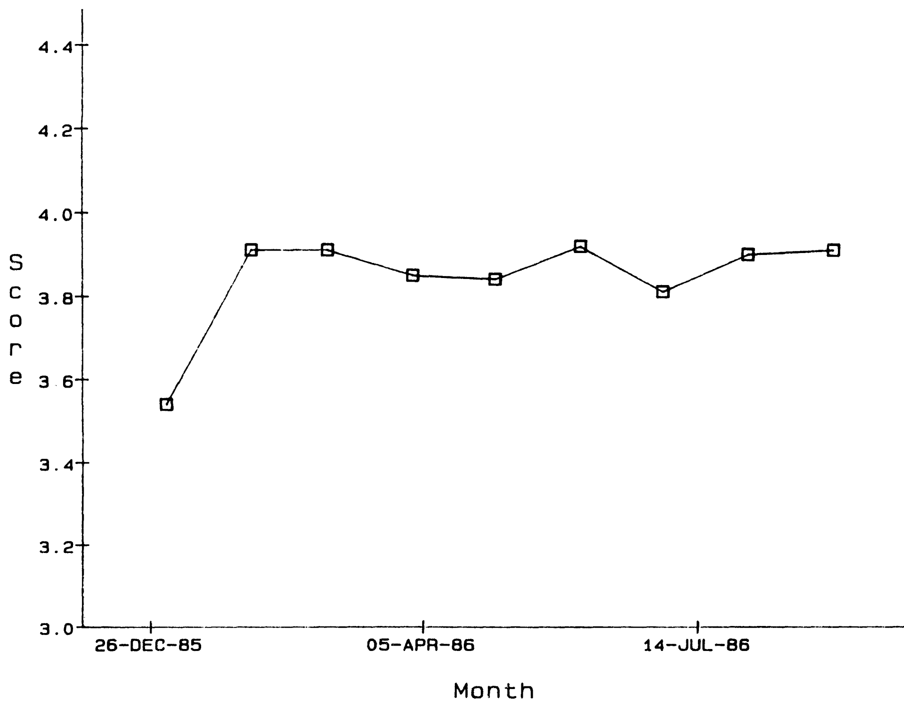



Figure 7. Response Scores for Node SCI

## Use On Other Systems

The program was used to assist in tuning the Tufts Image Analysis Facility VAX750. Before tuning, overall score was 3.30 ±0.08. After tuning, the score was 3.73 ±0.04. These were averages taken during the month before and the month after tuning.

Figure 7 shows the response scores on our MicroVAX that is dedicated to running RS/1. They are generally much higher than the 780 scores since this machine is more lightly loaded than the 780.

## Technical Details

We have defined user categories as shown in Figure 8. Other sites may define their own categories. The definition of each user's category must be inserted into their login.com file.

```
1 - GENERAL
2 - WORD11
3 - RS1
4 - ORACLE
5 - PROGRAMMER
```

Figure 8. User Categories

A section of the response log file is shown in Figure 9.

```
                               .
                               .
1-JUL-1986 15:32:48.01   13u B=3 FRADELLA    WORD11
1-JUL-1986 15:34:59.88   12u A=4 SUTTON      ORACLE
1-JUL-1986 15:48:20.39   10u C=2 CROISETU    GENERAL
1-JUL-1986 15:52:49.70   10u C=2 KESSLER     PROGRAMMER
1-JUL-1986 15:53:56.00   12u B=3 TAUSEVICH   GENERAL
1-JUL-1986 15:55:55.76   11u A=4 SYSTEM      UNDEFINED
1-JUL-1986 16:00:52.01   11u A=4 SYSTEM      UNDEFINED
1-JUL-1986 16:11:55.86   12u C=2 LITCHMAN    GENERAL
1-JUL-1986 16:14:33.81   13u B=3 MABUCHI     GENERAL
1-JUL-1986 16:14:49.21   12u --- SUTTON      ORACLE
1-JUL-1986 16:39:28.12   12u B=3 FRADELLA    WORD11
1-JUL-1986 16:52:33.93   10u A=4 TAUSEVICH   GENERAL
1-JUL-1986 17:00:45.19   10u A=4 SCOTT       ORACLE
1-JUL-1986 17:05:40.74    9u B=3 GLENN       PROGRAMMER
1-JUL-1986 17:14:55.18    9u B=3 RICHARD     GENERAL
1-JUL-1986 17:23:34.69    7u A=4 SYSTEM      UNDEFINED
1-JUL-1986 17:26:19.57    6u A=4 SUTTON      ORACLE
1-JUL-1986 17:35:36.21    4u C=2 KESSLER     PROGRAMMER
1-JUL-1986 17:48:23.49    3u A=4 STRECKER    WORD11
                               .
                               .
```

Figure 9. A Section of the file RESPONSE.LOG

The username, date, time, number of users, grade and user category is given on each line. If a user fails to give a grade, "---" is written in the grade field. If a user category has not been defined, "UNDEFINED" is written in this field.

Since the response log file is ASCII, any programmer can easily, in any language, write a program that reads the file and produces a report. The program we wrote, RESPREPT, produces the statistics illustrated in all of the previous figures. However, it is important that the program not operate directly on sys$response:response.log, since a user might be logging out at the time RESPREPT is run. This would cause a file access conflict and the user's logout would fail. Instead, a copy is made of response.log, and RESPREPT reads the copy.

## Conclusion

This program has been put onto the DECUS SIG tape. We think that this technique is a simple, easy, **visible** way to get a measure of response time. It shows the USER that you value his opinion and that alone makes for a better environment.

# MÖBIUS: NEW DIRECTIONS IN MICRO AND HOST INTEGRATION
(Controlling Your PC from a VAX)

E. William Merriam, President
FEL Computing
10 Main Street - PO Box 72
Williamsville, VT 05362
(802) 348-7171

ABSTRACT

Möbius is a system for integrating micro computers and DEC host machines (VAX, TOPS-20, TOPS-10) which has recently been enhanced to include several advanced features. From the viewpoint of the micro user, Möbius has always allowed the host to be a true extension of the micro, where host resources (files, devices, printers) are accessed exactly as if those resources were physically resident on the micro. The recent enhancements support the viewpoint of the host user, where the micro becomes a sophisticated peripheral device under total control of the host. To the end user, whether principally oriented toward the micro or host, the connection between the two machines is completely transparent.

Möbius is a micro/host integration system that has recently been extended to allow full access to and control over the resources of a personal computer from within host programs. Möbius previously provided full transparent access to host resources, such as disk files, printers, and programs, thus creating a friendly environment from the microcomputer user's point-of-view. With the new extensions, that point of view is reversed so that the microcomputer becomes a peripheral device to the host, thus creating a similar environment from the host computer user's point of view. Since the new features have been added without sacrificing the old ones, the two environments can be freely intermixed, so that the micro and host machines become powerful co-processing partners.

This paper shows how VAX programs interact with Möbius to provide a symbiotic micro/host environment to the user. This paper will also briefly describe the new Möbius Task-To-Task Communication facility. Finally, the high-level Task-Force task description language that is now a part of the Möbius system will be discussed, including its user-defined menu package.

## BACKGROUND AND UPDATE

A paper published in the DECUS Proceedings for Dallas, Texas, 1986 entitled "Using Möbius to Extend 1022 and 1032 Capabilities to Personal Computers" described how the capabilities of the 1022 and 1032 data base management systems could be provided to microcomputer users with Möbius. Briefly, that paper showed how the 1022/1032 systems can be activated, data extracted from the data base and made available to a microcomputer program (such as Lotus 1-2-3) all by typing a single microcomputer command. It was shown how this was accomplished through a Möbius feature that allows a program on the micro (in this case, 1-2-3) to directly access a file on the host (in this case, the data extracted from the data base).

Also discussed was a feature of Möbius that allows a host device, such as a printer, to be accessed directly from micro programs (e.g., with the Lotus 1-2-3 "/PRINT" command). Other issues addressed include how Möbius supports the host user with convenient mechanisms for activating a built-in VT-100 terminal emulator and for configuring the micro/host environment; how the programmer is supported through a micro-based Advanced Programmer's Interface; and how the Information Manager is supported through a variety of host system and Möbius access mechanisms that allow micro users to be managed identically to terminal users.

Since that paper was written, many enhancements have been added to the Möbius system. Those that relate to the previous paper are:

Micro-User Transparency:
All 26 drives (A: - Z:) may be defined to be any combination of directories, files, DECnet modes, etc. on the host. Up to three microcomputer printers (PRN:, LPT1 - LPT3) may be defined to be host devices.

System configuration and management:
The host system manager may specify system-wide parameters and configuration information that apply to all Möbius users.

Programmer:
The Advanced Programmer's Interface has been expanded to allow a program to perform virtually any operation that a user can perform by typing at the keyboard.

The enhancements that relate to the current paper include:

- The ability to control a personal computer from the host (either from the host system prompt or from within a program such as a data base management system).
- The ability to control the Möbius micro/host environment itself from the host or the PC (again, either from the system prompts or from within programs).
- Full task-to-task communication, where a program running on one machine can send and receive data directly to or from a program running on the other machine.
- The ability to set the micro's clock from the host computer's and thus to synchronize all of the computer clocks within an organization.
- A high level task description language called "Task-Force" which allows non-technical users to quickly develop applications which tie together all of the facilities of the host and microcomputer, as well as the Möbius system itself.
- Auto-dial/auto-login

There are many other enhancements that are beyond the scope of this paper. Even those that are within the scope will be discussed only so far as they relate to the examples presented. More complete information may be obtained by contacting the author.

## NEW THINGS TO DO FROM THE VAX

This section will illustrate the capabilities of the integrated micro/Möbius/VAX environment as it appears to a data base management system user. We use System 1032 to illustrate the concepts, but they apply equally well to only DBMS, All-In-One, financial analysis, graphics, or other VAX-based systems. It is assumed that the user has logged into the VAX and that the Möbius/1032 environment is active. Later, we will show how these functions can be performed automatically, if required.

The new capabilities are illustrated by using the 1032 "USE" command. This command can be issued from anywhere that normal 1032 commands can be issued. It simply causes a text file containing 1032 commands (called a "DMC" file) to be executed. These are standard 1032 commands and the only difference is that Möbius is operating behind-the-scenes to invisibly cause actions to occur on the microcomputer.

### USE MOBIUS

The "USE MOBIUS" command causes the DMC file named "MOBIUS" to be run. The purpose of the program is to allow the 1032 user to define the micro/host environment. It prompts the user by displaying the message "Enter Mobius Command -->". Once the command is entered, 1032 calls upon Möbius to perform the desired function. For example, if the user enters

DEFINE V: HOST *.*

then from this point on, whenever a microcomputer program references disk drive "V:", it will actually be referencing the files contained on the user's VAX directory.

Similarly,

DEFINE D: HOST *.DMC,[.DATA]*.DMC

would cause the microcomputer's drive "D:" to reference all of the DMC files in the user's current VAX directory and ".DATA" sub-directory.

Pre-defined micro/host environments can also be set up so that it is not necessary for the user to create them each time an application is run.

### USE STATUS

Displays the status of the integrated micro/host environment, including the current definitions of the host-based disk drives (referred to as "Möbius Disks" or "Möbius Drives" in the remainder of this paper).

### USE PC

Prompts with the message "Enter PC command -->". When the command is entered, it is sent through Möbius to the microcomputer where it is processed. For example, if the command is:

DIR A:

then the floppy drive "A:" is activated on the micro and the directory information describing those files is displayed on the screen. When done, the user is returned to 1032.

Likewise, if the command is

WS B:MANUAL.DOC

then the WordStar (WS) text editor would be started and it would edit the MANUAL.DOC file contained on the user's disk drive "B:". In this case, the user would interact with the WordStar program, editing the file completely on the microcomputer. When the WordStar program is exited, the user is again returned to 1032.

Notice that <u>any</u> microcomputer command can be executed in <u>this</u> manner. For example, the command

D:

would cause drive "D:" on the micro to become the currently logged drive.

Notice further that the VAX files referenced as a Möbius Disk are fully available to a micro program started through the 1032 DMC file. For example, the command

WS FILMS.DMC

would cause WordStar running on the micro to edit the FILMS.DMC file on the VAX.

### USE 123, USE DBASE, USE WS

Starts Lotus 1-2-3, dBASE, or WordStar on the microcomputer. These DMC files operate similarly to the PC DMC file described above, except they are tailored to run the specified program without requiring the user to enter the specific command.

We have used three popular microcomputer programs as examples here, but <u>any</u> microcomputer program can be run in this manner.

### USE PRINT

Asks for the data to be printed and directs that data to the microcomputer printer. Thus, VAX users can take advantage of printers, plotters, or other special devices connected to their microcomputers.

### USE ENTRY

ENTRY causes the dBASE data base management system to be started on the microcomputer and to display a screen form for entering data into the VAX data base system. This screen form is entirely constructed using the features of dBASE, including all editing and range-checking operations. Thus, all of this high-overhead processing is off-loaded from the VAX. Since dBASE is used, the flexibility of a general-purpose data entry system is retained. In addition, if more complex data operations are required, they can be easily handled using the features that dBASE provides.

As each data base record is created, it is written to a file on the VAX through a Möbius Drive. When the data entry session is complete, the dBASE program terminates and the new records are merged into the VAX data base.

Since dBASE is so familiar to microcomputer users, we chose that system for this application. However, any microcomputer program that allows data entry, including one written by a user, could be used for this purpose.

### USE GRAPH

GRAPH causes data to be extracted from the VAX data base and to be written to a VAX file. Then, it causes the Lotus 1-2-3 spreadsheet program to be started on the microcomputer. Using a special feature of Möbius Task-To-Task Communication, commands are entered into 1-2-3 just as if they had been typed by the user at the keyboard. Thus, the VAX data base management system can not only start any micro program desired, but through Möbius it can also tell it what to do.

The commands sent by 1032 in this case cause 1-2-3 to read the data that was extracted from the data base into a 1-2-3 spreadsheet and then to activate the features of 1-2-3 for graphing the data. The user is then presented with a 1-2-3 menu which allows the selection of the type of graph desired. As before, when the micro program is finished, the user is returned to 1032.

Notice that in this example, the VAX performed none of the time-consuming graphic calculations or screen-handling operations. All of that was performed by 1-2-3 on the microcomputer using its already-existing facilities.

## AND MORE ...

The above examples illustrate only a few of the things that can be accomplished with Möbius and a VAX program working together. In general, a VAX program can activate, control, send and receive data to and from any program on the microcomputer. Of particular importance is that micro programs can be used that already exist at a site and that users are already familiar with. These programs can be used for data entry, extraction, or output operations and can be custom-tailored to specific applications to form friendly and flexible front-ends or output processors. Thus, the functionality of the VAX program is extended to take full advantage of the microcomputer and the VAX system is simultaneously offloaded.

## HOW IT WORKS

The examples above are implemented through two new features of Möbius: "Task-To-Task Communication" and "PC Control". While a complete description of these facilities is beyond the scope of this paper, an overview of them is provided here:

### TASK-TO-TASK COMMUNICATION

Möbius Task-To-Task Communication allows a program (task) on the host computer (in this case, 1032) to send and receive data to or from a program (task) running on the microcomputer. This data is sent over the communication channel using a protocol that ensures that it reaches the other task without errors.

The host task (1032) communicates with the micro task by using VMS "mailboxes". Mailboxes are referenced as if they are files, except that the information in them is passed to another host task, in this case the Möbius program itself. The host Möbius program communicates with the micro Möbius program, which in turn passes the information to and from the microcomputer task. This information is usually passed to the program through calls to the Möbius Advanced Programmer's Interface (API). Since API calls appear as normal MS-DOS system calls, virtually any microcomputer program that can make MS-DOS calls can access the API.

With Möbius Task-To-Task Communication, data is directed from one process to another by means of "User Messages". The most common form of User Message allows a program running on the host to communicate directly with a program running on the micro. Other User Messages are defined by the Möbius system to have special meaning. By using a combination of these User Messages, virtually any micro/host application can be created.

### PC CONTROL

Möbius "PC Control" allows the microcomputer to be completely controlled from the host machine. This is accomplished through a set of programs supplied with the Möbius system. These programs implement a special set of User Messages that cause specific functions to be performed on the microcomputer. For example, the "PC" program allows microcomputer commands to be issued from the VMS "$" prompt, in a similar manner as the 1032 "USE PC" command mentioned above. For example, the command

```
PC DIR B:
```

typed at the "$" prompt would cause the microcomputer to display a directory of the files on its "B" disk drive.

Likewise, programs called "123", "DBASE", "WS", etc. can be quickly created so that when they are run at the VMS "$" prompt (or in a DCL command procedure, or from another program, etc.) the desired program is run on the microcomputer.

While the 1032 DMC files discussed earlier and the programs supplied with the Möbius system are quite different from one another, they use the exact same Möbius mechanisms for controlling the microcomputer. These mechanisms are general-purpose and can be implemented in any program that can access VMS mailboxes. Therefore, it is possible to develop a system-wide set of useful microcomputer functions that can be accessed from within virtually any host program or from VMS itself.

One of the particularly interesting features of PC Control is the ability of Möbius to cause data to be entered into a microcomputer program as if that data had been typed at the keyboard. The data can be provided to Möbius on the microcomputer through a special Task-To-Task Communication User Message from the host machine or directly from a microcomputer program through the Advanced Programmer's Interface. By having both methods of keyboard data entry available, many applications become easy to implement that would otherwise be impossible (e.g., the 1032 "USE GRAPH" example given earlier).

## TYING IT ALL TOGETHER: TASK-FORCE

The features described so far represent powerful elements of the Möbius micro-and-host integration system, which allow countless applications to be performed which could never before even be considered. Coupled with the other features of Möbius itself, the VMS system, 1032 and the multitude of other VAX and microcomputer programs, a bewildering array of interactions can take place. To help develop applications which utilize all of these capabilities, a new language called "Task-Force" was created and is an integral part of the Möbius system.

Task-Force is specifically designed to allow applications to be easily created where those applications involve the interaction of multiple programs on different machines. In addition, a Task-Force program, instead of controlling other processes, may itself be the primary microcomputer program in an application. It may function as a front-end using its user-definable menu package. It may also function as a full participant in a micro/host Task-To-Task Communication application, since it contains complete facilities for sending, receiving, and acting upon all of the various types of user messages.

An overview of the facilities provided by the Task-Force language can be obtained by considering the following menu which is typical of what might be presented to a microcomputer user:

A. Setup
B. Dial
C. Login
D. VAX/VMS
E. Mobius
F. Mail
G. VAX DBMS
H. 123
I. dBASE
J. PC-DOS
K. Logout
L. Exit Menu

Task-Force allows this menu to be created (with optional titles and an attractive border) through a menu package that is delivered with the Möbius system. This package allows a menu such as this to be created in only a few minutes. More importantly, it can be modified just as easily, saving considerable program maintenance time and money.

When the "Setup" option is selected, the Möbius "MSETUP" program is run which is itself composed of a variety of menus. It is used to set various parameters such as communication channel speed, tab settings, colors for the terminal

emulator, etc. It is a normal microcomputer program which, like selections H and I, was developed with no knowledge that it would be accessed from another program. However, through Task-Force, all such programs integrate easily into a unified environment. When the MSETUP program terminates, the menu is again displayed.

The "Dial" and "Login" menu items activate functions which are themselves Task-Force programs that are delivered with the Möbius system. They perform the functions of dialing an auto-dial modem and automatically logging into a host system, respectively. As with MSETUP, they were developed independently of this menu application, but integrate into it nicely.

The "VAX/VMS" item enters the Möbius terminal emulator and either presents the VMS "$" prompt or, if the user has not already logged in, allows the user to log in manually. Notice that the terminal emulator has been entered under program control. This means that the program itself (the one that displays the menu) is temporarily suspended while the microcomputer operates as a terminal to the VAX. However, when the terminal emulator is exited, the program will resume and the menu will again be displayed.

Item E, "Mobius", is provided so that the user can interact directly with the Möbius system to set up the micro/host environment. If Möbius is not running on the host system, it will be started automatically. This is an example of how Task-Force can detect the status of the interacting programs and perform the appropriate action. This relieves the user of the burden of performing the actions or even of knowing that they must be done.

Another example of this occurs completely hidden from the user. That is, the Task-Force program that implements this menu determines if the micro's clock has been set. If it has not been set, as soon as Möbius is activated on the host, the program causes the micro's clock to be synchronized with the host's clock.

When the "Mail" item is selected, the VMS MAIL program is run on the host computer and the Möbius terminal emulator is entered. The user then interacts with the MAIL program as normal. When the program is exited, the menu is re-displayed. Any host program can be run in this manner through Task-Force.

The "VAX DBMS" item is similar to "Mail" in that the VAX "S1032" program is run. Unlike MAIL however, 1032 uses some of the Task-To-Task Communication and PC Control facilities of Möbius through the 1032 DMC files described in Section 1.0 of this paper. Because these facilities are so widely useful, they are automatically available whenever Task-Force programs (as well as others) are being run. Therefore, absolutely no programming whatever is required on the microcomputer in order to implement the capabilities previously discussed, other than to simply run the S1032 program.

Item J, "PC-DOS" activates the microcomputer command processor. This is really another example of running a micro program from a Task-Force program and illustrates the flexibility of the concept.

The "Logout" and "Exit Menu" options both provide additional menus that allow the user to select various actions that should or should not occur at this time. Thus, the Task-Force menu package allows multiple menus to be defined and referenced.

## THE INTEGRATED PC/HOST ENVIRONMENT

The examples of the 1032 DMC files and of the Task-Force menu system show two ways in which an integrated micro/host environment can be structured. Further, when running programs from the menu, these two structures merge into a symbiotic co-processing relationship. Such a relationship provides an unusually powerful system for the VAX user who also utilizes a microcomputer.

# VMS FILE I/O VIA QIO TO AN ACP

Al Tyrrill
Digital Consulting
Garden Grove, California

## ABSTRACT

I/O to file structured devices is usually done through VAX Record Management Services (RMS). However, QIOs can be issued directly to such devices through the associated Ancillary Control Process (ACP) or extended QIO processor (XQP). Doing so avoids the overhead of RMS and provides special operations not otherwise possible. The disadvantage is that the services of RMS are not available. This paper describes the capabilities of file structured I/O via direct QIO through ACPs, with high level language examples (Fortran and Ada).

## INTRODUCTION

The control flow of an I/O operation issued by a high level language program is illustrated in figure 1. The program issues calls to routines in the language's runtime system (RTS) which in turn issues calls to VAX Record Management Services (RMS). RMS in turn issues QIO system service calls to an Ancillary Control Process (ACP). An ACP issues QIO services to the driver for the file structured device, which in turn controls the device hardware.

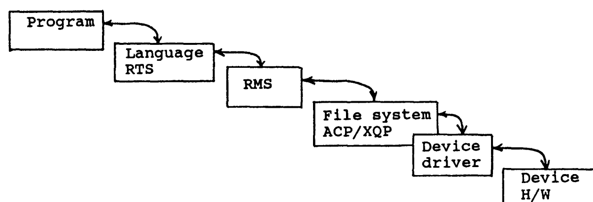



Figure 1) File I/O Control Flow

In VMS version 4.0 and later for ODS-2 structured disks (VMS native mode), the ACP is replaced by the extended QIO processor, which is kernel mode code running in the context of the calling process.

The language RTS is reponsible for the formatting of individual fields within a record. RMS creates and maintains the record structure of a file. The ACP or XQP maintains the file structure on the device. The device driver translates a generic logical structure to an actual physical device, which in the case of a disk will have a particular sector, track and cylinder arrangement.

It is possible to bypass the I/O features of the language RTS. This has several advantages. It eliminates the language RTS overhead, which in a language like Fortran can be significant. It makes the full functionality of RMS available to the application program, which is generally not available through the language I/O. The RMS interface is fully supported.

Disadvantages are that the language I/O functionality is lost (record formatting must be done by the application) and that RMS can be imposing to learn.

It is also possible to bypass RMS and issue QIOs from the application program directly to the ACP/XQP, as illustrated in figure 2. Advantages of doing so are that the overhead of RMS is eliminated. The QIO/ACP interface is supported and not hard to learn, although novices may need guidance.

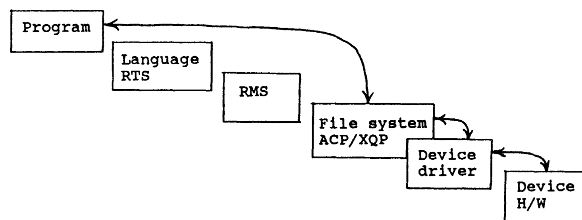



Figure 2) Direct Interface to ACP/XQP

Disadvantages are that the functionality of RMS is lost. In practical terms, files must be collections of fixed length, 512 byte directly accessed records. The documentation could be better (see reference 1) and users should prototype before incorporating this facility in application programs.

## ISSUING QIO FROM FORTRAN

Whenever VMS system services are to be issued from a Fortran program, files should be included in the program as shown.

```
INCLUDE '($SYSSRVNAM)'    ! system service names
INCLUDE '($IODEF)'        ! I/O function codes
INCLUDE '($SSDEF)'        ! system status codes

INCLUDE '($FIBDEF)'       ! control block
INCLUDE '($FIDDEF)'       !  structure definitions
INCLUDE '($ATRDEF)'       !   with codes
```

These files are in the library FORSYSDEF. $SYSSRVNAM contains INTEGER and EXTERNAL statements for each system service and commentary describing the required parameters. It is a large file and compilations will be speeded by extracting the needed parts into a separate file, once the services to be used have been identified. $IODEF defines symbolic names for the I/O function codes, $SSDEF does the same for the system service status (return) codes. $FIBDEF, $FIDDEF and $ATRDEF contain STRUCTURE statements defining the data structures required by the QIO/ACP interface.

A Fortran program that is to use the QIO system service must first assign a channel to the device or logical unit, as shown.

```
INTEGER*4  CHANNEL, RETURN_CODE
CHARACTER*20  DEVICE_NAME
 .
 .
RETURN_CODE = SYS$ASSIGN (DEVICE_NAME, CHANNEL,
1                         [acmode], [mbxnam])
IF (RETURN_CODE .NE. SS$_NORMAL) THEN
     {take corrective action}
ELSE
     {continue processing}
ENDIF
```

CHANNEL receives the channel number returned from the ASSIGN system service. RETURN_CODE receives the status code specifying the outcome of the system service. DEVICE_NAME contains a physical device or logical name that translates to a physical device. A logical name that includes a directory path may be used, but the path information will not be useable by the ACP.

It is important that the system service status code always be checked, otherwise a failed operation will have the appearance of a no-op. The parameters acmode (access mode of the channel) and mbxnam (logical name of the mailbox associated with the channel) are optional and not generally used in this application. Note that trailing optional parameters cannot be omitted. A system service with N parameters must have N-1 commas in the parameter list.

Use of QIO from Fortran programs will generally take the form illustrated. Computation is overlapped with I/O, as will usually be the case when this facility is used.

```
INTEGER  CHANNEL, RETURN_CODE, EVENT_FLAG/1/
INTEGER*2  STATUS_CODE(4)
 .
 .
RETURN_CODE =
2     SYS$QIO (%VAL(EVENT_FLAG), %VAL(CHANNEL),
3              %VAL(function), STATUS_CODE,
4              [astadr], [astprm],
5              P1, P2, P3, P4, P5, P6)
IF (RETURN_CODE .NE. SS$_NORMAL) THEN
     {take corrective action}
ELSE
     {processing to overlap I/O}
      .
      .
     RETURN_CODE = SYS$_WAITFR (%VAL(EVENT_FLAG))
     IF (STATUS_CODE(1) .NE. SS$_NORMAL) THEN
          {take appropriate action}
     ELSE
          {continue processing}
     ENDIF
ENDIF
```

CHANNEL is the value provided by the ASSIGN system service. RETURN_CODE receives the code describing the correctness of the call to the QIO service. It would indicate, for example, that a parameter was out of range. The value is available immediately after execution of the QIO call. STATUS_CODE receives the code describing the result of the operation requested by the QIO service. It would indicate, for example, that a specified file did not exist. Its value might not be available until some time after control returns from the QIO call.

EVENT_FLAG is the number of the VMS event flag used to synchronize the operation of QIO with the calling program. The flag (not the variable EVENT_FLAG) is cleared at the call to SYS$QIO and set when the requested operation is complete. SYS$WAITFR checks if the specified event flag is set, and if not, suspends execution until it becomes set. It is important that both the status codes are checked, otherwise failure will have the appearance of a no-op.

"function" specifies the operation to be performed by the QIO. "astadr" specifies the address of a routine that it to receive control when the requested operation completes. "astprm" is an optional parameter for the AST routine.

There are six function/device dependent parameters, called P1 .. P6. For QIOs directed to an ACP, these always take a particular form, to be described subsequently.

If overlap of processing and I/O is not necessary, then the SYS$QIOW system service can be used and reference to an event flag and SYS$WAITFR can be omitted.

When I/O operations are complete, the channel should be de-assigned.

```
RETURN_CODE = SYS$DASSGN(CHANNEL)
```

De-assignment will occur automatically during image rundown, so it is usually sufficient just to exit the program.

## ACP/XQP INTRODUCTION

The file structure on Files-11 devices is maintained by an ACP, except in the case of ODS-2 (VMS native mode) disks in VMS version 4.0 and later, where the extended QIO processor (XQP), kernel mode code that runs in the context of the calling process, performs the same function. Since the flow of control in an ACP is single threaded, concurrent accesses to the same file by multiple processes are serialized, at least on the same CPU. ODS-1 (RSX compatibility mode) disks cannot be accessed by more than one processor in a VAX cluster.

The control flow in the XQP is multithreaded and serialization of cuncurrent accesses by multiple processes, on the same or different CPUs, is implemented via the distributed lock manager.

The interface seen by application programs is the same for either an ACP or XQP and henceforth in this paper the term ACP will be used to refer to both facilities.

QIOs to an ACP are used to create, delete, open and close files and modify their allocation. QIOs to the device driver are used to read and write the file's data. Disk ACPs support seven major functions, as follows.

| | |
|---|---|
| IO$_CREATE | create directory entry/file |
| IO$_ACCESS | look up file in directory, open |
| IO$_DEACCESS | close file, write attributes |
| IO$_MODIFY | modify attributes, allocation |
| IO$_DELETE | delete directory entry/file |
| IO$_MOUNT | tell ACP volume mounted |
| IO$_ACPCONTROL | miscellaneous control functions |

Each of these symbols is equated to a code in $IODEF which can be used as the function argument in the QIO system service.

Four function modifiers are provided. Each symbol is equated to a mask in $IODEF which can be ORed with the QIO function argument to modify it effect.

| | |
|---|---|
| IO$M_ACCESS | open file |
| IO$M_CREATE | create file |
| IO$M_DELETE | mark file for deletion |
| IO$M_DMOUNT | dismount volume |

The modifiers have different implications for the different major functions and can sometimes be confusing. Note these modifiers are (in all but one case) different from the ACP subfunctions, which will be described subsequently.

The function/device dependent parameters P1 .. P6 have the same format for disk ACPs, as shown.

| | |
|---|---|
| P1 | File Information Block |
| P2 | Input filename |
| P3 | Output filename length |
| P4 | Output filename |
| P5 | Attribute list |
| P6 | -- unused -- |

Parameters P1, P2 and P4 are passed by descriptor, P3 and P5 are passed by reference. For P1 and P2, the Fortran programmer must build his own descriptors. In the case of P1, VAX Fortran rejects the %DESCR qualifier on RECORD types. With P2, it is usually necessary to set the length field of the descriptor, as will be described below.

The File Information Block (FIB), parameter P1, is the primary interface between the calling program and an ACP. It contains the following information.

| field | bytes | contents |
|---|---|---|
| FIB$L_ACCTL | 3 | access control flags |
| FIB$B_WSIZE | 1 | mapping window size |
| FIB$W_FID | 6 | target file ID |
| FIB$W_DID | 6 | directory file ID |
| FIB$L_WCC | 4 | wildcard position context |
| FIB$W_NMCTL | 2 | wildcard flags |
| FIB$W_EXCTL | 2 | extend control flags |
| FIB$L_EXSIZ | 4 | extend size |
| FIB$L_EXVBN | 4 | record to truncate to |
| FIB$B_ALOPTS | 1 | allocation option flags |
| FIB$B_ALALIGN | 1 | interp. of FIB$W_ALLOC |
| FIB$W_ALLOC | 10 | allocation location |
| FIB$W_VERLIMIT | 2 | version limit |
| FIB$B_AGENT_MODE | 1 | access protection mode |
| FIB$L_ACLCTX | 4 | ACL context |
| FIB$L_ACL_STATUS | 4 | ACL operation status |
| FIB$L_STATUS | 4 | alternate access control |
| FIB$L_ALT_ACCESS | 4 | alternate access mask |

For any given function, values do not need to be supplied for some or most of the fields, as will be described in the section on specific operations, which follows. Some of the fields have additional uses different than described above.

Disk ACPs support five subfunctions which can usually be invoked by several of the ACP major functions.

| Subfunction | Invocation |
|---|---|
| Directory lookup | FIB$W_DID nonzero |
| Access (open) file | IO$M_ACCESS modifier set |
| Extend file | Extend in FIB$W_EXCTL set |
| Truncate file | Truncate in FIB$W_EXCTL set |
| Read/write attributes | P5 present |

They will be described in more detail in the section on specific operations.

## FILE SYSTEM INTRODUCTION

All files on a Files-11 disk are uniquely identified by a three word record called a File ID, whose structure is shown in figure 3.

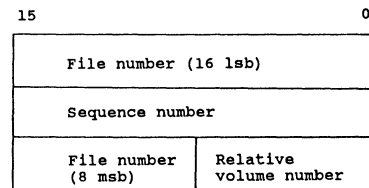



Figure 3) Format of File ID

Each file has a unique file number, a 24 bit value split into two fields. The sequence number is incremented each time a file number is reused, due to deletion and creation of a new file. The relative volume number is nonzero only for multidisk volume sets.

All DEC disks are organized as collections of 512 byte blocks, but each model has its own number of sectors/track, tracks/cylinder and total cylinders. Physical blocks of a disk are described in terms of sector, track and cylinder. Disks are viewed logically as a collection of blocks, numbered from 0 to the count of blocks on the disk. Each model has an algorithm for converting from physical to logical block number. In latest technology disks, this is implemented in hardware. Previously it was done by the device driver. Viewed in terms of logical blocks, all disks look the same except for total size.

Virtual block number refers to the relative position of a block within a file, starting from 1 to the file's block count. Mapping of file virtual blocks to disk logical blocks is done from information contained in the file's header, described below.

A disk's index file, named INDEXF.SYS;1, is the file through which all other files are located. The header of every file on a disk, including the index file itself, is a block in INDEXF.SYS. A file's file number, plus an offset determined when the disk is initialized, is the virtual block number of that file's header in the index file. The index file is created when the disk is initialized and always has file ID = (1,1,0), which means file number 1, sequence number 1, relative volume 0.

Occasionally a file will need more than 512 bytes to contain all its header information. In this case, one or more extension headers will exist in the index file to hold the header information.

The information contained in the file header is summarized below.

Header area
- Offsets to other areas
- File ID
- Extension header pointer, if any

Identification area
- File name, type, version
- Creation, revision, expiration, backup dates

RMS record area
- Record type, size and attributes
- File organization
- Location of end of data

Map area
- Retrieval pointers (LBN and size)

Access control area
- Access control list (ACL) in binary

Each retrieval pointer in the map area contains the logical block number and block count of a contiguous set of blocks belonging to that file. The file system uses the retrieval pointers to map file virtual block numbers to disk logical blocks.

To keep track of whether a disk block is allocated to a file or available for allocation, a bit map of all the disk blocks is maintained. Each bit represents one cluster of blocks. The cluster is the basic unit of allocation on a disk, cluster sizes of 1, 2 and 3 blocks are typically used. The file BITMAP.SYS holds the allocation bit map, this file is also created when the disk is initialized and has ID = (2,2,0).

Directories are sequential files with variable length records, with format illustrated in figure 4. Directories point to other directories and files via their file IDs.

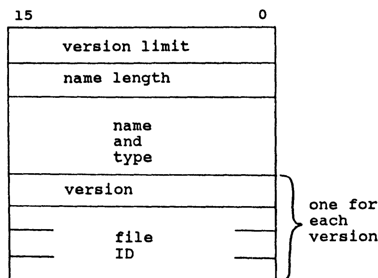



Figure 4) Directory Record Structure

The file system always maintains directories in alphabetical order so that optimized search algorithms may be used. The version limit is the maximum number of versions of a file the file system will allow to exist. It will delete the oldest version to keep the limit from being exceeded.

The master file directory is the top level directory of a volume. It is created when the disk is initialized and has file ID = (4,4,0).

The bad blocks on a disk are isolated and recorded by allocating them to a file named BADBLK.SYS. This file is also created when the disk is initialized and has file ID = (3,3,0). In latest technology disks, the bad block function has been moved to hardware, but BADBLK.SYS is still created and blocks can be allocated to it.

## COMMON ACP OPERATIONS

This section describes 12 common operations that applications typically will perform via an ACP.

- Look up file in directory
- Look up filename with wildcards
- Look up file by file ID
- Look up file and access (open)
- Create directory entry and file, open
- Look up file, create if not present, open
- Extend file
- Enter synonym in directory
- Read/write file attributes
- Read/write file data
- Deaccess (close) file with truncation
- Delete file and directory entry

Readers may refer to reference 1 for operations not described here.

### Look Up File In Directory

To look up a file in a directory, the IO$_ACCESS function (without modifier) is used. The directory ID is stored in FIB$W_DID and the filename is passed via the P2 parameter. It must have the form name.type;ver or name.type.ver. Trailing blanks, if present, should be removed because the filename parser, after it finds the semicolon, returns an error if more than six characters remain in the parameter.

The semicolon or period version delimeter must be present, but the version number can be defaulted. Zero or no version means the latest.

The filename cannot contain a directory specification. If the file ID of the directory is not known, it must be requested via a call of this type. If there is a long directory pathname, successive "look up file in directory" calls must be made, starting with the master file directory, ID = (4,4,0). This is illustrated in the Ada example at the end of this paper.

If all that is known about the file is a partial directory path and a logical name (which contains the rest of the path), the "translate logical name" system service must be used to obtain the full pathname.

The ACP returns the file ID of the target file in FIB$W_FID and the filename and length in P4, P3. The actual version number will be returned if zero or no version was input.

Look Up Filename With Wildcards

If a filename with the wildcard characters * and % is presented to the ACP, it will return all the filenames that match the specification. The IO$_ACCESS function is used, the directory ID is stored in FIB$W_DID and the filename with wildcards is passed in P2. The wildcard context field FIB$L_WCC must be set to zero and FIB$M_WILD in the name control field FIB$W_NMCTL must be set.

QIOs are repetitively issued, with each one another name and length matching the specification is returned in P4, P3. When a QIO status other than SS$_NORMAL is returned, the name in P4 is not valid and no more QIOs should be issued. Ignoring the bad status will cause the entire list to be returned again.

To cause all names, types and/or versions to match the given specification, one or more of the flags FIB$M_ALLNAM, FIB$M_ALLTYP, FIB$M_ALLVER in FIB$W_NMCTL can be set.

Look Up File By File ID

To look up a file in a directory when its ID is known but its name is not, use IO$_ACCESS function, enter the directory ID in FIB$W_DID, the target file ID in FIB$W_FID and set FIB$M_FINDFID in FIB$W_NMCTL. The target filename and length will be returned in P4, P3.

If multiple synonyms exist in the directory (entries with different names pointing at the same file) only the first will be found. They can be deleted with the IO$_DELETE function, however. Lookup by file ID is slower than by name because it involves a linear search through the directory while lookup by name takes advantage of the alphabetization.

Look Up File and Open

To open a file after locating it in a directory, add the IO$M_ACCESS modifier to the IO$_ACCESS function. Enter the directory ID in FIB$W_DID and the filename in P2. Set various flags in the access control field FIB$L_ACCTL to specify desired access.

Set FIB$M_WRITE if data is to be written to the file, clear it to read. Set FIB$M_NOWRITE to deny access to other processes wanting to write to the file. Set FIB$M_NOREAD to deny access to other readers. FIB$M_NOTRUNC will prevent others from truncating the file. FIB$M_DLOCK will cause the file to be locked if the program exits without closing the file or closes it improperly. Programs in higher modes than "user" can set FIB$M_EXECUTE to use the "execute" protection bit rather than the "read" bit to determine access.

FIB$W_SIZE is used to specify the size of the buffer used to hold retrieval pointers in memory. The larger the buffer, the fewer times the ACP will need to reread the file header to fetch pointers not in the buffer. A value of zero requests the volume default, -1 makes the buffer large enough to hold all the pointers (subject to the BYTELIM quota).

Create Directory Entry and File, Open

To create a new directory entry and file and open the file, the IO$_CREATE + IO$M_CREATE + IO$M_ACCESS function is used. The directory ID and filename are entered in FIB$W_DID and P2, respectively.

Flags in the name control field FIB$W_NMCTL control version handling. If FIB$M_NEWVER is set, the version number is incremented if the specified version already exists. Setting FIB$M_SUPERCEDE causes the old file to be deleted if the specified vesion exists. The ACP sets FIB$M_HIGHVER or FIB$M_LOWVER if a higher or lower numbered version, respectively, already exists.

A version limit for the directory entry (max number of versions allowed to exist at a time) should be entered in FIB$W_VERLIMIT. Access control information (see above) should be provided in FIB$W_ACCTL.

The ACP returns the filename (possibly with a different version number) and length in P4, P3, and the file ID in FIB$W_FID.

To create a scratch file, use the IO$M_DELETE modifier in addition to the others. Thus the function is IO$_CREATE + IO$M_CREATE + IO$M_ACCESS + IO$M_DELETE. The file will be marked for deletion when created and will actually be deleted when deaccessed or if the program exits without doing so.

Look Up File, Create If Not Present, Open

To create a file only if one with the same name, type and version does not already exist, the IO$_ACCESS + IO$M_ACCESS + IO$M_CREATE function is used. Note the difference from the previous function code.

The filename and directory ID are entered in P2 and FIB$W_DID, respectively. Version and access control information are provided in FIB$W_NMCTL and FIB$L_ACCTL, as above.

The filename and length are returned in P4, P3, and the file ID is returned in FIB$W_FID.

Extend File

In the previous paragraphs, when a file was created, only the file header and no space for data was actually allocated. To allocate data space, the extend function must be used, either when the file is created (with the IO$_CREATE function) or later (with the IO$_MODIFY function). In either case extension is enabled by setting FIB$M_EXTEND in the extend control field FIB$W_EXCTL.

Extend control options are enabled by flags in FIB$W_EXCTL. If FIB$M_NOHDREXT is set, the ACP will not create extension file headers, but will return an error if the operation requires an extension header.

FIB$M_ALCON forces contiguous allocation, with an error return if that is not possible. FIB$M_ALCONB

forces as close to contiguous allocation as possible (fewest number of pieces). Both FIB$M_ALCON and FIB$M_ALCONB being set causes the largest contiguous region that is not larger than the requested number of blocks to be allocated.

If FIB$M_FILCON is set, the file is marked contiguous. This can only be done on an initial contiguous allocation. The number of blocks to be allocated is specified in FIB$L_EXSZ. The ACP returns the number of blocks actually allocated in FIB$L_EXVBN. This may be larger than the number requested due to cluster roundup or smaller if both FIB$M_ALCON and FIB$M_ALCONB are set.

Options for specifying the location of the blocks to be allocated are available. Setting FIB$M_EXACT in FIB$B_ALOPTS forces allocation to the exact logical block specified, with an error return if not possible. FIB$M_ONCYL forces the entire allocation to be within one cylinder.

It is possible to specify allocation be as close as possible to a specified block of a related file. The file ID of that file is specified in FIB$W_LOC_FID and the block number in FIB$L_LOC_ADDR. See reference 1 for more details.

## Synonym Directory Entry

Multiple directory entries for the same file are called synonyms. To create a synonym for a file that already has at least one directory entry, the IO$_CREATE function is used with no modifiers.

The directory ID and target file ID are entered in FIB$W_DID and FIB$W_FID, respectively. The filename is provided in P2 and version control information is passed in FIB$W_NMCTL. The ACP returns the file name and length in P4, P3.

## Read/Write Attributes

Attributes are the information contained in the file header and certain information on open files maintained in internal buffers by the ACP. Some of the information that can be accessed and updated is as follows.

File name/type/version/revision count

Creation/revision/expiration/backup dates

UIC, protection masks, file characteristics

Record attributes area
- File organization, record type and size
- Highest allocated and end-of-file VBN
- Bucket size, VFC control size

Access control list (ACL)
- Add/find/modify/delete entries

Statistics block
- Starting LBN and size
- Access and lock counts
- Writer and truncate lock counts
- Number of reads and writes

Attributes are read or written if an attribute control list is present at P5. The attribute control list is a sequence of entries with format shown in figure 5. A longword of 0 terminates the list.

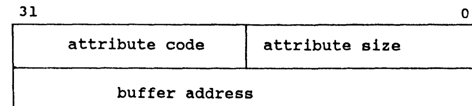



Figure 5) Attribute Control List Entry

The code specifies the attribute to be referenced. The size and address specify a buffer where the information is to be stored or originate. Any size from 1 byte up to a maximum defined for the particular attribute may be specified.

If the major function is IO$_ACCESS, the attribute information is read from the file header. If it is IO$_CREATE, IO$_MODIFY or IO$_DEACCESS, the information is written.

## Read/Write Data

To read or write a file's data, the file is first opened with a IO$_CREATE + IO$M_ACCESS or IO$_ACCESS + IO$M_ACCESS function. QIOs are issued on the open channel using the function codes IO$_READVBLK and IO$_WRITEVBLK. The function/device specific parameters are those for disk I/O drivers, as follows.

P1 - address of data buffer
P2 - number of bytes to transfer
P3 - virtual block number in file

The byte count does not need to be a multiple of 512, but all transfers begin at the first byte of a data block, so record sizes other than a multiple of 512 are inconvenient. If the file is to be sequentially accessed, the application is responsible for incrementing the VBN.

The only involvement of the ACP is to convert the VBN to an LBN for the disk, then the QIO request is passed on to the disk driver. If the VBN does not map, for example if it is greater than the number of blocks allocated to the file, an end-of-file error indication is returned. This can occur either reading or writing.

## Deaccess With Truncation

Deaccess means to close the file, truncate means to return allocated but unused space at the end of the file to the disk's free pool. The IO$_DEACCESS function is used. FIB$W_FID must contain the correct file ID of the target file or 0 (even though the ACP already has this information).

Truncation is enabled by setting FIB$M_TRUNC in FIB$W_EXCTL. The truncated blocks go to the volume's free pool unless FIB$M_MARKBAD in FIB$W_EXCTL is set, then they are allocated to the bad block file (SYSPRV is required). The ACP does have logic to perform further testing on blocks on which I/O errors occurred and to move those with hard errors to the bad block file.

FIB$L_EXVBN is used to pass the VBN of the first block to truncate to the ACP. It returns the VBN of the actual first block in the same location. This may be different due to cluster roundup. If truncation to the bad block file is specified, the truncation VBN is rounded downward.

The actual number of blocks truncated is returned in FIB$L_EXSZ. This field must be 0 prior to issuing the QIO.

### Delete File and Directory Entry

To delete a file and its directory entry, the IO$_DELETE + IO$M_DELETE function is used. The directory ID and target file ID are entered in FIB$W_DID and FIB$W_FID, respectively. The filename must be provided in P2 for directory entry deletion.

To delete the directory entry only (because there are other synonyms) use the IO$_DELETE function without the IO$M_DELETE modifier.

### Things To Remember

Allocation is separate from data transfer. The ACP will not allocate more space because the existing space is full. The ACP does not keep track of where data, if any, has been read or written. It returns an end-of-file indication any time a block specified by an IO$_READVBLK or IO$_WRITEVBLK cannot be mapped to an LBN.

When coding in Fortran, descriptors for P1 and P2 may need to be built with application code. Watch trailing blanks in the input filename in P2.

When extending a file, more blocks than specified may be allocated due to cluster roundup. When truncating, cluster roundup may cause less blocks to be truncated.

The ACP does not keep track of the location of end-of-data in a file nor does it update the end-of-file pointers in the RMS record area. The application must do this using the read/write attributes function.

If the directory ID field FIB$W_DID is nonzero, a directory lookup for the file is performed even if the operation doesn't require it. This can cause unnecessary disk accesses. FIB$W_DID should be set to zero unless the operation requires a directory lookup.

Normally, a QIO status other than SS$_NORMAL indicates an error and the ACP performs no operation other than to indicate the error. When SS$_BADPARAM is returned, the parts of the requested operation not involving the bad parameter may be completed anyway. This was observed with more than one major function.

Access control list operations performed by the read/write attributes function return binary data and are thus inconvenient to work with. They are intended for use in conjunction with the ACL editor and the ACL translation features of DCL. Applications could use them to do things like copy the entire ACL of one file to another, however.

Pay attention to fields in the File Information Block that must be 0 when a QIO to the ACP is issued. Some are not obvious, like FIB$L_EXSZ when truncating.

### FORTRAN EXAMPLE

The following example illustrates file I/O via the ACP from a Fortran program. Subroutine WRITE_TEMP_FILE creates a temporary file, writes data to it, then closes the file. READ_TEMP_FILE then opens the file created by WRITE_TEMP_FILE, reads the data and closes and deletes the file.

Each call to WRITE_TEMP_FILE passes a 512 byte data block. A flag is set on the last call. The routine creates the file with an initial allocation, then when the file fills an extension allocation adds more space. When the routine closes the file, it writes the end-of-file position in the file header.

READ_TEMP_FILE returns a data block each time it is called. When it reads the last block from the file (which it knows by examining the end-of-file position in the file header) it deletes the file and returns a flag to indicate data end.

```
C
C  WRITE A TEMPORARY FILE OF 512 BYTE FIXED LENGTH RECORDS.
C  FILE IS GIVEN AN ARBITRARY NAME AND NO DIRECTORY ENTRY IS CREATED.
C  CHANNEL NUMBER IS SAVED FOR USE BY READ_TEMP_FILE.
C
C  INPUTS:
C       DATA - 512 BYTE INPUT BUFFER
C       LAST - FLAG TO INDICATE LAST BLOCK OF DATA
C
        SUBROUTINE WRITE_TEMP_FILE (DATA, LAST)
        IMPLICIT NONE
C
        INTEGER*4 DATA(128)               ! INPUT BUFFER
        LOGICAL LAST                      ! .TRUE. ON LAST BLOCK
C
        INCLUDE '($SYSSRVNAM)'
        INCLUDE '($IODEF)'
        INCLUDE '($SSDEF)'
        INCLUDE '($FIBDEF)'
        INCLUDE '($FIDDEF)'
        INCLUDE '($ATRDEF)'
C
        LOGICAL FIRST_CALL/.TRUE./
        CHARACTER*12 DEVICE_NAME/'SYS$DISK:'/
        INTEGER CHANNEL, RETURN_CODE      ! QIO PARAMETERS
        INTEGER ACP_FUNC, QIO_FUNC        ! QIO FUNCTION CODES
        INTEGER*2 STATUS_CODE(4)          ! IO STATUS CODE FROM QIO
C
        RECORD /FIBDEF/ FIB               ! FILE INFORMATION BLOCK
        RECORD /ATRDEF/ ATR(2)            ! ATTRIBUTE LIST
C
        STRUCTURE /DESCR/
          INTEGER*2 COUNT, %FILL
          INTEGER*4 ADDR
        END STRUCTURE
C
        CHARACTER*80 FILENAME_IN          ! NAME OF TEMP FILE
        RECORD /DESCR/ NAME_IN_DESCR
        INTEGER BLOCK_NUMBER              ! NEXT BLOCK TO WRITE IN
        INTEGER BLOCKS_ALLOC              ! BLOCKS ALLOCATED SO FAR
C
        COMMON/WRITE_TEMP/CHANNEL, FIB, FILENAME_IN, NAME_IN_DESCR
        1       , BLOCK_NUMBER, BLOCKS_ALLOC
C
        RECORD /DESCR/ FIB_DESCR
        INTEGER*4 BUFFER                  ! BUFFER ADDRESS
        INTEGER*4 VIRTADDR                ! VIRTUAL BLOCK NUMBER IN FILE
        INTEGER*2 FAT(16)                 ! FILE ATTRIBUTES AREA
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
        IF (FIRST_CALL) THEN
          FIRST_CALL = .FALSE.            ! SUBSEQ CALLS WON'T GO THRU HERE
C
C-----------------------------------------------------------------------
C  ASSIGN CHANNEL
C-----------------------------------------------------------------------
C
          RETURN_CODE = SYS$ASSIGN (DEVICE_NAME, CHANNEL,,)
C
          IF (RETURN_CODE .NE. SS$_NORMAL) THEN
            WRITE (5, 910) RETURN_CODE
            CALL EXIT
          ENDIF
C
          FIB.FIB$L_ACCTL = FIB$M_WRITE ! ACCESS FOR WRITING
          FIB.FIB$W_EXCTL = FIB$M_EXTEND! EXTEND FILE ON CREATION
          FIB.FIB$L_EXSZ = 10           ! NUMBER BLOCKS TO ALLOC
          ACP_FUNC = IO$_CREATE + IO$M_CREATE + IO$M_ACCESS
                                        ! CREATE AND OPEN FILE
          FILENAME_IN = 'INTERMEDIATE.DATA;'
          NAME_IN_DESCR.COUNT = 18      ! NUMBER CHAR IN FILENAME
          NAME_IN_DESCR.ADDR = %LOC(FILENAME_IN)
                                        ! ADDRESS OF DATA
          ATR(1).ATR$W_SIZE = ATR$S_RECATTR ! DESCRIPTOR FOR RECORD ATTRIBUTES
          ATR(1).ATR$W_TYPE = ATR$C_RECATTR !  OF FILE
          ATR(1).ATR$L_ADDR = %LOC(FAT)
C
          FIB_DESCR.COUNT = 64          ! FULL FIB
          FIB_DESCR.ADDR = %LOC(FIB)
C
```

```
C
C-----------------------------------------------------------------------
C  CREATE AND OPEN FILE, NO DIRECTORY ENTRY
C-----------------------------------------------------------------------
C
          RETURN_CODE = SYS$QIOW (, %VAL(CHANNEL), %VAL(ACP_FUNC),
        2                         STATUS_CODE,,, FIB_DESCR,
        3                         NAME_IN_DESCR,,, ATR,)
C
          IF (RETURN_CODE .NE. SS$_NORMAL) THEN
            WRITE (5, 920) RETURN_CODE
            CALL EXIT
          ELSE IF (STATUS_CODE(1) .NE. SS$_NORMAL) THEN
            WRITE (5, 930) STATUS_CODE
            CALL EXIT
          ENDIF
C
          BLOCKS_ALLOC = FIB.FIB$L_EXSZ ! ACTUAL NUMBER OF BLOCKS ALLOC
          BLOCK_NUMBER = 1             ! FIRST BLOCK TO WRITE
C
        ELSE                           ! NOT FIRST CALL
          IF (BLOCK_NUMBER .GT. BLOCKS_ALLOC) THEN
                                       ! MUST ALLOCATE MORE BLOCKS
            FIB.FIB$L_EXVBN = 0        ! MUST INITIALLY BE 0
            ACP_FUNC = IO$_MODIFY
            FIB.FIB$L_EXSZ = 10        ! NUMBER BLOCKS TO ADD TO FILE
C
C-----------------------------------------------------------------------
C  ALLOCATE ADDITIONAL BLOCKS FOR FILE
C-----------------------------------------------------------------------
C
            RETURN_CODE = SYS$QIOW (, %VAL(CHANNEL), %VAL(ACP_FUNC),
        2                         STATUS_CODE,,, FIB_DESCR,,,,,)
C
            IF (RETURN_CODE .NE. SS$_NORMAL) THEN
              WRITE (5, 920) RETURN_CODE
              CALL EXIT
            ELSE IF (STATUS_CODE(1) .NE. SS$_NORMAL) THEN
              WRITE (5, 930) STATUS_CODE
              CALL EXIT
            ENDIF
C
            BLOCKS_ALLOC = BLOCKS_ALLOC + ! ACTUAL NUMBER OF BLOCKS ALLOC
        1        FIB.FIB$L_EXSZ
C
          ENDIF
        ENDIF
C
        QIO_FUNC = IO$_WRITEVBLK        ! CODE FOR WRITE VIRTUAL BLOCK
        VIRTADDR = BLOCK_NUMBER ! BLOCK IN FILE TO WRITE
C
C-----------------------------------------------------------------------
C  WRITE NEXT BLOCK TO FILE
C-----------------------------------------------------------------------
C
        RETURN_CODE = SYS$QIOW (, %VAL(CHANNEL), %VAL(QIO_FUNC),
        2                         STATUS_CODE,,, DATA,
        3                         %VAL(512), %VAL(VIRTADDR),,,)
C
        IF (RETURN_CODE .NE. SS$_NORMAL) THEN
          WRITE (5, 922) RETURN_CODE
          CALL EXIT
        ELSE IF (STATUS_CODE(1) .NE. SS$_NORMAL) THEN
          WRITE (5, 933) STATUS_CODE
          CALL EXIT
        ENDIF
C
        BLOCK_NUMBER = BLOCK_NUMBER + 1
C
        IF (LAST) THEN                  ! IF LAST BLOCK TO BE WRITTEN
                                        !  CLOSE FILE WITH TRUNCATION
          ACP_FUNC = IO$_DEACCESS       ! CLOSE FILE CODE
          FIB.FIB$W_EXCTL = FIB$M_TRUNC ! ENABLE TRUNCATION
          FIB.FIB$L_EXVBN = BLOCK_NUMBER! FIRST BLOCK TO TRUNCATE
          FIB.FIB$L_EXSZ = 0            ! MUST BE 0 OR FILE SYS COMPLAINS
C
          FAT(1) = 1                    ! FAT$B_RTYPE: SEQUENTIAL
          FAT(2) = 512                  ! FAT$W_RSIZE: RECORD SIZE
          FAT(4) = BLOCK_NUMBER         ! FAT$L_HIBLK (16 LSB)
          FAT(6) = BLOCK_NUMBER         ! FAT$L_EFBLK (16 LSB) (END OF FILE)
          FAT(9) = 512                  ! FAT$W_MAXREC
C
C-----------------------------------------------------------------------
C  CLOSE AND TRUNCATE FILE
C-----------------------------------------------------------------------
C
          RETURN_CODE = SYS$QIOW (, %VAL(CHANNEL), %VAL(ACP_FUNC),
        2                         STATUS_CODE,,, FIB_DESCR,,,, ATR,)
C
          IF (RETURN_CODE .NE. SS$_NORMAL) THEN
            WRITE (5, 920) RETURN_CODE
            CALL EXIT
          ELSE IF (STATUS_CODE(1) .NE. SS$_NORMAL) THEN
            WRITE (5, 930) STATUS_CODE
            CALL EXIT
          ENDIF
        ENDIF
C
        RETURN
C
  910   FORMAT (' ASSIGN RETURN CODE = ', Z8.8)
  920   FORMAT (' ACP QIOW RETURN CODE = ', Z8.8)
  922   FORMAT (' WRTVBLK QIOW RETURN CODE = ', Z8.8)
  930   FORMAT (' ACP STATUS CODE = ', 4Z5.4)
  933   FORMAT (' I/O STATUS CODE = ', 4Z5.4)
        END




C
C  READ A TEMPORARY FILE OF 512 BYTE FIXED LENGTH RECORDS, THEN DELETE IT.
C  WRITE_TEMP_FILE HAS ALLOCATED CHANNEL AND SET UP FIB.
C
C  OUTPUTS:
C       DATA - 512 BYTE OUTPUT BUFFER
C       LAST - FLAG TO INDICATE LAST BLOCK OF DATA
C
        SUBROUTINE READ_TEMP_FILE (DATA, LAST)
        IMPLICIT NONE
C
        INTEGER*4 DATA(128)
        LOGICAL LAST                    ! .TRUE. ON LAST BLOCK
C
        INCLUDE '($SYSSRVNAM)'
        INCLUDE '($IODEF)'
        INCLUDE '($SSDEF)'
        INCLUDE '($FIBDEF)'
        INCLUDE '($FIDDEF)'
        INCLUDE '($ATRDEF)'
```

```
C
        LOGICAL FIRST_CALL/.TRUE./
        INTEGER CHANNEL, RETURN_CODE
        INTEGER ACP_FUNC, QIO_FUNC     ! QIO FUNCTION CODES
        INTEGER*2 STATUS_CODE(4)       ! IO STATUS
C
        RECORD /FIBDEF/ FIB            ! FILE INFORMATION BLOCK
        RECORD /ATRDEF/ ATR(2)         ! ATTRIBUTE LIST
C
        STRUCTURE /DESCR/
          INTEGER*2 COUNT, %FILL
          INTEGER*4 ADDR
        END STRUCTURE
C
        CHARACTER*80 FILENAME_IN       ! NOT USED HERE
        RECORD /DESCR/ NAME_IN_DESCR   ! NOT USED HERE
        INTEGER BLOCK_NUMBER           ! NEXT BLOCK TO WRITE IN
        INTEGER BLOCKS_ALLOC           ! BLOCKS ALLOCATED SO FAR (NOT USED)
C
        COMMON/WRITE_TEMP/CHANNEL, FIB, FILENAME_IN, NAME_IN_DESCR
        1       , BLOCK_NUMBER, BLOCKS_ALLOC
C
        RECORD /DESCR/ FIB_DESCR
        INTEGER*4 BUFFER               ! BUFFER ADDRESS
        INTEGER*4 VIRTADDR             ! VIRTUAL BLOCK NUMBER IN FILE
        INTEGER*2 FAT(16)              ! FILE ATTRIBUTES AREA
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
        IF (FIRST_CALL) THEN
          FIRST_CALL = .FALSE.         ! SUBSEQ CALLS WON'T GO THRU HERE
C
          FIB.FIB$L_ACCTL = 0          ! FILE SYSTEM
          FIB.FIB$W_EXCTL = 0          !  REQUIRES THESE
          FIB.FIB$L_EXSZ = 0           !   ALL
          FIB.FIB$W_NMCTL = 0          !    BE
          FIB.FIB$L_EXVBN = 0          !     ZERO
          ACP_FUNC = IO$_ACCESS + IO$M_ACCESS
                                       ! ACCESS AND OPEN FILE
          ATR(1).ATR$W_SIZE = ATR$S_RECATTR ! DESCRIPTOR FOR
          ATR(1).ATR$W_TYPE = ATR$C_RECATTR !  RECORD ATTRIBUTES
          ATR(1).ATR$L_ADDR = %LOC(FAT)  !   OF FILE
C
          FIB_DESCR.COUNT = 64         ! FULL FIB
          FIB_DESCR.ADDR = %LOC(FIB)
C
C-----------------------------------------------------------------------
C  OPEN FILE FOR READING
C-----------------------------------------------------------------------
C
          RETURN_CODE = SYS$QIOW (, %VAL(CHANNEL), %VAL(ACP_FUNC),
        2                         STATUS_CODE,,, FIB_DESCR,,,, ATR,)
C
          IF (RETURN_CODE .NE. SS$_NORMAL) THEN
            WRITE (5, 920) RETURN_CODE
            CALL EXIT
          ELSE IF (STATUS_CODE(1) .NE. SS$_NORMAL) THEN
            WRITE (5, 930) STATUS_CODE
            CALL EXIT
          ENDIF
C
          BLOCK_NUMBER = 1             ! FIRST BLOCK TO WRITE
        ENDIF
C
        QIO_FUNC = IO$_READVBLK ! CODE FOR READ RITE VIRTUAL BLOCK
        VIRTADDR = BLOCK_NUMBER ! BLOCK IN FILE TO READ
C
C-----------------------------------------------------------------------
C  READ NEXT DATA BLOCK
C-----------------------------------------------------------------------
C
        RETURN_CODE = SYS$QIOW (, %VAL(CHANNEL), %VAL(QIO_FUNC),
        2                         STATUS_CODE,,, DATA,
        3                         %VAL(512), %VAL(VIRTADDR),,,)
C
        IF (RETURN_CODE .NE. SS$_NORMAL) THEN
          WRITE (5, 922) RETURN_CODE
          CALL EXIT
        ELSE IF (STATUS_CODE(1) .NE. SS$_NORMAL) THEN
          WRITE (5, 933) STATUS_CODE
          CALL EXIT
        ENDIF
C
        BLOCK_NUMBER = BLOCK_NUMBER + 1
        IF (BLOCK_NUMBER .GE. FAT(6)) THEN ! FAT(6) IS 16 LSB OF END-OF-FILE
                                       ! BLOCK, FAT$L_EFBLK
          ACP_FUNC = IO$_DELETE +      ! CLOSE AND DELETE FILE CODE
        1       IO$M_DELETE
C
C-----------------------------------------------------------------------
C  CLOSE AND DELETE FILE
C-----------------------------------------------------------------------
C
          RETURN_CODE = SYS$QIOW (, %VAL(CHANNEL), %VAL(ACP_FUNC),
        2                         STATUS_CODE,,, FIB_DESCR,,,,,)
C
          IF (RETURN_CODE .NE. SS$_NORMAL) THEN
            WRITE (5, 920) RETURN_CODE
            CALL EXIT
          ELSE IF (STATUS_CODE(1) .NE. SS$_NORMAL) THEN
            WRITE (5, 930) STATUS_CODE
            CALL EXIT
          ENDIF
C
          LAST = .TRUE.                ! INDIC END OF DATA
        ELSE                           ! MORE DATA TO READ
          LAST = .FALSE.               ! INDICATE THAT FACT
        ENDIF
C
        RETURN
C
  920   FORMAT (' ACP QIOW RETURN CODE = ', Z8.8)
  922   FORMAT (' WRTVBLK QIOW RETURN CODE = ', Z8.8)
  930   FORMAT (' ACP STATUS CODE = ', 4Z5.4)
  933   FORMAT (' I/O STATUS CODE = ', 4Z5.4)
        END
```

## CALLING SYSTEM SERVICES FROM ADA

Three packages are provided with VAX Ada to facilitate interfacing with VMS system services and VAX-unique features. Package STARLET contains control block structure definitions, I/O and system service status code definitions and system service interface specifications. Package SYSTEM contains VAX-specific data type definitions. Package CONDITION_HANDLING contains several functions for conveniently testing system service status codes.

The control block structure definitions in STARLET are in the form of Ada record type definitions and constant definitions (for initial values). The code definitions are Ada constants.

The system service interface specifications consist of three parts. For each system service, there is one or more Ada procedure specifications which define the parameters, there is a pragma INTERFACE which declares the procedure to be non-Ada and one or more pragma IMPORT_VALUED_PROCEDURE. The purpose of IMPORT_VALUED_PROCEDURE is to declare the external name of the system service, e.g. SYS$QIOW for QIOW and to declare the type and passing mechanism of each parameter.

Multiple procedure specifications and pragma IMPORT_VALUED_PROCEDURE are necessary to overload the system service name when parameters of different types are permitted.

## ADA EXAMPLE

The procedures GET_FIRST_FILE and GET_NEXT_FILE in package GET_FILE_NAMES accept a VMS file specification with wildcards and return a list of files that satisfy the specification.

GET_FIRST_FILE parses the file spec into device name, directory path and file name. It assigns the device to a channel, then starting with the master file directory, determines the file ID of each directory in the path. Finally, it does a directory lookup on the filename with wildcards and returns the first file in the list.

GET_NEXT_FILE returns an additional file each time it is called, indicating the end of the list with a flag that goes false.

```
--
-- ABSTRACT:
--      These procedures parse a VMS filespec that includes wildcards in
--      the filename, type and version and return a list of the filenames
--      satisfying the spec.  The following rules apply.
--
--      Network node names not recognized.  Device name must be a physical
--      device or a logical name that translates to a physical device,
--      cannot include part of a directory tree.
--
--      Full directory pathname must be given, default directory cannot be
--      assumed.  No special characters like - or ... can be in pathname.
--      File spec must be syntactically correct.
--
-- AUTHOR:
--      Al Tyrrill,  9/86
----
package GET_FILE_NAMES is

    CHANNEL_ERROR: exception;               -- COULD NOT ASSIGN CHANNEL TO DEVICE
    DIRECTORY_ERROR : exception;            -- COULD NOT FIND DIRECTORY FILE
    QIOW_ERROR: exception;                  -- DIRECTIVE ERROR ISSUING QIOW

--------------------------------------------------------------------------
--  PARSE FILE SPEC AND RETURN FIRST FILE NAME ON LIST IF THERE IS ONE.
--------------------------------------------------------------------------

    procedure GET_FIRST_FILE
              (FILE_SPEC : in STRING;
              SUCCESSFUL : out BOOLEAN;
              FILE_NAME : out STRING);

--------------------------------------------------------------------------
--  RETURN NEXT FILE NAME ON LIST, UNLESS LIST EXHAUSTED.
--------------------------------------------------------------------------

    procedure GET_NEXT_FILE
              (SUCCESSFUL : out BOOLEAN;
              FILE_NAME : out STRING);

end GET_FILE_NAMES;

with STARLET;                               -- VAX/VMS INTERFACE DECLARATIONS
with SYSTEM;                                -- VAX HARDWARE TYPES
with UNCHECKED_CONVERSION;
with CONDITION_HANDLING;

package body GET_FILE_NAMES is

    function FID_TO_DID is new UNCHECKED_CONVERSION
                (SOURCE => STARLET.FIB_FID_TYPE,
                TARGET => STARLET.FIB_DID_TYPE);

    CHANNEL: STARLET.CHANNEL_TYPE;

    type DESCRIPTOR is record
            SIZE : SYSTEM.UNSIGNED_WORD;
            FILL : SYSTEM.UNSIGNED_WORD;
            ADDR : SYSTEM.ADDRESS;
    end record;

    FILE_NAME_IN : STRING (1..86 );
    NAME_IN : DESCRIPTOR :=
                (SIZE => 0, FILL => 0,
                 ADDR => FILE_NAME_IN'ADDRESS);

    FILE_INF_BLK : STARLET.FIB_TYPE;
    FIB_DESCR : DESCRIPTOR :=
                (SIZE => 64,                -- ADA WOULDN'T ALLOW SYMBOLIC DE
                FILL => 0,
                ADDR => FILE_INF_BLK'ADDRESS);

--------------------------------------------------------------------------
--  PARSE FILE SPEC AND RETURN FIRST FILE NAME OF LIST IF THERE IS ONE.
--  SUCCESSFUL INDICATES FILE NAME RETURNED.
--------------------------------------------------------------------------

    procedure GET_FIRST_FILE
              (FILE_SPEC : in STRING;
              SUCCESSFUL : out BOOLEAN;
              FILE_NAME : out STRING) is

        INDEX: NATURAL := FILE_SPEC'FIRST;

        DEVICE_NAME: STARLET.DEVICE_NAME_TYPE (1..20);
        SYS_SERV_STATUS: CONDITION_HANDLING.COND_VALUE_TYPE;
        IO_STATUS: STARLET.IOSB_TYPE;

        FILE_NAME_OUT: STRING (1..86);
        NAME_OUT: DESCRIPTOR :=
                (SIZE => FILE_NAME_OUT'LENGTH,
                FILL => 0,
                ADDR => FILE_NAME_OUT'ADDRESS);
        NAME_INDEX: NATURAL;
        NAME_OUT_LENGTH: SYSTEM.UNSIGNED_WORD;

    BEGIN

--------------------------------------------------------------------------
--  EXTRACT DEVICE NAME AND ASSIGN CHANNEL TO DEVICE.
--------------------------------------------------------------------------

        DEVICE_NAME := "                    ";
        SCAN_DEVICE_NAME: loop
            DEVICE_NAME (INDEX-FILE_SPEC'FIRST+1) :=
                FILE_SPEC (INDEX);
            INDEX := INDEX + 1;
            exit when FILE_SPEC (INDEX-1) = ':';
        end loop SCAN_DEVICE_NAME;

        STARLET.ASSIGN
            (STATUS => SYS_SERV_STATUS,
            DEVNAM => DEVICE_NAME,
            CHAN => CHANNEL);

        if not CONDITION_HANDLING.SUCCESS (SYS_SERV_STATUS) then
            raise CHANNEL_ERROR;
        end if;

        FILE_INF_BLK.DID := (DID_NUM => 4,              -- FILE
                            DID_SEQ => 4,               --  ID
                            DID_RVN => (0, 0));         --   OF MFD

--------------------------------------------------------------------------
--  PARSE DIRECTORY PATH NAME STRING.
--------------------------------------------------------------------------

        if FILE_SPEC (INDEX ) = '[' then

            SCAN_DIRECTORY_STRING:
            while FILE_SPEC (INDEX) /= ']' loop

                NAME_INDEX := 1;
                INDEX := INDEX + 1;

                SCAN_DIRECTORY_NAME:
                while FILE_SPEC (INDEX) /= '.'
                    and FILE_SPEC (INDEX) /= ']' loop

                    FILE_NAME_IN (NAME_INDEX) := FILE_SPEC (INDEX);
                    INDEX := INDEX + 1;
                    NAME_INDEX := NAME_INDEX + 1;

                end loop SCAN_DIRECTORY_NAME;

                FILE_NAME_IN (NAME_INDEX..NAME_INDEX+4) := ".DIR;";
                NAME_IN.SIZE := SYSTEM.UNSIGNED_WORD (NAME_INDEX + 4);

--------------------------------------------------------------------------
--  GET FILE ID OF NEXT DIRECTORY LEVEL.
--------------------------------------------------------------------------

                STARLET.QIOW
                    (STATUS => SYS_SERV_STATUS,
                    CHAN => CHANNEL,
                    FUNC => STARLET.IO_ACCESS,
                    IOSB => IO_STATUS,
                    P1 => SYSTEM.TO_UNSIGNED_LONGWORD (FIB_DESCR'ADDRESS),
                    P2 => SYSTEM.TO_UNSIGNED_LONGWORD (NAME_IN'ADDRESS));

                if not CONDITION_HANDLING.SUCCESS (SYS_SERV_STATUS) then
                    raise QIOW_ERROR;
                elsif not CONDITION_HANDLING.SUCCESS (IO_STATUS.STATUS) then
                    raise DIRECTORY_ERROR;
                end if;
```

```
            FILE_INF_BLK.DID := FID_TO_DID (FILE_INF_BLK.FID);

         end loop SCAN_DIRECTORY_STRING;

         INDEX := INDEX + 1;
      end if;

------------------------------------------------------------------------------
-- EXTRACT FILE SPEC AND GET FIRST FILE NAME OF LIST
------------------------------------------------------------------------------

      FILE_NAME_IN (1..FILE_SPEC'LAST-INDEX+1) :=
            FILE_SPEC (INDEX..FILE_SPEC'LAST);
      NAME_IN.SIZE := SYSTEM.UNSIGNED_WORD (FILE_SPEC'LAST-INDEX+1);

      FILE_INF_BLK.NMCTL.WILD := TRUE;
      FILE_INF_BLK.WCC := 0;

      STARLET.QIOW
            (STATUS => SYS_SERV_STATUS,
            CHAN => CHANNEL,
            FUNC => STARLET.IO_ACCESS,
            IOSB => IO_STATUS,
            P1 => SYSTEM.TO_UNSIGNED_LONGWORD (FIB_DESCR'ADDRESS),
            P2 => SYSTEM.TO_UNSIGNED_LONGWORD (NAME_IN'ADDRESS),
            P3 => SYSTEM.TO_UNSIGNED_LONGWORD (NAME_OUT_LENGTH'ADDRESS),
            P4 => SYSTEM.TO_UNSIGNED_LONGWORD (NAME_OUT'ADDRESS));

      if not CONDITION_HANDLING.SUCCESS (SYS_SERV_STATUS) then
         raise QIOW_ERROR;
      elsif not CONDITION_HANDLING.SUCCESS (IO_STATUS.STATUS) then
         FILE_NAME (1..0) := "";
         SUCCESSFUL := FALSE;
      else
         FILE_NAME (1..NATURAL(NAME_OUT_LENGTH)) :=
            FILE_NAME_OUT (1..NATURAL(NAME_OUT_LENGTH));
         SUCCESSFUL := TRUE;
      end if;

   end GET_FIRST_FILE;

------------------------------------------------------------------------------
-- GET NEXT FILE NAME OF LIST IF ONE EXISTS.  SUCCESSFUL INDICATES PRESENCE
-- OF FILE NAME.
------------------------------------------------------------------------------

   procedure GET_NEXT_FILE
            (SUCCESSFUL : out BOOLEAN;
            FILE_NAME : out STRING) is

      SYS_SERV_STATUS: CONDITION_HANDLING.COND_VALUE_TYPE;
      IO_STATUS: STARLET.IOSB_TYPE;

      FILE_NAME_OUT: STRING (1..86);
      NAME_OUT: DESCRIPTOR :=
                  (SIZE => FILE_NAME_OUT'LENGTH,
                  FILL => 0,
                  ADDR => FILE_NAME_OUT'ADDRESS);
      NAME_OUT_LENGTH: SYSTEM.UNSIGNED_WORD;

   begin

      STARLET.QIOW
            (STATUS => SYS_SERV_STATUS,
            CHAN => CHANNEL,
            FUNC => STARLET.IO_ACCESS,
            IOSB => IO_STATUS,
            P1 => SYSTEM.TO_UNSIGNED_LONGWORD (FIB_DESCR'ADDRESS),
            P2 => SYSTEM.TO_UNSIGNED_LONGWORD (NAME_IN'ADDRESS),
            P3 => SYSTEM.TO_UNSIGNED_LONGWORD (NAME_OUT_LENGTH'ADDRESS),
            P4 => SYSTEM.TO_UNSIGNED_LONGWORD (NAME_OUT'ADDRESS));

      if not CONDITION_HANDLING.SUCCESS (SYS_SERV_STATUS) then
         raise QIOW_ERROR;
      elsif not CONDITION_HANDLING.SUCCESS (IO_STATUS.STATUS) then
         FILE_NAME (1..0) := "";
         SUCCESSFUL := FALSE;
      else
         FILE_NAME (1..NATURAL(NAME_OUT_LENGTH)) :=
            FILE_NAME_OUT (1..NATURAL(NAME_OUT_LENGTH));
         SUCCESSFUL := TRUE;
      end if;

   end GET_NEXT_FILE;

end GET_FILE_NAMES;
```

## REFERENCES


1. VAX/VMS I/O User's Manual: Part I, AA-Z600C-TE, Digital Equipment Corp., April 1986.

2. VAX Fortran User's Guide, AA-D035D-TE, Digital Equipment Corp., September 1984.

3. VAX Ada Programmer's Run-Time Reference Manual, AA-EF88A-TE, Digital Equipment Corp., February 1985.

# THE OVERSEER
# AN ACTIVITY BASED RESOURCE MANAGEMENT SYSTEM FOR VAX/VMS

Steven G. Duff and Joseph W. Fiedeldey

Ergodic Systems, Inc.
3900 Birch, Suite 105
Newport Beach, California 92660
(714) 752-2972

## ABSTRACT

**The OVERSEER**, a new VMS resource management and chargeback system uses a "real-time" philosophy similar to that found in traditional mainframe systems. By incorporating its own authorization database, monitoring processes, activity logs, embedded languages, and reporting system, the Overseer operates independent of VMS Accounting. The real-time nature of the system supports not only the capture of resource and billing information but makes possible many authorization, activity monitoring, and control functions as well.

## INTRODUCTION

This paper describes some of the problems associated with using VMS Accounting for resource management, and the architecture and underlying design of a new type of resource management system, The Overseer.

## PROBLEMS WITH VMS ACCOUNTING

Relative to mainframe systems, VMS has traditionally been weak in the area of accounting for system activity. Both Digital and a number of independent vendors have attempted to fill this gap with add-on products. These products usually base themselves on VMS Accounting log data, sometimes augmented with user-written records.

One serious problem with using VMS Accounting data is that it is essentially "point-in-time" information, that is post-processed to produce activity summaries. Systems that rely on information that VMS Accounting captures are thus limited in the quantity, accuracy, and timeliness of their base data. This prevents such systems from operating "live".

Products such as Digital's SPM, which recognize the inadequacies of VMS Accounting, collect and keep their own data. These systems, however, are designed for performance management and tuning, not activity control, authorization, monitoring and chargeback.

Another major area where VMS is inadequate is activity management. VMS has a "one-dimensional" authorization mechanism - the VMS username. VMS does not address the problem of managing system <u>activities</u> independently of usernames. Most sites either give different usernames to the same person for different activities, or attempt a partial solution to the problem with complicated ACL schemes that damage performance, and a system manager's sanity. Giving one user different usernames does help track activities (one can use either part of the username as a key, or the SYSUAF ACCOUNT field to consolidate), but the scheme compounds system management problems manyfold, makes the system difficult to use (which username do you send MAIL to, for example), and opens the door to security problems.

ACLs can control access, but do little to account for activity, and provide nothing in the way of an activity record that can be used for management and chargeback.

## THE OVERSEER

The Overseer is a complete resource management system for VMS version 4. The system takes advantage of many VMS version 4 features. It operates on the full range of Digital's VAX processors from the microVAX to VAX 8800, and was developed in a VAXcluster environment.

A major design goal was to produce a system that is truely integrated, with simple data paths, few files, and rigorous, documented protocols. Figure 1 is a diagram of the basic Overseer components and their interrelationship. The sections that follow describe each of them.

```
+-------+          --          --      +-------+
|  CUP  | <-----> |Authorization| ---> |Royalty|
|Utility| <--+    |  Database   | --+  | Image |
+-------+    |     --          --   |  +-------+
             |                      v       |
+-------+    |    --        --   +-------+  |
|  LOG  | <--+-- |Transaction|   | ACNTG |  |
|Utility| <---+  |   Logs    |   |Utility|  |
+-------+     |   --        --   +-------+  |
    |         |         ^              |    |
    |    --      --     |  +---------+ |    v
    v   |    OSL   |    |  |  Print  | |    |
    |   |Procedures|    ^  |Symbiont | v    |
    |    --      --     |  +---------+ |    |
    v        |          |       |      |    |
 --    --    |      +-------+   |      |    |
|Reports/|   |      |SCANNER| <-+      |    |
|Displays|   +--->  |Process| <--------+----+
 --    --           +-------+
```

(Fig. 1)
Overseer System
Architecture

Authorization Database
The authorization database provides the basic information needed to track system activities at various levels. It was decided early on in the design to adopt a scheme whereby the system manager could administer both "Customers" and activities ("Projects"), designated project managers could administer "Subprojects" and users within their project, and users themselves could designate "Task-ids" in order that activity could be authorized, tracked and consolidated at any level.

This approach necessitated the development of an authorization database composed of interrelated indexed files that contain information about Customers, Projects, Subprojects, Users, and Royalty images. This database does not replace the normal VMS SYSUAF file, the Overseer system only comes into play once a user has signed onto the system. The database contains authorization, reporting and usage information, as well as a "charge summary matrix" for every entity in the database.

This charge summary shows, for every category of charge, the charge-to-date, current period charges, prior period charges and forecast charges. Since the Overseer system tracks charges in real-time (as they are made), this charge summary is updated in such a way that it is always up-to-date. The charge summaries provide a means to implement optional budget, forecasting, and allocation checking, as well as imediate summary display.

CUP Utility
CUP maintains the authorization database. This utility provides controlled access to the database for system mangers, project managers, and other users, and does the extensive consistency checks needed to preserve the integrity of the database. With a single CUP command the system manager can seed the authorization database from the SYSUAF file .

Command Line Interface CUP, like all of the Overseer utililties supports full command line recall, define/key, and symbol substitution. A command line user interface was chosen because this type of interface is typically the most flexible and efficient. It is also one familiar to systems managers. Command line interfaces also integrate easily into menu systems such as All-In-1.

System Documentation All Overseer documentation is written in a formatting language especially designed for the project. It allows the same documentation source to be produced in both manual and on-line help format.

ACNTG Utility
ACNTG communicates requests to the Scanner process, using a mailbox message protocol, on behalf of a user or manager. Through ACTNG the system manager controls the tracking and billing of system resources, and users specify which activities they wish to charge.

One of the main functions of ACNTG is to change session accounting. This is a complex activity. Several of the main steps in this process are outlined below.

- **Authorization Checks** - The authorization database is checked to verify that the request is allowed. Then default information (UIC, default directory, privileges etc.) for the new session is retrieved, and optionally the user is prompted for a project password.

- **ACL Setup** - Access control lists are placed on the job logical name table and log file, giving the new UIC access.

- **Process Switch** - The new UIC is placed in the JIB and the subproject in the account field of the PCB.

- **Scanner Notified** - The Scanner is instructed to start tracking resource usage for the new session.

- **DCL Restarted** - New executive mode job logical names are assigned and DCL is restarted. If specified, up to four command files are executed.

ACNTG also has the ability to send direct charge requests to the Scanner, specifying the detail items, and amounts and counts for each detail item. The Scanner records these charges directly in the transaction log. This provides a means of making extraordinary charges, refunds, or charging for resources not tracked by the Scanner directly.

The Scanner Process
The Scanner is at the heart of the Overseer system. It is a hibernating detached process which tracks system resource usage for sessions, devices, and images.

The Scanner has a number of AST-driven interval timers and events that cause it to perform its update scans. Most of these timers can be adjusted by the system manager.

For example, every few seconds, the Scanner updates its internal tables that record the state of every process it is tracking. In order that this impose no noticeable overhead on the system GETJPI is not used, instead the Scanner goes directly to the

system control blocks and extracts the needed information. Other cycles are less frequent. A device allocation scan is done once a minute to update usage counts for devices it is tracking. A charge update cycle occurs at intervals, defined by the system manager, which calls the OSL billing procedures to update the session charges. A disk billing subprocess is started automatically by the Scanner at intervals according to the disk billing requests logged through ACNTG.

Checkpoint-Restart The Scanner also declares an interrupt every few minutes that causes it to save its internal state, and commit transactions to the active log. This gives the Scanner a restart mechanism, whereby it can recover session activity, and resource and billing information after a system crash.

Session Billing The Overseer system defines a session as a main process, and all subprocesses. The Scanner tracks a session as a unit, unlike VMS, which records process and subprocess activity separately. A process is free to change to a new activity at any time, unless authorizations prevent it.

Disk Billing When the Scanner starts a disk billing process, it allocation-locks the disk and scans the index file (INDEXF.SYS) to find the UICs to be charged. Disk performance information, such as average extents per file, can be displayed as the disk is scanned. Disks can be billed individually or in bulk. Since the process uses a high speed scan of the volume index, not the quota file, it is not necessary to enable quotas to account for disk space usage.

Network & Batch Job Billing For non-interactive sessions the Overseer will optionally search for the originating session to get authorization parameters such as project, subproject, and task-ids. For network jobs this is accomplished by a network server function within the Scanner process.

Print Symbiont

To track print jobs in real-time, the Overseer provides a special print symbiont. This symbiont communicates with the Scanner at the conclusion of a print job, and (optionally) displays the charge detail and job statistics on a trailer page.

This design permits the Overseer to record information about the print job that is often important for chargeback purposes, and that is missing from VMS Accounting logs. For instance, the Overseer print symbiont records the form name and submission queue name. The OSL billing procedures can examine these values, and base their charges on them.

For sites that cannot run a special symbiont, the system has the capability of reading VMS Accounting print job records off-line, and using them as a basis for billing.

Royalty Image

An important requirement for resource accounting is the ability to track royalty program execution. The nature of such programs usually precludes however, changes to the source code. Instead, to do this, the program is linked with a protected, loadable system service shared image provided with the Overseer. This causes the program to call routines in this image when it starts and via an executive-mode image rundown vector, at program completion. These routines check the image against an image authorization database, and extract authorization data based on the image name. The Scanner is notified of the image execution, and the image begins execution. At the end, the program goes back into the privileged image, which notifies the Scanner again. The Scanner returns the computed charges to the waiting image, where they can be displayed to the user.

This scheme allows the Overseer to perform authorization checks before an image is run. For example, the image authorization record has an expiration date after which the image cannot be run. More robust checks can be performed by the OSL rate procedures, which can inspect the authorization information, and choose to disallow execution if, for example, the individual is not allowed to use the program, or usage is not allowed during certain hours of the day.

A significant side benefit of this scheme is that if a user attempts to run a private copy of the image from his own account, it will not work, since the authorization check will fail.

As with print jobs, if it is impossible to link the image with the special shared image, the ACNTG utility has the ability to read VMS Accounting image records, and use them as a basis for billing. Of course, when this method is employed, the authorization functions, as well as a considerable amount of additional resource data normally given to the Scanner, is not available.

OSL Rate Procedures

The Overseer incorporates an embedded procedural language called the Overseer Specification Language (OSL). When the Scanner needs to charge for system resources, an OSL procedure is compiled and presented with information about the authorization context, and resource usage. The OSL procedure is executed to do any calculations, and record charges in the transaction log.

OSL is also used by the LOG utility to calculate adjustments, discounts, surcharges, and make correction to existing accounting records.

OSL compilation involves a recursive decent parse of the source procedure to produce a threaded code stream. This code stream is interpreted during successive rate table evaluations. Once a table is compiled in memory, requests for that table continue to use the compiled version. The Scanner, however, can be instructed to load a new version of the table at any time. This is accomplished through the use of reference counts and deferred deallocation.

The language incorporates integer, real, string and associative arrays (tables) datatypes, with automatic type conversion. It supports the familiar IF-THEN-ELSE, DO WHILE, GOTO and GOSUB language constructs, as well as an assortment of special functions. The language is highly extensible with a simple interface for defining site coded OSL functions.

**Charge Detail** What the Overseer charges for is determined by the site manager. A major design goal was to avoid any fixed charge structures, categories or billing structures. The system manager defines what items to bill for, what they are called, their cost, their associated resource (if any) and how they are displayed and summarized. These definitions are loaded into special logical name tables, and so can be changed at any time.

**Transaction Log File**
The design of the Overseer log file format was undertaken with great care. The result is a design that minimizes the risk of corruption, is efficient to process, and above all does not unreasonably consume space. As any site who has ever enabled system-wide image accounting knows, VMS log files can consume large amounts of disk space.

The "skeletal" structure of Overseer log files is very similar to VMS Accounting's. This primarily means that the records are packetized, and contain certain header information. The internal organization of the packet data is quite different however, since a more space efficient structure was desired.

The first thing one notices about VMS Accounting records is that there is a three byte overhead for every string field. Although this design permits immediate access to any string in the packet, this is of little value, since packets are usually packed and unpacked as a unit. The Overseer stores strings sequentially in the packet, with a length byte, but no offset word. This saves two bytes for every string. This savings is considerable since typically over a dozen string values are stored in each accounting record.

Secondly, the quadword date-time standard is, for accounting purposes, excessive. After experimentation, it was discovered that masking out the most and least significant bits leaves a 32-bit value that is accurate to less than 1/2 second. See figure 2. (For this reason, these compressed time values have been named "semi-seconds".) For absolute times, the date range for semi-seconds extends from about 1973 to well past the year 2000. Delta times subject to this compression have the same resolution (about 0.4 second), with a range of 9999+ days. This allows a savings of 4 bytes for every time value stored. The conversion to and from semi-seconds requires only 4 machine instructions.

**(Absolute time)**

```
XXXXXXXX XX000000 00000000 00000000 :0
00000000 10XXXXXX XXXXXXXX XXXXXXXX :4
```

**(Delta time)**

```
XXXXXXXX XX111111 11111111 11111111 :0
11111111 11XXXXXX XXXXXXXX XXXXXXXX :4
```

(Fig. 2)
Bit masks and shift values
to convert to/from semi-seconds

In addition, most of the binary values in Overseer log files are zero-trimmed with bit-stuffed length marks. This too is a very fast and simple commpression technique.

**LOG Utility**
The LOG utility is used to manipulate the transaction logs. It has the capability to report, summarize and dump Overseer accounting logs, with extensive sort and select options. Log files can be consolidated, dispersed, and corrected or adjusted with a single command. Through the LOG utility users can run a collection of standard reports, including graphic reports. All standard reports, though fixed in format, are highly user modifiable.

**Callable LOG** Standard Overseer reports and site coded custom reporting are implemented through the callable LOG interface. Callable LOG handles all the sorting, selecting, packing and unpacking of log records, and report summarization logic. Using the interface, producing new reports involves little more than writing the line summarization and formatting logic for the report. Source code for standard reports is provided to assist in the development of custom reports.

**OML Formatting** To aid with report formatting, Overseer incorporates OML - Overseer Macro Language, which provides macro evaluation of a string under user control. Essentially, a source string is passed to this package and a series of "result" strings are passed to a designated routine which is usually a display or print routine. OML supports many primitive macros for such things as centering a string, overlaying one string on another, formatting a date or amount, conditional evaluation, etc. Like OSL, OML is extensible.

## SUMMARY

The approach taken in the design of the Overseer circumvents many of the difficulties that have traditionally plagued the implementation of VMS resource management and chargeback systems. Avoidance of VMS Accounting, and implementation of separate monitoring and authorization components permits the extension of the Overseer to realize capabilities not presently available on VMS. These capabilities will be explored further in later releases of the system.

As an example, Overseer OSL programs presently can send messages to users and/or terminals. It is possible to expand such abilities to allow, for example, OSL programs to suspend processes which exceed a certain CPU usage, or even kill them outright. Another example is an OSL procedure that could monitor the state of disks, and issue alarms when they reach some defined percentage of their capacity. The issues of resource management, chargeback, activity management, and system monitoring are intimately tied together. The Overseer is a unique product that actually makes that happen.

# Ins and Outs of VMS Shareable Images

Ted A. Marshall
Britton Lee, Inc.
Los Gatos, California

ABSTRACT

The VAX/VMS system provides a mechanism for program segment sharing known as the "shareable image." This allows a set of subroutines to be used by many different programs without having to include this actual code in each program executable file, thus saving disk space. Also, the physical main memory containing this shared code can be used by multiple processes at once, thus saving system memory. Examples of shareable images include the VMS Run Time Library provided by DEC and the extended C run time library included in my company's product. DEC provided shareable images (such as the RTL) are almost always invoked by user written programs. However, few programmers understand the operation of shareable images or know how to write one. This paper will begin with a general explanation of what is a shareable image and how a programmer uses one in his program. Then it will proceed to writing code for and building a shareable image. Special emphasis will be placed on issues not well documented by DEC, including coding shareable images in high-level languages, sharing writable data between processes, and transfer vectors.

## 1. Introduction

Many companies, including mine, need to release software products in the form of a set of subroutines to be linked with both programs provided with the package and user written programs. The way this is frequently done is to provide either a set of object files or an object library containing the routines. However, this method poses three major disadvantages:

(1) As these routines are linked into a large number of programs, a large amount of disk space is required to hold all of the executable images, each containing a copy of the routines.

(2) As several users run copies of programs using this library of routines, each will have his or her own copy of the routines in virtual memory, thus impacting system performance.

(3) Finally, if a bug is found and fixed in one of the routines or the code is just somehow improved, each and every program using that routine must be re-linked to bring in the change.

However, there is an alternative method of packaging this code library so as to overcome these disadvantages. That is the VMS shareable image.

The VMS shareable image is a mechanism by which one or more object modules of an executable program may be placed in separate image files. These separate, non-executable image files are loaded with the main image by the VMS image activator when the program is run. This provides an answer to all three of the problems previously stated:

(1) A singe shareable image can be used by many different programs, thus saving disk space for the image files.

(2) The system manager can install the shareable image to allow the main memory for it to be shared, even when included by different user written programs that are not themselves sharable.

(3) If properly written, a shareable image can be replaced with a newer version without even relinking the programs that use it.

Examples of shareable images provided by DEC as part of the standard VMS distribution include the VMS run-time library and run-time libraries for languages such as C and FORTRAN. My company provides a shareable image for a run-time library that provides access to our Intelligent Database Machines. DEC's shareable images are mostly coded in MACRO-32 and BLISS-32 and our IDM shareable image is coded in VAX C. However, any DEC supplied language processor producing VAX native mode code can be used.

This paper will explain how shareable images work and how to write and use them. Several areas that are not well documented by DEC will be covered, including writing upward compatible shareable images, using a shareable image to share read/write data between processes and coding in high level languages.

## 2. How it works

To understand how shareable images work, one must first understand another concept that I believe many VMS programmers do not. That is the program section or PSECT. A PSECT is a group of code or data that has some common attributes that is described below. Each PSECT is assigned a name. A PSECT is one of the fundamental units processed by the linker and the contents of a PSECT will always be assigned consecutive virtual memory addresses when the program is loaded into memory. When a compiler or assembler on VMS produces an object file, the code and data contained in it are divided into one or more PSECTs. When coding in MACRO-32, the programmer has complete control over the division of the PSECTs. In high level languages, the compilers generally automatically divide the module into PSECTs, generally one for the generated code (usually named something like "$CODE") and one for each global data structure (i.e. C global variable or FORTRAN *COMMON* block), named the same as the data structure.

The attributes that are of interest here are RD or NORD, WRT or NOWRT, EXE or NOEXE, CON or OVR, SHR or NOSHR, and PIC

or NOPIC. The RD/NORD, WRT/NOWRT and EXE/NOEXE control whether the code or data contained will be accessible for reading, writing and execution respectively when it is finally loaded into virtual memory for execution. CON/OVR controls how the linker will combine PSECTs with the same name from different modules. PSECTs containing code use CON meaning to concatenate the PSECTs so that each module's code will be loaded at different virtual addresses. PSECTs for global data structures use OVR to overlay the PSECTs to start at the same virtual address so that, for example, the variables in a FORTRAN *COMMON* in two modules will be the same. If the PSECT ends up in a shareable image that has been installed sharable, the SHR/NOSHR attribute controls whether a process shares the virtual memory with other processes or if each process get its own copy. Finally, the PIC/NOPIC attribute informs the linker whether the PSECT contains only position independent code. That is, code that can be placed at any virtual address without requiring any changes to the generated instructions.

When the linker combines object modules and libraries to produce a self-contained executable image file, it combines PSECTs with the same name according to the CON/OVR attribute and then concatenates PSECTs with the same RD/NORD, WRT/NOWRT, EXE/NOEXE and SHR/NOSHR attributes into a single "image section" in the executable image. When the image is run, the image activator loads each image section into a set of consecutive virtual addresses, sets the protection information on those locations, and finally starts the program. All cross module references have already been resolved at link time.

When a shareable image is linked, the same basic operations occur. However, the linker also stores some symbol table type information in the shareable image. Specifically, for each PSECT, its name, attributes and offset from the beginning of the shareable image are saved. Also, for each global symbol (i.e. routine entry) that is declared to be accessible from outside the shareable image, the name and offset are saved. These symbols are called "universal symbols".

When the executable image is linked against the shareable image, the shareable image's symbol table is examined. The universal symbols are used to resolved undefined external references. In addition, references to PSECTs (with the OVR attribute) that also exist in the shareable image are changed to references to the shareable image. This effectively overlays the PSECT, just as if the two modules had been directly linked together. The value recorded in the referencing image for the reference into the shareable image is the offset from the base of the shareable image.

Finally, when the executable image is to be run in a process, the Image Activator loads it into virtual memory as before. The Image activator then also loads each referenced shareable image into virtual memory. As it loads each one, it selects the virtual address range that the shareable image will occupy for the process and finalizes each reference to the shareable image by adding the base address to each offset recorded by the linker in the referencing image.

Because the final fixing of the references into the shareable image is deferred to program startup, the contents of the shareable image can be placed at any location within virtual memory. This allows other shareable images that have already been loaded to be different sizes then when they were originally linked. On the other hand, because the linker only puts into the referencing image the offset into the shareable image for the reference, the position of an externally referenced symbol must not change relative to the base of the shareable image.

**3. Coding a shareable image in MACRO-32**

Coding a set of routines in MACRO-32 to be included in a shareable image requires care in two areas: position independence and PSECT declarations. Because the shareable image may end up positioned at just about any virtual address in the process running the user program, the code in it must be position independent. This basically requires three things: avoiding use of the absolute addressing mode ("@#address"), replacing .LONG directives referring to addresses within the shareable image with .ADDRESS directives and including the PIC attribute on the PSECT declaration.

The .ADDRESS directive defers final resolution of the contents to the VMS Image Activator when the program is actually run. In the longword generated by .ADDRESS, the linker records the offset from the base of the shareable image. Then, when the program is loaded by the image activator and the actual base address is established, that is added to each of these values so that they now contain the full virtual address.

For PSECT declarations, executable code and read-only data that is to be shared must have the RD, NOWRT, EXE, PIC and SHR attributes. This allows the memory to be shared between processes. Read/write data must be in a PSECT with the RD, WRT, NOEXE and NOSHR attributes. This guarantees that each process will get its own copy of this section.

The following is an example of a routine coded to be included within a shareable image:

```
        .TITLE   DISPAT for XXXLIB
        .PSECT   $CODE,RD,NOWRT,EXE,SHR,CON

        .ENTRY   DISPAT,M^<R2,R3>
        ; Routine passes its two arguments to one of
        ; several routines based on value in variable
        ; TYPE.

        MOVL     L^TYPE,R2     ; Get dispatch value.
        MOVL     TABLE[R2],R3  ; Get routine address
                               ; from dispatch table.
        CALLG    AP,(R3)       ; Call routine.
        RET

        .PSECT   DISPATDATA,RD,WRT,NOEXE,NOSHR,OVR

TYPE::  .BLKL    1             ; Dispatch value written
                               ; by our caller.

TABLE:  .ADDRESS ROUTINE1      ; Dispatch table.
        .ADDRESS ROUTINE2
        .ADDRESS ROUTINE3

        .END
```

Note the PSECT declarations and the use of .ADDRESS directives in the dispatch table. Because the longwords created by the .ADDRESS directives are modified in each invocation by the Image Activator, the table is in the read/write, non-shareable PSECT.

The following example LINK command produces the shareable image:

```
$ LINK/MAP/SHAREABLE=XXXLIB DISPAT,-
    <other object files>,X.OPT/OPTIONS
```

where the file X.OPT contains the following

```
UNIVERSAL=DISPAT
UNIVERSAL=TYPE
<similar universal commands for other "external" symbols>
```

This creates the shareable image file XXXLIB.EXE. Note that there must be a UNIVERSAL command in the options file for each global symbol within the shareable image that is to be known to the referencing program.

**4. Calling a shareable image from MACRO-32**

Just as a shareable image can be coded in MACRO-32 with very little special work, it may be called from a very ordinary MACRO-32 program. The only major restriction is in the way the referencing image accesses locations in the shareable image. MACRO-32 includes two special constructs for addressing shareable images, among other uses. These are general mode addressing ("G^symbol") and the .ADDRESS directive. If symbols in the shareable image are addressed otherwise, for example with normal relative mode addressing or with the .LONG directive, the linker is forced to finalize the references to the shareable image. This is known as "basing" the image. In this case, each shareable image is assigned a specific virtual address at link time when used with this executable image. Therefore, if one of the shareable images is replaced with newer version which is larger, even if it is otherwise

upward compatible, the executable image may have to be relinked to allow the enlarged image to fit.

However, if only general mode addressing and .ADDRESS are used, the linker only fixes the offset of the symbol from the base of the shareable image. Later, when the program is run, the image activator assigns virtual addresses for the shareable images and fixes up the references. .ADDRESS references are fixed by adding the base of the shareable image to the offset recorded by the linker. General mode references are translated by the linker to indirect references into a "fixup vector" added into the executable image. Each longword of the fixup vector is equivalent to a .ADDRESS entry, one for each location referenced. These are then fixed up the same way.

The following code fragment gives an example of calling the routine in the example shareable image:

```
MOVL    #1,G^TYPE
CALLG   <args>,G^DISPAT
```

The first instruction write a value to a read/write location within the shareable image and the second calls the routine. Note that both of these references are as general mode references.

The following LINK command shows linking the program, including the reference to the shareable image:

```
$ LINK/MAP program,X.OPT/OPTION
```

where the file X.OPT contains

```
XXXLIB/SHAREABLE
```

Note that the reference to the shareable image must be within a linker options file. LINK will not accept a reference to a shareable image on the main command line.

The executable image can then be invoked as normally, using the DCL RUN command or the $CREPRC system service, for example. The image activator will load the shareable image along with the executable image. One note here is that the image activator by default loads all shareable images from the disk directory specified by the logical name SYS$SHARE, which is normally defined to be the same as SYS$LIBRARY. The easiest way to have a specific shareable image loaded from another location is to define a logical name that is the same name as the shareable image. For Example:

```
$ DEFINE XXXLIB DRA6:[TED]XXXLIB.EXE
```

### 5. Making Upward Compatable Shareable Images

One of the advantages to using shareable images is that under proper conditions, you can modify and recreate the shareable image without having to even re-link the executable images that reference it. The key to doing this is to make sure that each referenced location (universal symbol and overlayed data PSECT) is at the exact same offset from the base of the shareable image as before. This is required because the linker records only this offset for references into the shareable image. Thus if a routine entry moves 7 bytes forward because of additions to the previous routine, the call to that routine will go to the wrong location.

DEC has provided a mechanism in MACRO-32 and the linker to make it easy to have immobile code entry points. This is known as transfer vectors. This is done generally in a separate module linked to be at the base of the shareable image. To create the transfer vectors, you need a MACRO-32 module that contains the following statements for each universal symbol called by a CALL instruction (this includes most routines coded in or called by high level languages):

```
.TRANSFER   FOO
.MASK       FOO
JMP         L^FOO+2
```

where FOO is the universal's name. These statements do the following:

(1) The .TRANSFER directive instructs the linker to make FOO a universal symbol and to point all references from outside the shareable image to FOO to this location instead of the actual FOO entry point. Note that internal references (including this JMP instruction) still refer to the actual FOO entry point.

(2) The .MASK directive instructs the linker to copy the value at the actual location FOO (that is, the routine's entry mask) to the word allocated at this location.

(3) Finally, the JMP instruction transfers control to the first instruction of the actual routine FOO which is located just after the original entry mask.

Thus the above example generates the following code:

```
<label FOO for use by external references>::
        WORD    <contents of entry mask for FOO>
        JMP     <first instruction of actual FOO>
```

The transfer vectors should be in a single MACRO-32 module. Because these vectors must not move relative to the base, the order of the transfer vectors must never change; new entries must be added at the end. The entire module must be in a unique PSECT name. Commands must be added to force the linker to place the transfer vectors at the beginning of the shareable image. Without my going into details here, you can place these two commands in the link options file to do this:

```
CLUSTER=<psect-name>
COLLECT=<psect-name>,<psect-name>
```

These lines must be located before any other CLUSTER or COLLECT options.

Also, the GSMATCH option must be used when linking the shareable image to allow the image activator to bring in a different version of the shareable image. This option specifies a major and minor version of the shareable image and a matching option. For our upward compatible shareable image, we can include the following line in the link option file:

```
GSMATCH=LEQUAL,1,1000
```

This specifies that the shareable image has a major version of 1 and a minor version of 1000. When a referencing image is linked against the shareable image, these version numbers are saved in the referencing image. The LEQUAL tells the image activator to load the shareable image only if the minor version in the referencing image is less than or equal to that in the shareable image (and the major versions are the same). Thus, if the referencing image was linked with XXXLIB version 1,1000, it will load with XXXLIB version 1,1001 or 1,1002, etc. The next time we modify and relink the shareable image, the minor version should be changed to 1001. If in the future, we completely change XXXLIB so that it is no longer downward compatible, we can change the major version to 2 to force all referencing images to be relinked.

Finally, if the routine is entered by a JSB or BSB instruction, the transfer vector should instead look like this:

```
.TRANSFER   FOO
JMP         L^FOO
.BLKB       2
```

At the end of this paper is a template MACRO-32 module for creating transfer vectors.

Unfortunately, there is no easy way to set up transfer vectors for data references. For upward compatible shareable images, I would suggest that, if at all possible, the referencing image not directly access data within the shareable image. Instead, include routines within the shareable image to return the data or a pointer to it.

### 6. Sharing Read/Write Data Between Processes

One interesting application of shareable images is implementing very efficient interprocess communications. This is done by including a PSECT within the shareable image with the RD, WRT and SHR attributes. This way, multiple processes loading the shareable image will share memory for this writable section. Thus, reads and writes to those locations by one of the processes will immediately be reflected in all of the other processes. One process could write a large block of

information into this memory and then signal the other by some mechanism (for instance, mailboxes). The other process then can examine the information within its own memory. This saves the overhead of transferring all of the data through the VMS mailbox driver.

The major disadvantage of this method is that the shareable image must be installed by the INSTALL utility with the /SHARED and /WRITEABLE options. The image activator will not load a shareable image containing writable, shareable image sections unless it has been installed this way. Since installing images requires special privileges and costs some system resources (specifically global pages and sections), this may not be practical for the general programmer.

Also, you must be sure that only the variables that you intend to be shared are in shareable PSECTs. Otherwise, you will end up with a program that works erratically as one process modifies variables used by the other.

Finally, it should be noted that when a shared writable location is modified, VMS will eventually write the page containing it back to the shareable image file. Thus, using this technique will add somewhat to disk I/O overhead on the system.

### 7. High Level Languages

It is fairly easy to code and access shareable images in high level languages. The DEC provided compilers take care of most of the special requirements that MACRO-32 programmers must deal with. They produce position independent code and make general mode references to external symbols. The transfer vectors, if used, still have to be coded in MACRO-32, but this is mainly filling in a standard template and really does not require knowledge of MACRO-32.

There is, however, one problem that must be watched out for. Several of the DEC compilers by default put global data in writable shareable PSECTs. These include global variables in C and *COMMON* blocks in FORTRAN. In C, this can be overridden by including the non-portable keyword *noshare* in the variable declaration. Note that this must be included in the declaration of the variable in all modules that access it, otherwise, the linker will give a warning that PSECTs with the same name have conflicting attributes.

A general solution is the PSECT_ATTR option in the linker option file. For each C global variable or FORTRAN *COMMON* block, include the following line in the options file:

PSECT_ATTR=<name>,NOSHR

where <name> is the C variable or FORTRAN *COMMON* block name. This will override the SHR or NOSHR attributes generated by the compiler and set those PSECTs to be non-shareable.

Also, be careful of non-DEC provided compilers. I have seen an older version of one third-party C compiler that did not always generate general mode references when needed.

### 8. Acknowledgements

I want to thank my colleagues Jon Forrest and Jeff Gorris for their assistance in preparing this paper.

### 9. References

(1) *VAX/VMS Linker Reference Manual*, Digital Equipment Corporation, order number: AA-Z420A-TE, September, 1984.

(2) *VAX/VMS Install Utility Reference Manual*, Digital Equipment Corporation, order number: AA-Z417A-TE, September, 1984.

(3) *VAX MACRO and Instruction Set Reference Manual*, Digital Equipment Corporation, order number: AA-Z700A-TE, September, 1984.

(4) *Programming in VAX C*, Digital Equipment Corporation, order number: AA-L370B-TE, April, 1985.

(5) *VAX/VMS Internals and Data Structures* by Lawrence Kenah and Simon Bate, Digital Press, 1984.

```
        .TITLE  TRVEC   - Transfer vectors for a shareable image (prototype)
        .IDENT  /1.0.0/

;
; Author:       Ted Marshall, Britton Lee, Inc.
;               14600 Winchester Blvd, Los Gatos, CA 95030
; This program may be copied and used without restriction.
;
; This file has been set up to allow the creator of a shareable image to
; easily set up transfer vectors for it. This will do most of the work
; required to make the shareable image upward compatable.
;
; INSTRUCTIONS:
;
; For each externally accessable symbol (universal symbol) within the
; shareable image that corresponds to a code entry point, add a line
; to the section below as indicated. If the routine is to be called by
; the MACRO-32 CALLx instruction or most high level lenguage routine calls,
; add a line of the form:
;       PROCVEC <symbol-name>
; If the routine is called by the MACRO-32 JSB or BSBx instructions, add a
; line of the following form:
;       SUBVEC  <symbol-name>
; For example, if you had two procedures (CALLx/high-level) named FOO and
; BAR and one subroutine (JSB) named XYZZY, you would add the following lines:
;       PROCVEC FOO
;       PROCVEC BAR
;       SUBVEC  XYZZY
; Note that the semi-colons at the beginnings of the lines are not included.
;
; Note: the order of the lines must be maintained between versions. Always
; add new entries at the end of the list.
;
; Use the following DCL command to assemble this file:
;       $ MACRO/NOLIST TVPROTO
;
; When you link the shareable image, add the following two lines at the
; beginning of the link options file (excluding the semi-colons):
;       CLUSTER = $$$TRVEC
;       COLLECT = $$$TRVEC,$$$TRVEC
; This will guarantee that the transfer vectors go at the absolute beginning
; of the shareable image.
;


;
; Macro definitions: PROCVEC produces a transfer vector for a procedure (called
; by CALLx) and SUBVEC produces a transfer vector for a subroutine (called by
; JSB or BSBx).
;
        .MACRO  PROCVEC entryname
        .TRANSFER entryname
        .MASK   entryname
        JSB     L^entryname+2
        .ENDM

        .MACRO  SUBVEC  entryname
        .TRANSFER entryname
        JSB     L^entryname
        .BLKB   2
        .ENDM

;
; Define a separate program section for the transfer vectors.
;
        .PSECT  $$$TRVEC,RD,NOWRT,EXE,SHR,CON,PIC

;
; Add your PROCVEC and SUBVEC statements here. Always add new entries at the
; end.
;


; Always add new entries just before this line!
;

        .END
```

# A NEW TECHNIQUE FOR "SYSTEM PERFORMANCE EVALUATION"

Schumann Rafizadeh
MBA Systems Automation
Columbus, Ohio

## ABSTRACT

The traditional methods of the systems performance evaluation rely on periodic sampling of the statistics collected by the operating systems to be stored and later graphed or reported. The sampling routines for these methods must operate at a high priority in order to collect all the samples. Now, it is more efficient and relevant to use LORE method for collecting performance statistics whcih are very reliable and meaningful to the users at a very low priority. This method also poses no contentions with user applications for valuable system resources nor requires knowledge of monitor internals to understand the results.

## OVERVIEW

There is a need to study the requirements and develop all the necessary tools to measure the performance of a system and all its related resources based on a set of variables defined below. Part of this paper will deal with providing a tool to measure the response factors for users in a manner which they can relate to their applications and be able to verify independently based on the major system resources.

In the past, this issue has been addressed on most operating systems based on the CPU stretch factor and a constant interval sampling scheme. This simple approach can provide numbers which are valid for the parameters concerned and, given knowledge of the operating systems internals and some guess work, can be interpreted to demonstrate a relative performance measure.

For the future, we not only need the same capability (in a more comprehensive form) but also the capabilities to:

1) Performance threshold limit calculations for each resource to develop "Base Performance Levels" (BPL), measurement units and to define the "acceptable Performance Level" (APL).

2) Forecast system performance under new criteria and load by measuring the level of free (or used) resources.

3) Continuous and on demand measurement of valid system utilization parameters ( for resources, applications, etc. ) and comparison of this figure to the APL and BPL.

4) Reporting and presentation of the information in a comprehensive and conclusive manner (graphic and relative) both to the users and system managers.

5) Forecast system performance under new criteria and to determine the optimum load mix for different computers and configurations.

6) Configuration Performance Threshold (CPT) limits for each system and specific configurations to determine the effects of additional resources and loads on the system as a whole. This will be used as a basis for determining the APL, and APL may be a multiple of this limit.

It is also desirable to be able to measure application loads to better forecast system responses, perform load balancing in the network and identify unreasonably inefficient (or efficient!) programs.

## PERFORMANCE

Performance can generally be defined as: the amount of useful work done per unit of time (this should be true for anything that works!) The performance of computer resources can be measured regardless of their brand names for the same types of application processing.

Response time then can be defined as the time interval between successful submission of a request or response to the system until the next prompt or completion result (or error message!) is returned.

Under timesharing environment, the requests will be honored and carried out depending on the scheduling system and priority of each user. Depending on the demand for the resources, the request will be queued and in this fashion it is possible to have requests queued faster than they can be dequeued by the proper service module. This will result in delays of responses and in turn reduce the number of requests until the system can catch up.

The most efficient use of the machine can be acheived at a point where the maximum amount of work can be carried out per unit of time and this requires tuning of all the components involved in a delicate balance of supply and demand. These components are hardware, system software, and user applications.

It is important to know the performance threshold of the major components of each system and overall available resources for each machine. Based on these values and expected return on each machine, it is possible to define Acceptable Performance Limit

(APL) for each system. These limits will provide users with the necessary information about our resources to allow for budgeting of their usage and balancing of their workloads.

Operations usually commit to provide an X percentage of the time the response time which is better than APL. It is possible to identify this performance level to the user and using the tools described here ensure monitoring this commitment to maintain a high level of confidence in operations by documenting the percentage of the time operations were able to meet their goals between the hours of interest to the users. This also means that the new systems will be turned over to the operations only if they meet their specified performance requirements.

SYSTEM A PERFORMANCE (period)

```
  !        AA
P ! AA     AA     AA
E ! AA     AA     AA
R ! AA     AA     AA     AA
F ! AA     AA     AA     AA
O ! AA     AA     AA     AAF
R ! AAF    AAF    XX     AAF
M ! XXF    AAF    XX     AAF
A ! XXF    XXF    XX     AAF
N ! XXF    XXF    XXF    XXF
C ! XXF    XXF    XXF    XXF
E ! XXF    XXF    XXF    XXF
  ! XXF    XXF    XXF    XXF
  ------------------------------
    CPU    MEM    DSK    COM
```

FOR THE MONTH OF JUNE 1986

WHERE,

| | |
|---|---|
| AA | is the resource BPL |
| XX | is the resource APL |
| F | is the average Free |
| period | is a given time period |

The above figure is a sample performance chart for a System A. The AA level denotes the available Base Level Performance. The acceptable performance level is the lowest level of free resources that can deliver acceptable performance to the users. The average of these APL's (normallized) will be APL for the system or System Performance Level (SPL).

The same charts can be obtained for each major application function. This can be used for proper load distribution on the systems and the continual monitoring, tuning and load balancing will allow operations to achieve the optimum performance levels and to prove that the system is not the cause of poor response time, where appropriate.

Using this method to determine the resource requirements for each function will allow the development group to build the required performance level into each application system as it is designed and developed or to take necessary precautions not to raise the user expectations unreasonably at initial installation before full utilization of the system (see System Usability).

STRETCH FACTOR COMPONENTS

It is not sufficient to measure the stretch factor based on the CPU availablility alone. Because, it is possible for a user to experience very high stretch factors based on a bottlenecked resource such as disk or memory without ever being recognized by looking at the CPU stretch factors. This is like measuring the demand for a resource while users are in a waiting queue for another resource. (The stretch factor for a user waiting for a report to be printed on a device which has other files queued before his cannot be measured by looking at the CPU usage or any other resource but the printer's speed and the size of the files to be printed yet and the throughput of the printer over the period necessary to print a line or a character.)

Therefore, the stretch factor must be measured for the particular application and/or system based on the resources required to perform that application or the application load and the current demand on those resources. For this reason, to measure the performance or the stretch factor, one will need to know the performance threshold (maximum available throughput) for each individual resource and for that resource in conjunction with other resources and their availability. Furthermore, these performance thresholds should be standardized in form of the units to compare and measure the response delays based on different demand levels.

Even though the availability of every component involved in the system to perform a request should be included for proper performance measurement (and there are many of them), we shall concentrate on Primary Factors (PF) for now.

Following is a list of resources which are to be included as resources (Primary Factors) of the Performance Evaluation utilities:

1) Processor (CPU)
2) Main Memory (MEM)
3) Disk Subsystem (DSK)
4) Communication (COMM)

Then the Stretch Factor (SF) for a load should be calculated as:

SF (ALL) =
SF(CPU)+SF(MEM)+SF(DSK)+SF(COMM)

On computers designed under PMS (Processor, Memory, Switch) notation, such as VAX the first 3 factors are considered critical factors. The overall system performance is dependent on efficient inter-play of all of its components. Also, the relative cost of the first three components are significantly higher than the last component in majority of distributed environments.

PERFORMANCE MEASUREMENTS

The three commonly used methods for performance measurements are

1) Benchmarking
2) Statistics Collection
3) Simulation

Benchmarking is a common method for performance evaluation. In this method a benchmark set is run and performance measures are taken based on the elapsed time and the demand placed on the resources under

test. This technique is good for comparative analysis and is repeatable for static environments. It is usable for timesharing environment but not by itself.

Statistic Collection is the most common method. This method is usually built into the operating system and can be optionally enabled to collect and report vital information about the performance related parameters on the system. The draw back of this method is that it totally relies on the expertise of a very knowledgeable system specialist to remedy the inefficiencies or to perform tuning.

Simulation is a valuable performance measurement tool for conditions where the prototype or a model is to be used to simulate the environment or the load and thus may not be exact. This method is the best way of forecasting the performance. The forecasted information can be used to tune the performance without requiring the specialist.

A system can be developed to use benchmarking upon installation to measure the Base Performance Level (BPL) of the resources and the system. This system can also use the periodic benchmarking at the lowest priority to measure the percentage of the free (available) resources based on previously measured BPL's. This eliminates the need for high priority process of sampling monitor collected statistics. Then use simulation to measure the impact of adding users which take known percentage of the resources for forecasting purposes. Finally, this system can use a graphing routine to present the results for daily, weekly and monthly review by the system manager.

Benchmarking and simulation testing of the application systems will ensure that the application is capable of providing the expected response time and also establish the impact the application has on the computer when running at different load levels and also might point out hardware solutions and limitations. The application profiles produced in this manner can be extremely useful in finding the optimal application mixture on a computer and predicting results.

NOTES

1. Statistical method used by SPM which relies on MONITOR sampled statistics of VMS counter is a valuable tool to measure the relative performance of different versions of VMS. For example, a given fixed load which requires a set of I/O and computations may require different amount of machine resources to complete under different versions of the VMS. However, this method cannot generate reliable results on heavily utilized machines or resources since the statistics gathering method itself demands a high amount of resources which it robs from the users it is trying to monitor!

2. The current chargeback systems based on the systems collected statistics are also invalid, because parameters such as connect time are inversely related to system performance. This means a user pays a higher amount for the same functions when the system response is worse.

LORE METHODOLOGY

This method is based on measuring the BPL of all the resources on the machine for a given configuration with no other user load using a fixed set of functions. Then on an on-going basis it tries to perform the same measurement routine at lowest priority, without contending with users, and collecting these statistics. The amount of Left Over Resources Evaluated (LORE) subtracted from the BPL will determine the user loading for resources and the machine. The left-over resources as a percentage of the total resources (100*LORE/BPL) determines the percentage of free resources. The level of the resources on a machine that can still deliver Acceptable Performance to all of its users is then defined as APL.

REPORTING AND PRESENTATION

Once the Base (Threshold) Performance Limits (BPL) have been established, acceptable responses can be measured in terms of BPL units (or percentage). This means that system response for a request then can be represented in terms of a percentage of this acceptable limit. For example,

"SYSTEM RESPONSE WAS ACCEPTABLE 95 PERCENT OF THE TIME FOR DEVELOPMENT"

or,

"5 PERCENT OF THE TIME SYSTEM RESPONSE WAS NOT ACCEPTABLE DURING THE LAST PERIOD."

or,

"SYSTEM RESPONSE IS WELL WITHIN ACCEPTABLE LIMIT 95 PERCENT OF THE TIME"

The graphic displays of the system performance are much more understandable. The presentation part of the utility programs involved in this project must be developed modularly to allow incorporation of plotting routines and support for a variety of display devices.

Another desired feature will be estimating load and stretch factors for given application systems and their impact on a host. This can be used for dynamic balancing of application systems on the hosts during host and node failures.

SUMMARY

This paper presents a new approach for performance evaluation and capacity planning, especially in a mixed vendor environment. This approach emphasizes and justifies the views and perceptions of the users about the system performance and availability of its subsystems. While all the traditional methods of performance evaluation measure what system the (especially the given operating system) feels about the user applications and basic components of their tasks such as the page faulting rates.

# PROGRAMMING WITH THE VAX/VMS SCREEN MANAGEMENT ROUTINES

Michael D. Orosz
E M C Engineers, Inc.
Denver, Colorado

## ABSTRACT

As the computer is increasingly called upon to perform a variety of tasks, computer programs designed to accomplish these tasks must exhibit a high degree of user friendliness. Since the majority of computer users are not data processing professionals, a well-planned and well-written package is essential in order for these users to effectively work with the system. Such features as function keys and attractive video displays can go a long ways toward making an application package user friendly. To include these features in a program, the programmer usually researches the various control codes and sequences for the desired video terminals. This procedure typically involves a considerable amount of time and is usually terminal dependent. To help reduce this time constraint and develop a program without concern for the type of terminal being used, a set of VAX/VMS utility routines can be used. These utilities, collectively called the Screen Management Facility, enables a programmer to utilize function keys and create sophisticated video displays with a minimum of programming effort.

## INTRODUCTION

With the introduction of VAX/VMS V4.x, a set of routines designed for managing the terminal screen have been added to the VAX/VMS Run-Time Library [1]. These routines, collectively called the Screen Management Facility or Screen Management Guidelines (SMG), will enable a programmer to develop sophisticated I/O (input/output) techniques without concern for the type of device being addressed. This paper will discuss the screen management concept, why the routines should be used, and cover various programming techniques which utilize the SMG routines.

## SCREEN MANAGEMENT CONCEPT

The Screen Management Facility utilizes internal data structures to represent a physical device. These structures are addressed by an application program when I/O operations are performed. SMG is responsible for the transactions between the internal data structures and the physical device that is mapped to them. This design eliminates the need for a programmer to research the escape and control code sequences required for sophisticated terminal interaction.

Although SMG is designed for video terminals, hardcopy terminals and other devices can also be addressed. Care should be taken when using these nonvideo devices since different I/O techniques are usually employed when addressing them. For example, an application that relies on cursor positioning will not work on a hardcopy terminal. Since video terminals are typically found in most installations, they will be the focus of this paper.

Three data structures are used by SMG to represent a video terminal; the pasteboard, the virtual display, and the virtual keyboard. These structures are addressed by the application program when performing I/O operations.

### Pasteboards

A pasteboard is a two dimensional data structure that SMG uses to represent a video screen. Images are displayed on the pasteboard and SMG converts these images into instructions and sends them to the terminal screen. Since the application program never performs any physical operation with the screen, terminal independence is achieved.

In theory, the pasteboard is as large as memory will allow. However, the portion of the pasteboard that is visible to the user is the rectangular region that maps directly to the physical screen.

The upper left-hand corner of the pasteboard maps to the upper left-hand corner of the video screen. The size of the visible rectangle depends on the number of rows and columns available in the video screen (Figure 1).

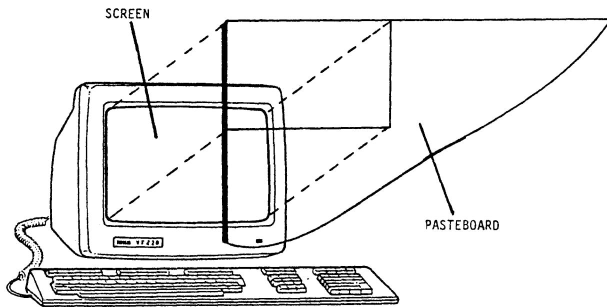



Figure 1

Since only a small portion of the pasteboard is visible to the user, it is important that images be placed within this visible region. If images extrude beyond the visible boundaries, then part or all of the image will be invisible to the user (Figure 2).

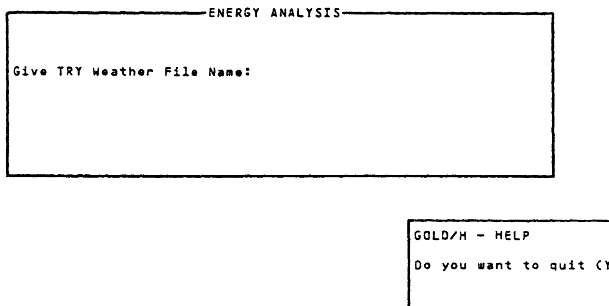



Figure 2

Virtual Displays

Virtual displays are two dimensional areas on the pasteboard in which an application program's output is concentrated. In addition to output operations, these displays, along with a virtual keyboard (see below), can also be used for data entry operations.

By utilizing multiple virtual displays, each one designed for a specific task, the application program can control where on the pasteboard (screen) I/O is to take place. This control of the screen can be used to provide a user friendly interface to a program (Figure 3).

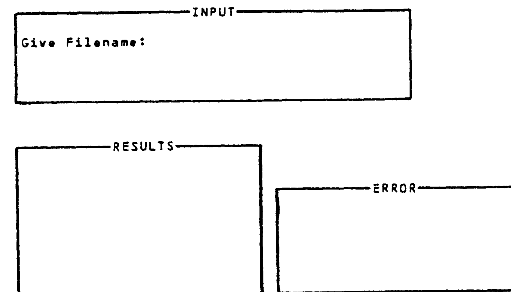



Figure 3

Virtual Keyboards

In addition to pasteboards and virtual displays, SMG utilizes a data structure called a virtual keyboard to represent a physical input device. SMG collects the data from the physical device and stores the information in this data structure. The application program reads the data from the virtual keyboard; therefore, eliminating any direct contact with the physical input device.

WHY USE SMG?

Since visual aids typically provide informative and interesting results to a user, considerable effort should be expended on developing them. In addition, time should be spent on developing sophisticated data entry techniques in order to reduce user fatigue and error. SMG can be employed to help achieve these results with a minimum of programming effort.

SMG can also address special events such as broadcast message trapping (VAX/VMS MAIL messages for example [2]) or control/key interrupts [3]. These events typically interfere with normal program execution and therefore require special attention. By using SMG to control these events, the QIO and related system service routines can be avoided [4].

When using SMG for program I/O, less source code is required in program development. The Screen Management Facility is responsible for the data structures and related logic required for sophisticated I/O operations. In developing an application, the only I/O related instructions required are the calls to the appropriate SMG routines.

Finally, as indicated in previous paragraphs, terminal independence is achieved when using SMG to perform I/O operations. The transfer of data is conducted with internal data structures instead of the physical device. This design saves considerable development time since escape and control code sequences for each terminal do not have to be research.

## PROGRAMMING TECHNIQUES

The following discussion is intended to demonstrate some of the I/O techniques that can be developed by incorporating the SMG routines. This discussion does not exhaust all the possible methods or techniques that exist in creating user friendly programs, rather, it is intended to provide the user with some insight on how SMG can be utilized.

### Pasteboards

As indicated, a pasteboard must be defined in order to use SMG in performing I/O operations. The routine SMG$CREATE_PASTEBOARD is used to create a pasteboard and map it to a video screen. SMG$CREATE_PASTEBOARD returns to the calling program an identification number (ID) that identifies the pasteboard. Care should be taken to assure that this number is not altered. If the ID number is modified, then SMG will not be able to locate the pasteboard in internal memory. In addition, this value should be passed to all subprogram modules (via an argument list or common storage technique) that will be using SMG to perform I/O operations. By passing the ID number, considerable overhead is reduced since the subprogram module will not have to call SMG$CREATE_PASTEBOARD in order to obtain the ID number (Figure 4).

```
      IMPLICIT INTEGER*4 (A-Z)          !Declare data types

      INCLUDE '($SMGDEF)'               !Include system definitions.

      COMMON/ID/PBID,VID                !Store SMG ID numbers in memory

C Create pasteboard and virtual display.  Virtual display have 10 rows
C and 60 columns (and a border).

      STATUS=SMG$CREATE_PASTEBOARD(PBID,,,,)
      STATUS=SMG$CREATE_VIRTUAL_DISPLAY(10,60,VID,SMG$M_BORDER,,)

C Call routine to paste virtual display (make visible) to pasteboard.

      CALL OUTPUT

C Done - Delete pasteboard and erase screen.

      STATUS=SMG$DELETE_PASTEBOARD(PBID)

      STOP
      END

      SUBROUTINE OUTPUT                 !Routine used to paste virtual display.

      IMPLICIT INTEGER*4 (A-Z)

      COMMON/ID/PBID,VID                 !Common memory holds SMG ID numbers.

C Paste virtual display on pasteboard with upper left-hand corner at row 5,
C column 5.

      STATUS=SMG$PASTE_VIRTUAL_DISPLAY(VID,PBID,5,5)

      RETURN
      END
```

Figure 4

SMG$CREATE_PASTEBOARD also returns (as an option) the number of rows and columns the physical screen contains (Figure 5). These values define the visible region of the pasteboard and can be used to assure that images are placed within this visible area.

```
$CREATE TEST.FOR

        IMPLICIT INTEGER*4 (A-Z)

        STATUS=SMG$CREATE_PASTEBOARD(PBID,,ROW,COLUMN,)

        WRITE(*,10)ROW,COLUMN
10      FORMAT(1X,'Number of Rows on screen:    ',I3,/,
     1         1X,'Number of Columns on screen: ',I3)

        STOP
        END

$FORTRAN TEST
$LINK TEST
$RUN TEST

Number of Rows on screen:    24
Number of Columns on screen: 80
FORTRAN STOP
$
```

Figure 5

### Virtual Displays

To display information on a pasteboard, a virtual display is required. The routine SMG$CREATE_VIRTUAL_DISPLAY is used to define a virtual display. The number of rows and columns the display will encompass on the pasteboard is passed to SMG$CREATE_VIRTUAL_DISPLAY via the argument list. It is important that the dimensions of the virtual display do not exceed the dimensions of the visible region. If the display's dimensions exceed the boundaries of the pasteboard's visible region, then each dimension should be adjusted accordingly (Figure 6).

```
      IMPLICIT INTEGER*4 (A-Z)                       !Declare data types.

      DISPLAY_ROW=25                                 !Virtual Display's
      DISPLAY_COL=87                                 !Dimensions.

C Create pasteboard.

      STATUS=SMG$CREATE_PASTEBOARD(PBID,,ROW,COLUMN,)

      IF(DISPLAY_ROW.GT.ROW)THEN
          DISPLAY_ROW=ROW                            !Adjust Dimension.
      ENDIF
      IF(DISPLAY_COL.GT.COLUMN)THEN
          DISPLAY_COL=COLUMN
      ENDIF

C Create virtual display.

      STATUS=SMG$CREATE_VIRTUAL_DISPLAY(DISPLAY_ROW,DISPLAY_COL,VID,,,)

C Make display visible to user (paste upper left-hand corner to row 5,
C column 5 on pasteboard).

      STATUS=SMG$PASTE_VIRTUAL_DISPLAY(VID,PBID,5,5)

C Done.

      STATUS=SMG$DELETE_PASTEBOARD(PBID)

      STOP
      END
```

Figure 6

SMG$CREATE_VIRTUAL_DISPLAY returns an unique identification (ID) number used to identify the display in internal memory. As with pasteboards, this number should not be altered. This requirement is important since an application

program can have many virtual displays defined and the ID number is used to uniquely identify a particular display. Unfortunately, once a virtual display has been created, there is no method for determining the ID number. Therefore, the display's ID number needs to be passed (via an argument list or common storage) to subprogram modules that will be using the display for I/O operations (Figure 7).

```
        IMPLICIT INTEGER*4 (A-Z)                !Declare data types

        INCLUDE '($SMGDEF)'                     !System Definitions.

        COMMON/ID/PBID,VID1,VID2                !Common storage for
                                                !SMG ID numbers.

C Create pastboard, and two virtual displays (each one with a border).

        STATUS=SMG$CREATE_PASTEBOARD(PBID)
        STATUS=SMG$CREATE_VIRTUAL_DISPLAY(8,60,VID1,SMG$M_BORDER,,)
        STATUS=SMG$CREATE_VIRTUAL_DISPLAY(8,60,VID2,SMG$M_BORDER,,)

        CALL OUTPUT                 !OUTPUT is used to paste displays.

C Done.

        STATUS=SMG$DELETE_PASTEBOARD(PBID)

        STOP
        END

        SUBROUTINE OUTPUT

        IMPLICIT INTEGER*4 (A-Z)

        COMMON/ID/PBID,VID1,VID2                !Common Storage.

C Paste displays.

        STATUS=SMG$PASTE_VIRTUAL_DISPLAY(VID1,PBID,2,10)
        STATUS=SMG$PASTE_VIRTUAL_DISPLAY(VID2,PBID,12,10)

        RETURN
        END
```

Figure 7

Finally, a border can be specified for the virtual display. This border surrounds the display when it is pasted (made visible) to the pasteboard. By specifying a border, the display can be distinguished from surrounding images on the screen (Figure 8).

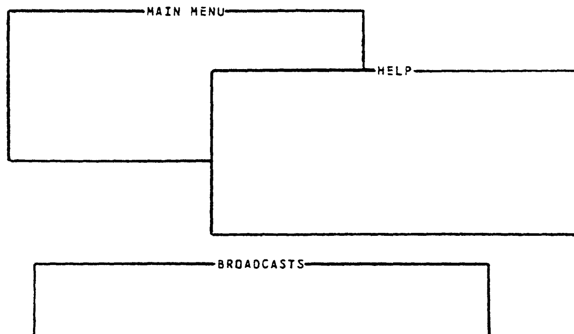



Figure 8

Reading

SMG provides several routines for collecting data from the virtual keyboard. The routine SMG$READ_STRING however, is unique since most features of the terminal driver can be utilized by the programmer. Escape sequence detection, purging the type-ahead buffer [5], and converting input to upper case characters are some of the features available. By utilizing SMG$READ_STRING, the programmer is able to use these features and avoid using the complicated QIO system service.

To use SMG$READ_STRING, a virtual keyboard identification (ID) number needs to be specified. This ID number is returned by SMG$CREATE_VIRTUAL_KEYBOARD and is used to uniquely identify the virtual keyboard in memory. As with pasteboards and virtual displays, this value should not be altered and needs to be passed to subprogram modules that will be performing data entry operations.

To utilize the terminal driver's features, a mask is defined and passed to SMG$READ_STRING via the argument list. Based on the bits set in the mask, the appropriate feature will be enabled. Fortunately, SMG provides a set of symbols (defined by the $TRMDEF macro/module in the DIGITAL supplied system libraries) that can be used to set the desired bits. The following FORTRAN instruction illustrates how to set the bits so that escape character detection and the purging of the type-ahead buffer are enabled:

```
MASK=TRM$M_TM_ESCAPE.OR.TRM$M_TM_PURGE
```

A terminator is used to end user's input from the virtual keyboard. Such terminators include carriage returns and function key escape sequences (assuming that TRM$M_TM_ESCAPE was specified when defining the terminal driver bit mask). This terminator value is returned by SMG$READ_STRING via the argument list and is in symbolic form. The symbols used to represent terminators are defined by the $SMGDEF macro/module (part of the DIGITAL supplied system libraries). This feature is useful in applications that utilize function key data entry (see below).

To assure that the read operation is performed in conjunction with the appropriate virtual display, the display's ID number is passed to SMG$READ_STRING. If the ID number is omitted from the argument list, then the read operation will take place as though no virtual display or pasteboard exists. If this should occur, the read operation will begin in column 1 of the screen instead of column 1 of the virtual display (Figure 9).

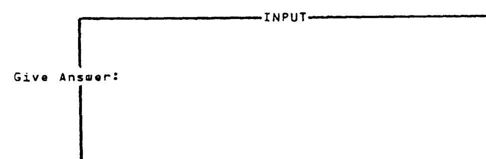



Figure 9

Special Events

SMG is useful for controlling special events such as broadcast messages and control/key interrupts. These events usually disrupt images on the screen or effect program execution. By utilizing the SMG routines to control these events, the complicated QIO and related system services can be avoided.

The routine SMG$SET_BROADCAST_TRAPPING is used to trap broadcast messages before they are sent to the video screen. When a message is trapped, program execution halts and an AST routine is called (the AST routine is specified as an argument in SMG$SET_BROADCAST_TRAPPING). This AST routine can then collect the message by using SMG$GET_BROADCAST_MESSAGE. Once the AST routine has obtained the message, a virtual display can be used to inform the user. The AST routine pastes (makes visible) the display to the pasteboard and writes the captured message to the display (Figure 10). Care should be taken to assure that the display does not occlude or interfere with images currently pasted (visible) on the pasteboard. The key here is to inform rather then confuse the user.

```
        IMPLICIT INTEGER*4 (A-Z)              !Declare data types.

        INCLUDE '($SMGDEF)'                   !System Definitions.

        COMMON/ID/PBID,BROADVD                !Common Storage for
                                              !storing SMG ID numbers.

        EXTERNAL BROADCAST                    !Declaring AST routine used
                                              !by SMG$SET_BROADCAST_TRAPPING

C Create pasteboard and broadcast display (label broadcast display).

        STATUS=SMG$CREATE_PASTEBOARD(PBID)
        STATUS=SMG$CREATE_VIRTUAL_DISPLAY(5,78,BROADVD,SMG$M_BORDER)
        STATUS=SMG$LABEL_BORDER(BROADVD,'BROADCASTS')

C Set broadcast message trapping.

        STATUS=SMG$SET_BROADCAST_TRAPPING(PBID,BROADCAST,)

                    .                   !Main Body of Program.
                    .
                    .

C Done.
        STATUS=SMG$DELETE_PASTEBOARD(PBID)

        STOP
        END

        SUBROUTINE BROADCAST

        IMPLICIT INTEGER*4 (A-Z)              !Declare data types.
        INTEGER*2 LEN
        CHARACTER MESSAGE*80

        COMMON/ID/PBID,BROADVD

C Get broadcast message and display to user.

        STATUS=SMG$GET_BROADCAST_MESSAGE(PBID,MESSAGE,LEN)
        STATUS=SMG$PASTE_VIRTUAL_DISPLAY(BROADVD,PBID,18,2)
        STATUS=SMG$PUT_CHARS(BROADVD,MESSAGE(1:LEN),3,1)

        RETURN
        END
```

Figure 10

In addition to broadcast trapping, control/key sequences can also be addressed by SMG. Such events as CTRL/Y and CTRL/C (pressing the control key and then the Y or C keys simultaneously) usually serve as a means of prematurely ending program execution. Unfortunately data files and other resources currently being used by the program may be altered or destroyed due to these events.

The SMG routine SMG$SET_OF_BAND_ASTS will trap control/key events and, if specified in the argument list, call the appropriate AST routine. This AST routine can then be designed to handle the control/key event in a user friendly manner. A mask is used to specify which control/key events will be trapped. This mask is defined by setting the bit that corresponds to the desired control/key character [6]. If no bits are set then no control/key event will be trapped. Care should be taken that only those control/key events desired are specified in the mask. For example, if CTRL/M is set in the mask, then a carriage return (typically used to terminate input) will stop program execution and call the specified AST routine instead of terminating user input (Figure 11).

```
        IMPLICIT INTEGER*4 (A-Z)              !Declare data types.
        INTEGER*2 TERMINATOR
        CHARACTER ANS*5

        COMMON/ID/FLAG,PBID,VDID,KID          !ID numbers for ASTSUB.

        INCLUDE '($SMGDEF)'                   !System definitions.
        INCLUDE '($TRMDEF)'
        INCLUDE '($SSDEF)'

        EXTERNAL ASTSUB                       !Declare routine used
                                              !by SMG$SET_OUT_OF_BAND_ASTS

        FLAG=.FALSE.
        MODIFIER=TRM$M_TM_PURGE.OR.TRM$M_TM_ESCAPE.OR.TRM$M_TM_CVTLOW
        MAINSTATUS=.TRUE.

        MASK=0                                !Set up mask.

C Very important that CTRL/M is enabled if CR is used to terminate input.

        MASK=JIBSET(MASK,13)
        MASK=JNOT(MASK)                  !Sets Most Control/key trapping
                                         !(compliment)

C Create pasteboard, display and keyboard.  Declare out of band asts.

        STATUS=SMG$CREATE_PASTEBOARD(PBID)
        STATUS=SMG$CREATE_VIRTUAL_DISPLAY(10,60,VDID,SMG$M_BORDER)
        STATUS=SMG$LABEL_BORDER(VDID,'MAIN DISPLAY',,,SMG$M_BOLD)
        STATUS=SMG$CREATE_VIRTUAL_KEYBOARD(KID)
        STATUS=SMG$PASTE_VIRTUAL_DISPLAY(VDID,PBID,3,3)
        STATUS=SMG$SET_OUT_OF_BAND_ASTS(PBID,MASK,ASTSUB,LIST)

C Get user input.

        DO WHILE(MAINSTATUS)
                STATUS=SMG$SET_CURSOR_ABS(VDID,3,1)
                STATUS=SMG$READ_STRING(KID,ANS,'PROMPT> ',5,MODIFIER,,LEN,
     1                  TERMINATOR,VDID)
                IF(STATUS.NE.SS$_ABORT)THEN      !If abort then READ_STRING
                                                 !cancelled.
                        IF(FLAG)THEN
                                FLAG=.FALSE.
                                STATUS=SMG$DELETE_CHARS(VDID,25,8,1)
                                STATUS=SMG$DELETE_CHARS(VDID,25,9,1)
                        ENDIF
                        IF(ANS(1:4).EQ.'EXIT')THEN
                                MAINSTATUS=.FALSE.
                        ENDIF
                        STATUS=SMG$PUT_CHARS(VDID,'User response: ',6,1)
                        STATUS=SMG$PUT_CHARS(VDID,ANS(1:LEN),6,18)
                ENDIF
        ENDDO

        STATUS=SMG$DELETE_PASTEBOARD(PBID)       !Done with program.

        STOP
        END

        SUBROUTINE ASTSUB(LIST)

        IMPLICIT INTEGER*4 (A-Z)         !Declare important data types.
        CHARACTER TEMP*4
        COMMON/ID/FLAG,PBID,VDID,KID     !Common contains values from calling
                                         !routine.

        INCLUDE '($SMGDEF)'              !System definitions.

        STATUS=SMG$CANCEL_INPUT(KID)     !Cancel outstanding read operation.

        FLAG=.TRUE.                      !Notify user.
        STATUS=SMG$PUT_CHARS(VDID,'Control/Key pressed',8,1,,SMG$M_BOLD)

        RETURN                           !Done.
        END
```

Figure 11

## Scrolling

SMG allows for the scrolling of text within a virtual display. Unlike hardware scrolling, where the whole width of the screen in involved, SMG scrolls only lines within the virtual display. In applications that scroll one line at a time, SMG erases and then repaints the display for each line scrolled. Unfortunately, this technique requires considerable processing time and the resulting sluggish performance will often frustrate the user.

A solution to this problem involves a technique called jump scrolling. Instead of erasing and repainting the display for each line scrolled, jump scrolling erases and then repaints the display with the next N lines, where N is the number of lines in the display. In Figure 12, two sequences of displays are present. The displays under column A illustrate scrolling line by line. The displays under column B illustrate jump scrolling.

```
        A                      B

  ------HELP------      ------HELP------
  Line 1                Line 1
   .   2                 .   2
   .   3                 .   3
   .   4                 .   4
   .   5                 .   5
   .   6                 .   6
   .   7                 .   7
  Line 8                Line 8

  ------HELP------      ------HELP------
  Line 2                Line 9
   .   3                 .   10
   .   4                 .   11
   .   5                 .   12
   .   6                 .   13
   .   7                 .   14
   .   8                 .   15
  Line 9                Line 16

  ------HELP------      ------HELP------
  Line 3                Line 17
   .   4                 .   18
   .   5                 .   19
   .   6                 .   20
   .   7                 .   21
   .   8                 .   22
   .   9                 .   23
  Line 10               Line 24

  ------HELP------      ------HELP------
  Line 4                Line 25
   .   5                 .   26
   .   6                 .   27
   .   7                 .   28
   .   8                 .   29
   .   9                 .   30
   .   10                .   31
  Line 11               Line 32
```

Figure 12

To include jump scrolling in an application, design the program logic to erase (SMG$ERASE_DISPLAY) and then repaint (SMG$PUT_LINES) the display with the next N lines when a scrolling operation is required (Figure 13).

```
                               .
                               .
                               .
      DISPLAY_LINE=1             !Marks location cursor is at
                                 !in display.

      LINE=1                     !Used to mark location in text
                                 !buffer.
                               .
                               .
                               .

      DO WHILE(MAINSTATUS)
              STATUS=SMG$SET_CURSOR_ABS(VID,DISPLAY_LINE,1)
              STATUS=SMG$READ_STRING(KID,ANSWER,,2,MASK,,,,TERMINATOR,
     1                     VID)
C Check to see if display should be scrolled.

              IF(TERMINATOR.EQ.SMG$K_TRM_DOWN.AND.DISPLAY_LINE.EQ.8)THEN
                           STATUS=SMG$ERASE_DISPLAY(VID,PBID)

C Repaint display with next 8 lines of text.

                           DO 10 I=1,8
                                LINE=LINE+1
                                STATUS=SMG$PUT_LINE(VID,BUFFER(LINE))
10                         CONTINUE
                           DISPLAY_LINE=1
              ELSE

                           DISPLAY_LINE=DISPLAY_LINE+1
                           LINE=LINE+1

              ENDIF

       ENDDO

                               .
                               .
                               .
```

Figure 13

## Function Keys

To minimize user input error and fatigue, function keys can be utilized. These keys are located on the keyboard and when pressed by the user, a special escape sequence is passed to the application program. Based on the key pressed, the appropriate program function is activated. The routine SMG$READ_STRING returns to the calling program (via the terminator argument) the function key entered by the user. As indicated earlier, SMG assigns an unique symbolic name (defined by the $SMGDEF macro/module in the DIGITAL supplied system libraries) to the escape sequence. The program logic then checks the terminator argument to determine which key was pressed by the user. Terminal independence is achieved since symbols are used instead of escape sequences for function key detection.

An unique approach to using function keys is to incorporate the GOLD/key terminology in the program. GOLD/key involves pressing the PF1 key (located on the auxiliary keypad) and then pressing a key on the keyboard/keypad. The application program is designed so that when the PF1 key is pressed (the symbol SMG$K_TRM_PF1 is returned by SMG$READ_STRING), the program logic automatically issues a second read operation to the user (Figure 14). This second read operation is then used to collect the second input of the GOLD/key operation. Based on the key entered, the appropriate program function is called. Most of today's applications (such as word processing packages or the EDT editor) use the GOLD/key technique for user data entry. By incorporating this terminology in program development, start-up time is reduced since it is likely the user is already familiar with this technique.

In addition, by designing programs that use familiar names or letters for accessing program modules, less training time is required in order to get the new program running. Features such as GOLD/H (for accessing on-line help) and GOLD/E (for exiting a program) are logical methods of accessing subprogram modules and do not have to be remembered by the user. This technique provides a convenient user friendly interface to a program (Figure 14).

```
         IMPLICIT INTEGER*4 (A-Z)                          !Declare data types.
         INTEGER*2 TERMINATOR,LEN
         CHARACTER ANSWER*5,PROMPT*13

         INCLUDE '($SMGDEF)'                               !System Definitions
         INCLUDE '($TRMDEF)'

C Create pasteboard, virtual display, and virtual keyboard.

         STATUS=SMG$CREATE_PASTEBOARD(PBID)
         STATUS=SMG$CREATE_VIRTUAL_DISPLAY(10,60,VID,SMG$M_BORDER,,)
         STATUS=SMG$CREATE_VIRTUAL_KEYBOARD(KID)
         STATUS=SMG$PASTE_VIRTUAL_DISPLAY(VID,PBID,5,5)

C Create terminal driver mask for READ_STRING

         MASK=TRM$M_TM_ESCAPE.OR.TRM$M_TM_PURGE.OR.TRM$M_TM_CVTLOW
         PROMPT='Give Answer: '

C Get user input.

         MAINSTATUS=.TRUE.
         DO WHILE(MAINSTATUS)
                 STATUS=SMG$SET_CURSOR_ABS(VID,3,5)
                 STATUS=SMG$READ_STRING(KID,ANSWER,PROMPT,5,MASK,,,
     1                          LEN,TERMINATOR,VID)

C Check user input.

                 IF(TERMINATOR.EQ.SMG$K_TRM_PF1)THEN        !GOLD key.
                         STATUS=SMG$SET_CURSOR_ABS(VID,3,18)
                         STATUS=SMG$READ_STRING(KID,ANSWER(1:1),,1,
     1                                  MASK.OR.TRM$M_TM_NOECHO)
                         IF(ANSWER(1:1).EQ.'H')THEN
                                 CALL HELP                  !Help
                         ELSE IF(ANSWER(1:1).EQ. S')THEN
                                 STATUS=LIB$SPAWN()         !Spawn
                         ELSE IF(ANSWER(1:1).EQ.'E')THEN
                                 MAINSTATUS=.FALSE.         !Done.
                         ENDIF
                 ENDIF

                 STATUS=SMG$DELETE_CHARS(VID,5,3,18)

         ENDDO

C Done.

         STATUS=SMG$DELETE_PASTEBOARD(PBID)

         STOP
         END
```

Figure 14

Menus

Menus are a convenient method of informing the user of the options and functions available in a program. By using several menus, each one displaying only the information necessary for program execution, the user is not overwhelmed by a single congested menu. In addition, using several menus will allow the programmer to establish logical paths to the various functions and features available in the program.

A virtual display is created for each menu. When creating the display, a border is recommended. This border will distinguish the menu from additional displays and images currently visible on the pasteboard. When a particular menu is desired, the application program pastes (makes visible) the display on the pasteboard. As a user moves through the program accessing the various menus, displays are pasted to the pasteboard. The key here is to offset each menu from the previous menu (Figure 15). This cascading technique is helpful since it allows the user to quickly determine where he or she is currently located in the program logic.

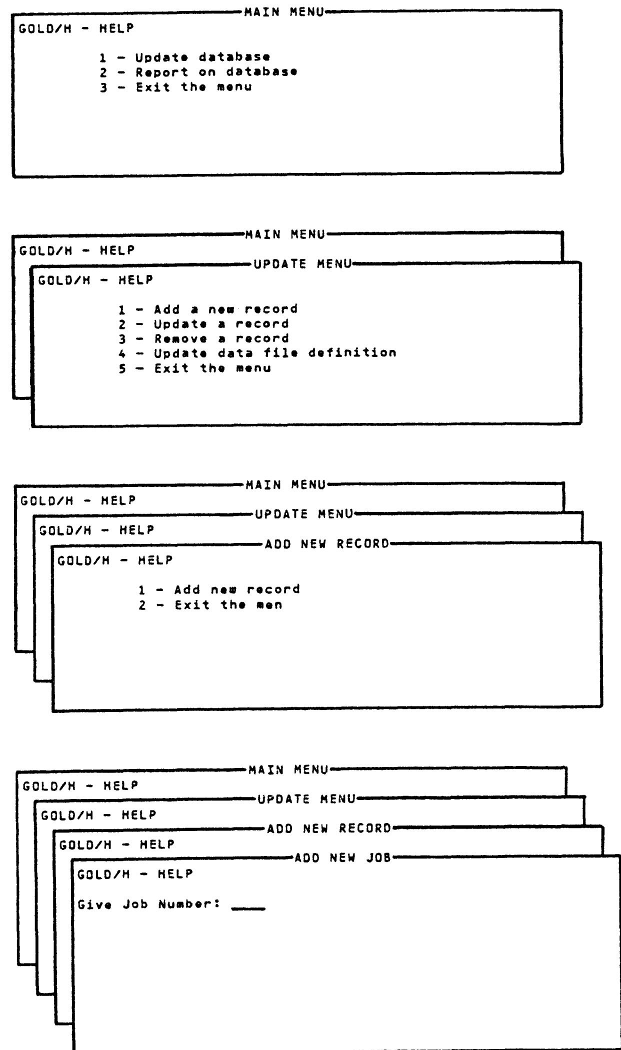



Figure 15

Function keys can be included in a menu driven package. Help for a particular menu can be accessed by entering GOLD/H. GOLD/S can be employed to allow the user to gain access to the system level (DCL level). GOLD/E is useful for allowing the user to quickly exit the program. To utilize these techniques, displays are created and when the the particular function is activated,

pasted to the pasteboard. For example, if GOLD/H is entered, a help display is pasted to the pasteboard (Figure 16). When help is no longer required by the user, the display is unpasted and resides in memory until needed again.

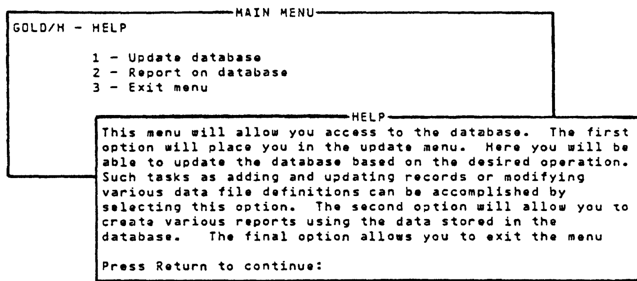



Figure 16

## SPECIAL CONSIDERATIONS

Since SMG maintains an internal representation of all images on the video screen, routines or subprogram modules that perform non-SMG output should be avoided. The images placed on the video screen by a non-SMG routine will not be included in SMG's internal screen representation. This situation can confuse the user since these non-SMG images are placed on a screen without concern for the SMG output currently visible. In addition, when the non-SMG routine is finished processing and control is passed to a routine that is using SMG, updating or modifications to the pasteboard will not reflect the output from the non-SMG routine.

To eliminate this problem, the following technique is recommended. Before calling the non-SMG routine, save the pasteboard contents to internal memory and erase the screen by using the routine SMG$SAVE_PHYSICAL_SCREEN. Call the non-SMG routine and allow it to run to completion. Finally, call SMG$RESTORE_PHYSICAL_SCREEN to erase and then repaint the screen. This technique will help reduce some of the user confusion that may occur.

## CONCLUSION

The Screen Management Facility is a set of VAX/VMS routines that will allow a programmer to develop applications that utilize device independent I/O techniques. These routines reduce the amount of source code required and therefore reduce program development costs. Due to the ease of use and abundant features, program developers should consider these routines whenever developing applications.

## REFERENCES

(1) VAX/VMS Run-Time Library Routines Reference Manual
(2) VAX/VMS Mail Utility Reference Manual
(3) VAX/VMS I/O User's Reference Manual Part I, Section 8.2.1.2
(4) VAX/VMS System Services Reference Manual
(5) VAX/VMS I/O User's Reference Manual Part I, Section 8.2.1.5
(6) VAX/VMS I/O User's Reference Manual Part I, Section 8.4.3.5

# A VMS Facility for Data Encryption to the Data Encryption Standard.

Dr John Yardley
JPY Associates Limited
138 High Street
New Malden, Surrey
ENGLAND KT3 4EP

**ABSTRACT**

This paper discusses data encryption, the Data Encryption Standard (DES) and DATA-LOCK, a VAX/VMS facility for DES encryption. The DATA-LOCK facility was written originally for the European PDP-11 market, mainly in response to export restrictions placed on US implementations of the DES by the NBS. The paper describes the subsequent native-mode VAX/VMS version which, it is believed, is the fastest known software implementation of the DES algorithm. The paper also covers the use of DATA-LOCK, in particular the way it has been designed to operate transparently with VMS facilities and applications using a novel psuedo-device driver. This driver permits regular FILES-11 files to be treated as psuedo-device units in their own right.

## 1. Data Encryption

Traditionally, large computer systems have been protected by a system of user passwords, whereby individual users are given selective access to only certain parts of the system database. The allocation of passwords and privilege is usually in the hands of the System Manager - someone is chosen on his technical merit rather than his corporate importance and who is rarely the "owner" of system data. Such password protection is designed to manage the allocation of resources (such as disk space, CPU time, etc.) and the protection of the "system" rather than the privacy of data.

Systems such as VAX/VMS which are password protected in this way are open to three main abuses:

1. The user who "breaks" the protection system and may discover how to allocate himself more privilege than he is entitled to.

2. The user who discovers the System Manager's password - either deliberately or by accident.

3. The corrupt System Manager. Nothing is ever secret from the System Manager.

While, VAX/VMS handles password protection very securely, users who achieve System Manager status obtain global access to all files on the system so can indeed be a great threat to security. This is because the passwords themselves do not actually protect or do anything to the data, they simply tell the operating system whether the user may access it. Hence, it may even be possible to take data on a removable disk from an "unfriendly" site to a "friendly" one.

The way to prevent these possible abuses is to retain password protection for the allocation of resources - administered by the System Manager - but to allow each individual user to protect his/her files as required. This may be accomplished by ENCRYPTING (or scrambling) the actual data in the file in some way known only to the owner. It then does not matter where the data resides, or who can access it, providing its contents are meaningful to only the owner. This is the principle by which DATA-LOCK works.

## 2. The DES Algorithm

Essentially, there are two methods to encrypt data so as to render it secret; the first is to perform some secret transformation on it - the opponent must then discover the secret transformation to DECRYPT (or unscramble) the data. The second is to make the transformation public, but use some KEY which controls the way it takes place.

The advantage with the public algorithm method for computer-based systems is that there need be no effort in maintaining the secrecy of the encipherment program itself - only the key should

**remain private. There is no point** in disassembling **a published encryption algorithm.** Furthermore, if **the algorithm remains constant,** there is very **little information required** to unlock the data, and **the same computer** program can be run on any **computer** site.

In 1973, recognising the need to adopt a standard algorithm for the encryption of computer data, the US National Bureau of Standards solicited proposals for:

> "Cryptographic algorithms for the protection of computer data during transmission and dormant storage."

The requirements that NBS imposed for acceptable encryption algorithms included the following:

1. They must be completely specified and unambiguous.
2. They must provide a known level of protection, normally expressed in length of time or number of operations required to recover the key in terms of perceived threat.
3. They must have methods of protection based only on the secrecy of the keys.
4. They must not discriminate against any user or supplier.

Subsequent to this, an algorithm submitted by the International Business Machines Corporation (IBM) was selected, to become known as the Data Encryption Standard. The DES algorithm is used by DATA-LOCK.

The Data Encryption Standard defines the encryption of an input block of 64-bits, called the PLAINTEXT, into an output block of 64-bits, called the CIPHERTEXT. This encryption is a function of a 64-bit KEY, of which 56-bits are significant. There are, for any given block of plaintext, 2 to the power 56, or 70,000,000,000,000,000 different ways to encrypt it.

Segments of plaintext greater than 64-bits, may be broken into blocks of 64-bits and enciphered block by block. Any residual bits, in so-called SHORT blocks, may be padded out with random filler bits to form a complete block.

Given any ciphertext block, the only way to discover the original plaintext is to try every possible key in turn. With so many keys this is, in most cases, an impractical proposition. The DES then, is both an extremely strong and well-designed cipher algorithm. This is illustrated by the particularly interesting quality of interdependence of input and output bits - only one bit in the input block need be changed to yield, on average, 32 bits changed in the output block. Or another way, every output bit is a function of every input bit.

## 3. The DES in software

The DES is, by definition, a hardware standard and so the algorithm is very much geared towards execution in hardware. This means that certain operations which are trivial in hardware yet complex to execute in software result in most software implementations being considerably slower than dedicated DES devices. For example, the initial permutation performed on input plaintext (this a one-one transformation) can be implemented in hardware by "hardwiring" bits between input and input-permuted blocks. The execution of this permutation takes place in the time it takes a signal to travel down a wire - a very short time indeed. In software, this is a much more difficult and time-consuming exercise.

While other methods exist for data encryption that are more amenable to software implementation, none has the security or general acceptance of the DES. So that despite the technical difficulty of the DES in software, its benefits make it worthwhile.

The reason that the DES is defined for hardware execution is not, however, for speed considerations. What is important is that the algorithm be hardwired and hence impossible to alter by an opponent. Software cannot offer the same degree of inviolability. However, in systems adopting hardware DES devices, it is not uncommon for the management of keys to be handled by a computer. Since the key management algorithm may be subject to the same abuse as a DES encryption algorithm, the overall system can only ever be a secure as the software.

## 4. The development of DATA-LOCK

In England, and indeed most of Europe, the DES is regarded as the de-facto standard by most banks and major companies. However, hardware for DES encryption is scarce due to the requirement of the US National Bureau of Standards that any product incorporating a US implementation of the DES, needs an individual export licence. Since all the commercially available DES chips are of US manufacture, relatively little hardware has been made or used outside the USA. DATA-LOCK was developed in response to the need for DES encryption facilities - at least for DEC users who would not be "compromised" by a software implementation.

The initial product, developed in 1984, was written for the PDP-11. Before development, there was no real idea of how fast the algorithm could be made to operate, and it seemed quicker to implement it than estimate its speed of operation. The aim was to be able to encrypt at 9600 bit/sec on an 11/23. This speed was eventually exceeded.

DATA-LOCK for the PDP-11 comprised a Transfer Utility, a Key Manager and a CIPHER procedure.

The Transfer Utility operated much like PIP (or COPY), but performed simultaneous encryption or decryption. There were a variety of switches to control the deletion of input files; conversion to

HEXADECIMAL; the passing of keys from the Key Manager; and the choice of encryption mode (cipher-feedback, block-mode, etc).

The Key Manager performed regular database-type operations on user keys. Facilities were included for generation of random keys with user-friendly aliases. These could be passed to the encryption utility in internal mailboxes.

The CIPHER procedure formed the basic building block for DATA-LOCK. It was modelled on the procedure definition proposed by Maytas and Mayer (1) and took the form:

```
CALL CIPHER(PLNTXT,-
            CFRTXT,-
            KEY,-
            LENGTH,-
            FNC,-
            ICV,-
            CHAIN,-
            OCV,-
            SHORT,-
            IPMODE)
```

where:

| | |
|---|---|
| PLNTXT | = Address of plaintext array |
| CFRTXT | = Address of cyphertext array |
| KEY | = Address of key array (8 bytes) |
| LENGTH | = Length of plaintext/cyphertext array |
| FNC | = Function (0=encipher 1=decipher) |
| ICV | = Address of initial chain value (8 bytes) |
| CHAIN | = Encryption mode (0=block 1=chain) |
| OCV | = Address of output chain value (8 bytes) |
| SHORT | = Short block handling mode (0=pad 1=stream) |
| IPMODE | = Initial permutation mode (0=execute 1=suppress) |

DATA-LOCK was originally implemented under RT-11 (and TSX-Plus). The CIPHER procedure was written in MACRO and all other components in Pascal. As a result, DATA-LOCK was relatively easy to convert for use under RSX-11 and RSTS.

The original VMS version of DATA-LOCK ran under the VMS RSX/AME in PDP-11 compatability mode and therefore was suitable only for use on the 730/750/780 series of VAXes. In this mode on the 750 it operated at about the same speed as the LSI-11/73 version.

## 5. Data-Lock and the VAX

It was clear that it would be necessary to develop a native mode version of DATA-LOCK for the VAX. This was because machines such as the MicroVAX 2 did not support compatibility mode and, more importantly, the VAX instruction set was far more amenable to the DES algorithm than the PDP-11 instruction set. Many of the operations in the DES algorithm involve 32-bit and 64-bit manipulations, some of which can be executed in a single VAX instruction.

As a result, a VAX version of the Algorithm was produced at the end of 1985. The CIPHER procedure was completely written from scratch in MACRO-32 rather than converting PDP-11 code into equivalent VAX instructions. The native mode procedure is able to operate at a minimum speed of 80,000-bits/sec on the MicroVAX 2 giving theoretical file transfer rates of some 20 blocks/sec.

The DATA-LOCK DCL interface was made compatible with DEC's VMS Encryption version 1.0 to permit keys to be passed via logical name assignments. The facility to pass keys explicitly or via mail boxes was retained.

A drawback of the encrypting utility approach was that files had to be encrypted and decrypted by issuing a special command. If this was forgotten, un-encrypted files could be left lying around the disk. The solution to this was to somehow intercept reads and writes to disc and encrypt data on write and decrypted on read. This was accomplished by developing a special purpose device driver. Such a driver was designed make DATA-LOCK totally transparent to the user.

## 6. The VMS Virtual Volume Driver (VV) and BIND program

The virtual volume driver operated rather like a logical disk handler under RT-11, whereby users could treat a standard disk file as though it were a device in its own right. The user began by creating a file of specified length on a physical disk, then a virtual volume unit was "bound" to the file using a special program. The virtual volume unit thus had it's own directories, user accounts and files and was completely independent of the real disk on which it resided.

The user made use of the VV driver by invoking a program called BIND. With this, he/she could "bind" to a newly created virtual volume file or to an existing one. This could be done using a DCL command thus:

```
BIND VVA0: VVF.DSK
```

to BIND to an existing file, or:

```
BIND VVA0: VVF.DSK/CREATE/ALLOCATE=200
```

to BIND to a newly created file of, say, 200 blocks.

## 7. The VMS Encryption Driver

A logical extension of the VV driver was to undertake any processing required on read/write operations to the virtual volume file. In the case of DATA-LOCK, the processing was naturally DES encryption/ decryption. The key and indeed any other user-specific information could be passed to the driver as an extension to the BIND program. For example, the command:

```
BIND/ENCRYPT/KEY=SECRET VVA0: VVF.DSK
```

could be used specify DES encryption with the key stored in the logical name "SECRET".

Since all the contents (including volume header) of a newlyinitialised virtual volume file were encrypted with the same key, it would be impossible to MOUNT the same file, following a later BIND operation, with a different key. The user was thus prevented from mixing keys on the same virtual volume file.

With the resulting implementation, it was intended that each user should be allocated one virtual volume unit for his/her exclusive use. In principle however, there was no reason why several users should not have shared access to global file - provided they all knew the correct key.

## 8. Summary

Work done on DATA-LOCK has shown that data encryption to the DES can be performed on most VAX processors at acceptable speeds and in a way that is almost totally transparent to the user. DATA-LOCK can be integrated with almost any other VMS facility to provide extremely high file security.

The specially developed virtual volume handler, has a number of possible applications aside from data encryption. Future efforts at JPY Associates are concentrated on yet faster implementations of the DES and exploitation of these other applications the VV driver.

---

**Reference**


(1) Carl H. Meyer and Stephen M. Matyas, "Cryptography", Wiley-Interscience, 1982.

# VAX NETWORK BACKUPS

**D.G. Darkangelo**

General Electric Company
Corporate Research and Development
Schenectady, New York 12345

This report is aimed at VAX computer owners and managers. It is based on two premises: (1) person hours are more expensive than a reasonable investment in hardware (because of the direct and overhead costs), and (2) backups on a routine basis are necessary on all computers. This report will show how a reasonable investment in disk space for backup files can save money in doing routine backups in a reliable manner.

## 1. INTRODUCTION

This report is aimed at VAX computer owners and managers. It is based on two premises: (1) person hours are more expensive than a reasonable investment in hardware (because of the direct and overhead costs), and (2) backups on a routine basis are necessary on all computers. This report will show how a reasonable investment in disk space for backup files can save money in doing routine backups in a reliable manner.

Note that the method set forth does not use disk shadowing. This is because disk shadowing is not available yet in VMS. In addition disk shadowing, when available, will support only cluster backups, not network backups. Sometime in the future, disk shadowing may be incorporated as part of the cluster backup, but it will not completely replace the method described here.

For clarity the network backup procedures have been broken down into four parts. The first part deals with incremental daily backups across the network from all VAXs to one central VAX. The second part shows procedures for copying the network incremental backups from the central VAX disk to tape so that the structure of the backup save sets remains intact. The third part covers a procedure for doing periodic full backups from any VAX across the network to one central VAX. The fourth part covers the copying of the full backups to tape and listing the save sets. Note: none of the procedures here will replace the need for periodic system disk image backups. Again disk shadowing, when available, may replace system disk image backups on the cluster.

The following procedures have been used for the last year and a half or so for a VAX cluster [with two VAX 8600s, two 11/785s, two 11/750s with thirteen RA81 disks (500 MB each)]. One of these computers was selected to be the recipient of backup data for each individual operation. This is referred to as the SERVER node. Several machines can be SERVER nodes if many backups are being done. Non-clustered machines include two 11/750s, one 11/730, one 11/725, eight MicroVAX IIs, and thirteen MicroVAX Workstations. All machines are, of course, connected via DecNet/Ethernet.

## 2. INCREMENTAL NETWORK BACKUPS

The period for doing incremental backups may vary from computer to computer, but it has been our policy to do daily incremental backups with the /MODIFIED/SINCE=BACKUP qualifiers. This picks up all files modified since the last backup with the /RECORD qualifier. We do NOT use the /RECORD qualifier in any incremental backup commands. We reserve this qualifier only for full disk backups to minimize the amount of time to restore a disk if there is a hardware problem.

For our network incremental backups we have dedicated two of the cluster RA81 disks. One disk will store all the networked incrementals, and the second will store all the cluster users and system disk incrementals. The amount of disk space needed will depend on the number and size of the disks being backed up. These disks should have a directory structure to keep all the backup save sets organized. The directories should be owned by a nonprivileged account. The directory structure that we chose was one main directory on each backup disk, [REMOTEBAC], with subdirectories under it for each VAX node name (i.e., [REMOTEBAC.WSCAD1]) where its save sets could be stored. The common VAX backup account should be a new one with its default login to be the

main directory of the network backup disk (i.e., [REMOTEBAC]).

A backup account must be set up on each of the VAXs to be backed up. This account should have BYPASS privilege. It also must be proxied into the VAX backup account on the SERVER node or nodes. This proxy is done in the authorize file of the SERVER VAX. A command file, as shown in Figure 1a (modified for each specific system), would be submitted to one batch queue on each VAX to be backed up from the backup account, except the SERVER VAX. The SERVER VAX would have a file such as that shown in Figure 1b run as part of its nightly cleanup from the system account (necessary if the backup account is nonprivileged). Note that Figure 1b uses the second incremental backup disk because our SERVER VAX is on the VAX cluster and backs up all the cluster disks (many RA81s).

Once these procedures are set up, incremental backups will automatically be performed across the network onto a common disk. This is important since the save sets on the common disk can be archived with no impact on users. Further, this procedure, once set up, requires no person hours to make incremental backups to save sets on disk. Part 2 will show a procedure to archive these backup save sets to tape.

## 3. COPYING INCREMENTAL BACKUP SAVE SETS TO TAPE

Since we do not want our backup disks filling up with incremental backup save sets, we copy them to tape for storage. The policy that we use is that incremental backups will be stored on 20 rotating tape sets. This means a tape is written over only every 20 working days. This schedule is adequate since we do a full backup on each major disk on a four-week schedule (more on this in the network full backup section).

We do all of our backup copies to tape from a privileged captive account on a cluster VAX. This allows a less experienced person to do the actual work and maintains a degree of consistency. The captive account runs a simple command file as its default login.com as specified in the authorize file using the LGICMD parameter. The command file is shown in Figure 2a. This file will contain all the specific command options necessary to do all the tape and cleanup work for our backup files (most of this work is shown in Figure 2). This file can be modified easily as needs change.

This command file calls subordinate command files for each option which requires more than a few lines of code. This method is used to keep the structure simple. Most of the command files are relatively self-explanatory. The logout (lo at the end of each section is to ensure that the account is logged out after each procedure in case the operator steps away. In the copy subcommand files notice that the MOUNT/BLOCK_SIZE=xxxx option is used when the tape is mounted. This option is necessary for the tape copies to look like original save sets. The xxxx number is taken from the Record Format of the save set on the disk (a constant).

The REPLY subcommand is used when the copying of a save set spans more than one tape. When the operator is asked to mount a second tape, he does so, and then logs into the captive account again from the systems console and uses the REPLY subcommand with the number specified in the mount request received. Note that the actual REPLY command the VAX sees has the /BLANK_TAPE option on it. This option is needed for a proper save set continuation to take place.

This completes the network incremental backups. The person hours needed to accomplish this task will vary according to the number of backups that need to be written to tape. This procedure on our mix of computers takes about 3/4 person hour per day. This is a savings of almost a full person day in terms of computer logistics and the difference in media speed available on each VAX.

## 4. PERIODIC NETWORK FULL BACKUPS

As with incremental backups, the period for doing full backups may vary from computer to computer, but they should be coordinated with the incremental backup procedures. Our policy on full backups is to do each user and system disk on four-week intervals. This coincides with the rotation of 20 working days of incremental tape sets. The MicroVAXs and VAX stations at our site do NOT have network full backups done on them, because most have only a single disk functioning as both system and user, and the system disk must have a minimum system with DecNet software running to do the restore operation.

The disk used on the SERVER node to receive a network full backup should have enough space to accommodate the largest disk being backed up. We have dedicated one cluster RA81 (500 MB) disk because the largest disk backed up is an RA81. This allows for one network full backup per night. This disk is set up with the same upper level directory structure as the incremental disks and is owned by the remote backup account. This backup account must be proxied into the remote backup account on each client node to have a network full backup done on it.

On the SERVER node a data file such as that in Figure 3 must be set up. This has the date for a backup to take place, the disk name to be backed up, and the network node name of the computer to be backed up. The command file which drives the backup is dependent upon the syntax of this file. On the SERVER node a command file such as the one in Figure 4 must be submitted to a batch queue from the remote backup account. This command file is largely

self-explanatory. The procedure first modifies the text file to reflect a new backup date of four weeks out for every backup it does, except that it needs to be modified for leap year. It writes two command files to the remote node, the first to actually do the backup, and the second to submit the first one to the batch queue on the remote node. The backup in the first file uses a /RECORD option. The second command file is deleted after it is executed.

Once these network full backup procedures are set up, no further person hours are required to create full backups to save sets on disk. The next procedure will discuss archiving these save sets to tape.

## 5. COPYING NETWORK FULL BACKUPS TO TAPE

To copy the full backups to tape we use a subcommand file of the captive account described in Part 2. This subcommand file is shown in Figure 2e. The block size for the tape mount is used for the same reasons as in the incremental copy section. The backup file is deleted from the disk after copying it to tape to make room for the next full backup. Most full backups will require more than one tape for the copy. The procedure for the reply is the same as in the incremental copy section.

There is no listing file generated for the full backups because of the possible lack of room on the backup disk. Therefore, one more step is necessary to finish the full backup procedure, a listing of the full backup tapes. This procedure is performed by using the captive account subcommand FBLIST (see Figure 2f). This file does only a listing of the backup save set tapes and saves it in the directory defined as SYS$BACKUP for future reference. In addition, reading through the tape ensures that the copy is error-free.

The entire network full backup procedure is now complete. The procedure for a mostly full large disk such as an RA81 (500 MB) will require about one and a half person hours for both the copy and the list procedures. This represents a substantial savings over the non-network method, and minimizes the impact on users since their files are accessible during our prime-time ordinary activity.

## 6. SUMMARY

The above procedures have been set up for DEC VAX computers and have worked very well for us. At some time in the future we plan to expand this method to some other types of computers in our organization.

The exact command files used and the site-specific elements within the files are not the important things to note. Modify them as you wish. All systems and environments are different. The important idea is that a network procedure for backups can and has been developed and it does save expensive person hours.

```
$!
$!  REMOTEINC.COM
$!
$!  to be submitted on each remote node to be incrementally backed up
$!
$ set verify
$ set noon
$!
$!  get local node name
$!
$ node = F$GETSYI("NODENAME")
$!
$!      this file is to do incremental backups of all files
$!      modified or created since the last image backup
$!      written 10-22-84 by d.darkangelo
$!
$!  resubmit itself tomorrow for scheduled time (3:15am)
$!  - needed if many network backups are going  -  only
$!  about 4 or 5 can go at the same time to the same vax
$!
$ submit/que=sys$batch/after="tomorrow+03:15:00" sys$manager:remoteinc.com
$!
$!  clean up the disks to be backed up (only one disk on this system)
$!
$ purge dua0:[000000...]
$!
$! dua0: backup to directory on ASLVAX (common backup node)
$!
$ backup/ignore=interlock/list=aslvax::[.'node]'node.log-
/modified/since=backup dua0:[000000...]*.*;*/EXCLUDE=(*.SYS) -
aslvax::[.'node]'node.bck/save_set
$!
$!   clean up the specific directory on the common backup disk
$!
$ purge aslvax::[.'node]
$!
$!  NOTHING TO DO WITH BACKUP
$!       this is just a convenent place for this function
$!
$!  get the lastest network data base from the master network node
$!     neede decnet 4.2 for this
$!
$ ncp = "$ncp"
$ ncp copy known nodes from aslvax:: using volatile to both
$!
```

**Figure 1a.**

```
$!
$!  NIGHTBACK.COM
$!
$!  submitted on a cluster vax from the system account
$!      to do incremental backups
$!
$!      written by D.DARKANGELO
$!
$ assign nl: sys$print
$ set proc/prio=5
$!
$ set noon
$!
$!  set file ownership to the remotebackup account
$!
$ set uic [333,2]
$!
$!  keep 20 copies for the log files to correspond to the 20
$!    rotating backup tapes
$!
$ purge/keep=20 sys$disk8:[remotebac.cluster]*.log
$!
$!
$!      Reassign sys$output so the list of saved files goes to a log file
$!
$ assign sys$disk8:[remotebac.cluster]temp.log sys$output
$!
$ set noon
$!
$!      User disk #1
$!
$ backup/log -
sys$disk1:[*...]/modified/since=backup -
sys$disk8:[remotebac.cluster]userdisk1.bck/save
$!
$!      User disk #2
$!
$ backup/log -
sys$disk2:[*...]/modified/since=backup -
sys$disk8:[remotebac.cluster]userdisk2.bck/save
$!
$!  fill in all the other user disks on the cluster
$!  mine have been omitted for space reasons in this example
$!
$!
$!      System disk - excluding *.SYS files
$!
$ backup/log -
sys$sysdevice:[*...]/modified/since=backup-
/exclude=([*...]*.sys,[*.syscommon...]*.*;*) -
sys$disk8:[remotebac.cluster]sysdisk.bck/save
$!
$!      All done - close up shop
$!
$ deassign sys$output
$ rename sys$disk8:[remotebac.cluster]temp.log -
sys$disk8:[remotebac.cluster]cluster.log
$!
$ exit
```

**Figure 1b.**

```
$!
$!  ALLBACKUPS.COM
$!
$!   captive account backup command file
$!     written by D. DARKANGELO
$!
$ set noon
$ set prot=w:re/default
$ write sys$output ""
$ write sys$output "Good morning"
$ write sys$output ""
$ write sys$output "Available commands:  "
$ write sys$output " CLUSTER_LIST, REPLY, SMALL_COPY,"
$ write sys$output " CTSVAX_COPY, CLUSTER_COPY, FULLBACK_COPY"
$ write sys$output ""
$!
$!  command loop
$!
$GET_COMMAND:
$ inquire command "Command"
$ if command .eqs. "CLUSTER_LIST" then goto CLUSTER_LIST
$ if command .eqs. "REPLY" then goto REPLY
$ if command .eqs. "SMALL_COPY" then goto SMALL_COPY
$ if command .eqs. "CTSVAX_COPY" then goto CTSVAX_COPY
$ if command .eqs. "CLUSTER_COPY" then goto CLUSTER_COPY
$ if command .eqs. "FULLBACK_COPY" then goto FULLBACK_COPY
$ goto GET_COMMAND
$!
$!  used to make a listing file of a fullbackup of a cluster disk
$!
$CLUSTER_LIST:
$ define/user sys$input sys$command
$ @sys$backup:fblist
$ lo
$!
$!  used when a continuation tape is needed
$!
$REPLY:
$ write sys$output ""
$ write sys$output " You must be logged into the system console to use this "
$ write sys$output ""
$ define/user sys$input sys$command
$ inquire num "Request number replying to "
$ reply/blank_tape='num
$ lo
$!
$!  all small disk backups are saved on the same daily tape
$!  - 20 days worth of tapes on a rotational basis
$!
$SMALL_COPY:
$ define/user sys$input sys$command
$ @sys$backup:smallcopy
$ lo
$!
$!  used to copy the CTSVAX backups to tape (1600bpi) for use on their
$!  computer system.  the LIST process is done on the CTSVAX.
$!
$CTSVAX_COPY:
$ define/user sys$input sys$command
$ @sys$backup:CTSVAXCOP
$ lo
$!
$!  used to copy the cluster backups to tape (6250bpi)
$!
$CLUSTER_COPY:
$ define/user sys$input sys$command
$ @sys$backup:CLUSTCOPY
$ lo
$!
$!   used to copy all fullbackups from disk to tape
$!
$FULLBACK_COPY:
$ define/user sys$input sys$command
$ @sys$backup:FULLCOPY
$ lo
$!
```

**Figure 2a.**

```
$!
$!  CLUSTCOPY.COM
$!
$!   written by D. DARKANGELO
$!
$!  if there is any errors in this procedure exit
$!
$ on error then exit
$!
$! set all backup logs to world read, execute
$!  so that non privileged users can see what files
$!  may be recovered
$!
$ set prot=w:re sys$disk8:[remotebac...]*.log;*
$ set prot=w:re sys$disk6:[remotebac...]*.log;*
$!
$!      to copy save sets to tape for the cluster
$!
$ write sys$output "   CLUSTER COPY ROUTINE "
$ write sys$output ""
$!
$!  this is for the weekday holidays which will generate multiple
$!   versions of the incremental backups
$!
$ write sys$output " If there is more than one version of the .bck file "
$ write sys$output " you will need to write to more than one tape.  "
$!
$!  set default to the directory from which the backup files are to be copied
$!
$ set def sys$disk8:[remotebac.cluster]
$!
$!  give the user a directory list from which to answer questions
$!
$START:
$ dir/size/date *.bck
$!
$! get tape transport number which is being used
$!
$GET_TAPENUM:
$ inquire tapenum "What tape transport are you using (MFA0 or MFA1) "
$ if tapenum .eqs. "" then goto get_tapenum
$!
$! can only write to tape if a write ring is in place
$!
$ inquire ans  " Is a tape with write ring hung "
$ if ans .eqs. "YES" then goto next1
$ exit
$NEXT1:
$ inquire version "Lowest version of .bck files. [exit] "
$ if version .eqs. "" then exit
$ init 'tapenum':/density=6250 backup
$!
$!  the block size is important to have the tape look like the
$!  true backup saves sets
$!
$ mount 'tapenum':/block=32256/density=6250 backup
$!
$! write a directory file
$!
$ assign/user test.out sys$output
$ dir/size *.bck
$ open/read input_file test.out
$ lines = 0
$!
$!  skip over the header information
$!
$HEADER:
$ read input_file scan
$ lines = lines + 1
$ if lines .eq. 3 then goto WORK
$ goto HEADER
$!
$!  find a backup file of the version number specified
$!
$WORK:
$ read/error=close_up input_file input
$ lengt = f$locate(" ",input)
$ file_version = f$extract(lengt-1,1,input)
$ if file_version .nes. version then goto work
$!
$!  get backup file name and copy it to tape
$!
$ file = f$extract(0,lengt,input)
$ copy/prot=(g,w)/log 'file 'tapenum':*
$!
$! delete backup file after complete copy
$!
$ del/log 'file
$!
$!  continue until all of the specified version is done
$!
$ goto WORK
$!
$!  clean up directory
$!
$CLOSE_UP:
$ write sys$output "ALL DONE"
$ close input_file
$ delete test.out;*
$ dismount 'tapenum':
$ write sys$output ""
$ write sys$output "To continue this procedure for a new version"
$ write sys$output "  a new tape must be hung. "
$goto START
```

**Figure 2b.**

```
$!
$!   SMALLCOPY .COM
$!
$!     written by D. DARKANGELO
$!
$!      to copy save sets to tape for all small disks
$!      to one tape set
$!
$! get tape transport number
$!
$ get_tapenum:
$ inquire tapenum "What tape transport are you using (MFA0 or MFA1) "
$ if tapenum .eqs. "" then goto get_tapenum
$!
$ inquire ans  " Is a tape with write ring hung "
$ if ans .eqs. "YES" then goto next1
$ exit
$NEXT1:
$ init 'tapenum':/density=6250 backup
$!
$!  block size is important for the copies to look like backup save sets
$!
$ mount 'tapenum':/block=32256/density=6250 backup
$!
$!  set default to directory from which the backup files are to be copied
$!
$ set def sys$disk6:[remotebac.imsvax]
$ copy/prot=(g,w) *.* 'tapenum':*
$!
$ set def sys$disk6:[remotebac.ccpvax]
$ copy/prot=(g,w) *.* 'tapenum':*
$!
$!  and so on for all the small disks
$!  program shortened for this artical
$!
$ dismount 'tapenum':
```

**Figure 2c.**

```
$!
$!  CTSVAXCOP.COM
$!
$!  written by D. DARKANGELO
$!
$ on error then exit
$!
$!      to copy save sets to tape for
$!      CTSVAX DRA1, DRA2, DRA0
$!
$ write sys$output "   CTSVAX COPY ROUTINE "
$ write sys$output ""
$!
$!  this can be caused by a weekday holiday
$!
$ write sys$output " If there is more than one version of the .bck file "
$ write sys$output " you will need to write to more than one tape.  "
$!
$! set default to the directory from which the backups files are to be copied
$!
$ set def sys$disk6:[remotebac.ctsvax]
$!
$! give the user a  directory list from which to answer questions
$!
$START:
$ dir/size/date *.bck
$!
$! get tape transport number which is being used
$!
$ get_tapenum:
$ inquire tapenum "What tape transport are you using (MFA0 or MFA1 ) "
$ if tapenum .eqs. "" then goto get_tapenum
$!
$!  can't write to a tape without a write ring in place
$!
$ inquire ans  " Is a tape with write ring hung "
$ if ans .eqs. "YES" then goto next1
$ exit
$!
$!  get specific backup files version to be copied
$!
$NEXT1:
$ inquire version "Lowest version of .bck files. [exit] "
$ if version .eqs. "" then exit
$ init 'tapenum':/density=1600 backup
$!
$! block size is very important to have copy tapes look like
$!  backup save set tapes
$!
$ mount 'tapenum':/block=32256/density=1600 backup
$!
$!  write a directory file
$!
$ assign/user test.out sys$output
$ dir/size *.bck
$ open/read input_file test.out
$ lines = 0
$!
$!  skip over the header info
$!
$HEADER:
$ read input_file scan
$ lines = lines + 1
$ if lines .eq. 3 then goto work
$ goto HEADER
$!
$!  get a backup file name for the specified version
$!
$WORK:
$ read/error=close_up input_file input
$ lengt = f$locate(" ",input)
$ file_version = f$extract(lengt-1,1,input)
$ if file_version .nes. version then goto work
$ file = f$extract(0,lengt,input)
$!
$!  copy save set to tape
$!
$ copy/prot=(g,w)/log 'file 'tapenum':*
$!
$!  delete save set after complete copy
$!
$ del/log 'file
$ goto WORK
$!
$!  clean up directory
$!
$CLOSE_UP:
$ write sys$output "ALL DONE"
$ close input_file
$ delete test.out;*
$ dismount 'tapenum':
$ write sys$output ""
$ write sys$output "To continue this procedure for a new version of backups,"
$ write sys$output " a new tape must be hung. "
$goto START
```

**Figure 2d.**

```
$!
$!  FULLCOPY.COM
$!
$!   written by D. DARKANGELO
$!
$!  on any error exit this procedure
$!
$ on error then exit
$!
$!      to copy save sets to tape for
$!      full disk save sets
$!
$ write sys$output "   FULLBACK COPY ROUTINE "
$ write sys$output ""
$!
$! set default to directory from which the fullbackup file is to be copied
$!
$ set def sys$disk7:[remotebac]
$!
$! give the user a directory to answer questions from
$!
$START:
$ dir/size/date *.bck
$!
$! get tape transport number
$!
$ get_tapenum:
$ inquire tapenum " What tape transport are you using (MFA0 or MFA1) "
$ if tapenum .eqs. "" then goto get_tapenum
$!
$!  can't write to a tape without a backup write ring in place
$!
$ inquire ans  " Is a tape with write ring hung "
$ if ans .eqs. "YES" then goto next1
$ exit
$!
$!  get specific fullbackup version number to copy
$!
$NEXT1:
$ inquire FILE "FULL file name of .bck file including version. [exit] "
$ if FILE .eqs. "" then exit
$!
$!  ask what tape density to use
$!
$ inquire dens "Is this for a 1600BPI tape yes or no [6250] "
$ if dens .eqs. "YES" then goto init1600
$ init 'tapenum':/density=6250 backup
$!
$! block size is important for the copied files to look like
$!  backup save sets
$!
$ mount 'tapenum':/block=32256/density=6250 backup
$ goto jump1
$INIT1600:
$ init 'tapenum':/density=1600 backup
$!
$! block size is important for the copied files to look like
$!  backup save sets
$!
$ mount 'tapenum':/block=32256/density=1600 backup
$!
$!  copy the fullbackup files
$!
$JUMP1:
$ copy/prot=(g,w)/log 'file 'tapenum':*
$ del/log 'file
$!
$!  clean up
$!
$CLOSE_UP:
$ write sys$output "ALL DONE"
$ dismount 'tapenum':
$ write sys$output ""
$ write sys$output "To continue this procedure, a new tape must be hung. "
$goto START
```

**Figure 2e.**

```
$!
$!      FBLIST.COM
$!
$!      Performs listing on full magtape backup
$!
$!      Ask user which disk  backup is to be listed
$!
$ save_verify = 'f$verify("NO")'
$!
$ get_disknum:
$ inquire disknum "Enter disk number to be listed (0,1,2,3,4,5)"
$ if disknum .gt. 5 .or. disknum .lt. 0 then goto get_disknum
$!
$! get tape transport number which is being used
$!
$ get_tapenum:
$ inquire tapenum "What tape transport are you using (MFA0 or MFA1 ) "
$ if tapenum .eqs. "" then goto get_tapenum
$!
$!      Make sure the tape is ready to go
$!
$ inquire rdy "Physically hang first tape, put on-line and hit RETURN"
$!
$!      Determine log filename  ( standard for my site )
$!
$ log_file := SYS$BACKUP:FULLUSER'disknum'.LOG
$ if disknum .eq. 0 then log_file := SYS$BACKUP:SYSTEM.LOG
$ container_file := FULLUSER'disknum'.BCK
$ if disknum .eq. 0 then container_file := SYSTEM.BCK
$!
$!      Mount the tape
$!
$ mount 'tapenum':/foreign
$!
$!      Perform listing of full backup tape
$!
$ set noon
$ backup/list='log_file' 'tapenum':'container_file'
$!
$!      All done - close up shop
$!
$ dismount 'tapenum':
$!
$ if save_verify then set verify
```

**Figure 2f.**

```
31-MAR-1986 SYS$DISK1: ASLVAX
17-MAR-1986 SYS$DISK2: ASLVAX
24-MAR-1986 SYS$DISK3: ASLVAX
27-MAR-1986 SYS$DISK4: ASLVAX
10-MAR-1986 SYS$DISK5: ASLVAX
 6-MAR-1986 SYS$DISK1: CTSVAX
13-MAR-1986 SYS$DISK2: CTSVAX
 5-MAR-1986 SPS$DISK1: ASLVAX
```

**Figure 3.**

```
$!
$!  BFULLBACK.COM
$!
$!*****************************************************************
$!
$!      This file is used to automate the full backup procedures
$!      on the cluster. It will be run in batch each night.
$!
$!      Written by      John M. Spaeth  (26-Jun-1985)
$!
$!*****************************************************************
$!
$ ASSIGN NL: SYS$PRINT
$ SET PROC/PRIO=5
$!
$!  Resubmit itself for tomorrow early
$!
$ submit/noprint/que=SYS$BATCH/after="tomorrow+00:01:00" -
SYS$DISK6:[remotebac]bfullback.com
$!
$!      Main routine to do find the disk and do the backup (if any)
$!
$       OPEN/READ INPUT_FILE SYS$DISK6:[remotebac]FULLBACKS.TXT
$       OPEN/WRITE OUTPUT_FILE SYS$DISK6:[remotebac]FULLBACKS.TMP
$       SEEIT = F$EXTRACT(0,11,F$TIME())
$       DISK = ""
$ LOOP:
$       READ/ERROR=DONE INPUT_FILE INPUT
$       SUBDATE = F$EXTRACT(0,11,INPUT)
$!
$!      Rewrite text data file for disks not needing to be backed up
$!
$       OUT = F$EXTRACT(12,10,INPUT)
$       DSKNUM = F$EXTRACT(20,1,INPUT)
$       VAXNAM = F$EXTRACT(23,6,INPUT)
$       IF F$CVTIME(,,"DATE") .EQS. F$CVTIME(SUBDATE,,"DATE") -
THEN GOTO HERE_WE_GO
$!
$       WRITE OUTPUT_FILE "''SUBDATE' ''OUT' ''VAXNAM'"
$       GOTO LOOP
$!
$!      Add 28 days to get new backup date
$!      Rewrite text data file for disks needing to be backed up and
$!      set up symbols used to do the backup
$!
$ HERE_WE_GO:
$       SUBDATE1 = F$CVTIME("+28-","ABSOLUTE","DATE")
$!
$!      ADD LEADING BLANK TO DATE FOR PROPER DATA FILE FORMAT
$!
$       CKIT = F$LOCATE("-",SUBDATE1)
$       IF CKIT .EQ. 1 THEN SUBDATE1 = " "+SUBDATE1
$       WRITE OUTPUT_FILE "''SUBDATE1' ''OUT' ''VAXNAM'"
$       DISK = OUT
$       DSKOUT = DSKNUM
$       VAXOUT = VAXNAM
$       GOTO LOOP
$!
$!      Close all opened files and cleanup newly created files
$!
$ DONE:
$       CLOSE OUTPUT_FILE
$       CLOSE INPUT_FILE
$       RENAME SYS$DISK6:[remotebac]FULLBACKS.TMP SYS$DISK6:[remotebac]*.TXT
$       PURGE SYS$DISK6:[remotebac]FULLBACKS.TXT
$!
$!      If there is a backup to do, go off and do it
$!
$       IF DISK .EQS. "" THEN EXIT
$!
$!  Write network file to do the remote fullbackup
$!
$ open/write data 'vaxout'::netfulbac.com
$ write data "$ set verify"
$ write data "$ set proc/prio=5"
$ write data "$ backup/record ''disk'[*...]*.* -"
$ write data "aslvax::SYS$DISK7:[remotebac]fulluser''dskout'.bck/save"
$ write data "$!"
$ close data
$!
$!  Write network file to submit NETFULBAC.COM to remote computer queue
$!  The /AFTER is to delay the backup for its scheduled time
$!
$ open/write data2 'vaxout'::netfulcon.com
$ write data2 "$ define/user sys$output sys$net"
$ write data2 "$ submit/que=sys$batch/delete/after=04:00:00 netfulbac.com"
$ close data2
$!
$!  Execute NETFULCON.COM across the network and delete it
$!
$ type 'vaxout'::"task=netfulcon"
$ delete 'vaxout'::netfulcon.com;
$       EXIT
```

**Figure 4.**

# POSTER PAPERS

# Creating Common Spooled Resources in a VAX-Cluster

Bob Rasmussen and Bob Nestor
Teledyne-Geotech
Garland, Texas

ABSTRACT

This paper describes a technique for creating and managing spooled output queues that are transparently available, cluster-wide, to all user in a cluster. The technique does not rely on DECnet for remote access to a spooled device residing on another node in the Cluster, and eliminates the need for having the VAX to which the output device is connected operational within the Cluster. No custom software or hardware is required; all is accomplished using the standard VMS Queue Manager and spooler software.

In March of 1979 we replaced our out-dated second generation CDC mainframe with a new state-of-the-art VAX-11/780, one of the first to be delivered by DEC in the Southwest. At the time it was an impressive configuration; two 67-Megabyte RM03's, two dual-density TE16's and two megabytes of main memory. This over-configuration was deemed necessary since the system was to support all Company computing activities including Accounting, Inventory, On-line shop floor control, Engineering and R&D efforts. Since Fortran was the only native-mode software offered in 1979, most of our users were supported with compatability mode software running either under the RSX AME or the RSTS/E AME (ROSS/V).

Over the intervening years we have added to the system both in terms of hardware and user load. In March of 1986 we made our latest configuration change by adding a second VAX-11/780 CPU and forming a VAX-Cluster. Our current Computer Center configuration includes two 780's, an HSC-50 with an RA80 common system disk and three RA81 user disks in a Volume Set, and approximately 120 user terminals, printers and plotters scattered throughout our complex.

We have, unfortunately, not blazed any new ground in developing upward-compatible, configuration independent, transportable software in our shop. Most of our user developed software is tied to various system configurations which have existed in the past. And reguardless of the availability of native-mode VAX software today, most of our users continue to run the out-dated, compatibility mode software we started with 7 years ago. The most glaring example of this is our On-line shop floor control package written in Basic-Plus which crawls run RSTS/E emulation. In addition, despite numerious warnings to the contrary, many users have imbedded physical device names into their programs and command files. This occurs most frequently with our spooled output devices which includes a mixture of line printers, plotters, and hard-copy devices such as LA100s and LA120s. With the introduction of the VAX-Cluster, device names are node-specific not cluster-wide, i.e. the terminal port TTA6 may exist on all cluster nodes, but only one is physically connected to the target output device. Our challenge then was to construct a VAX-Cluster that would have minimum impact on existing user software and procedures.

First, we decided to put all RSTS/E and COBOL accounting functions on a single processor. That left all Engineering and R&D activities for the second processor where we hung all of the special devices including spooled terminals. The special devices include an Array Processor, various plotters and the DECnet interconnect to a PDP-11/24 which is dedicated to Word Processing and A/D conversion activities.

We also decided to run both systems from a Common System Disk. Neither system has any dedicated disk drives, only dedicated tape units primarily due to the inability of the HSC50 to support the older technology drives. In going to the Common System disk we eliminated the need to maintain two copies of the VMS Software which is quite large. After examining the proposed structure of a Common System Disk, we determined that a single RA80 would provide sufficient room for all VMS components for both systems. Since this runs off the HSC-50, we eliminated all of the older RH Massbus interfaces to disks.

In addition to running the Common System Disk, we chose to continue running the RA81 user volumes as a Volume Set. This gives us the freedom to add new user disk storage with no user impact in the future. (Apparently we are one of the few VAX sites to actually use Volume Sets since we continue to find "glitches" in various VAX components that all relate back to Volume Sets.)

Most user terminals are connected through a Gandalf Switch which we have had in-place now for a number of years. This gives users an opportunity to select the processor they want to connect to during the LOGIN procedure, and gives us the ability to switch all users to either one of the processors in the

event of a processor failure or scheduled down-time.

Our only remaining problem was that of the spooled output devices. Any user on either system had to have access to any of the spooled output devices. Also, in the event of a processor failure, we needed to insure that user programs and procedures would run correctly from either processor with respect to the use of output devices. The most obvious place to start solving this problem was with a Cluster-Wide Queue Manager which makes all print and batch queues available to all users reguardless of the node they are currently running on.

By defining Logical Queue Names in the Queue Manager and associating them with physical output devices in the Cluster we thought we had the problem solved. Unfortunately, this solution was only effective for users who queued their output to the device via the PRINT DCL command. Programs that tried to use transparent spooling to the devices would get RMS OPEN or CLOSE errors since VMS does not allow output to be directed to a Queue; it must be directed to a device which is in turn spooled by the Queue Manager.

On subsequent reading of the VMS documentation describing the Queue Manager, it appeared that our problems would be solved by defined a Generic Queue of the same name as our Logical Queue Names and defining a Phyisical Terminal or Printer Queue for each actual spooled device. The generic queue would be de-spooled through the appropiate physical queue. Unfortunately these definitions were still not Cluster-wide. The Queues themselves were available cluster-wide, but spooled device names were still node specific. Since we needed cluster-wide access to spooled devices, we investigated DECnet.

Cluster-wide spooled device access seemed to be solved with DECnet and Proxy LOGINs on both nodes in the Cluster. This permits a user to remotely access, via DECnet, a device on any other Cluster node. With Proxy LOGIN the user was not required to enter any access information to get to the remote device. And, since we ran a Common System Disk with a single User Authorization File, all users have accounts on both systems. However, our experience with DECnet in the Cluster brought out the following problems:

- DECnet introduced another user session on the remote node whenever a remote device was being accessed. Typically this is the FAL process run on behalf of the local user.
- Most output requests were in the form of PRINT commands which typically involve queuing a disk file for printing to a particular device or queue. With the Cluster-wide Queue Manager this is no problem. But, user programs and procedures which take advantage of VMS transparent device spooling and output directly to the device can only run on the processor which "owns" the device unless the device name is fully qualified with a DECnet node name. For native-mode programs and command procedures this was not a major problem. Compatibility mode programs, some layered DEC software and some third party software would not operate in this mode.
- Using DECnet to access the device remotely requires that DECnet be operational. Since this is only true if the target processor is also operational, we could only use the DECnet technique if both processors in the Cluster were on-line. If either were in a failure mode, most spooled output devces were unavailable and most user programs could not be executed on the remaining node without modifications. This seemed to be in direct conflict with the justifications for use of a VAX-Cluster.

One possible solution was to dedicate a given number of terminal ports to output spooling functions. Both processors in the Cluster would have to reserve the same terminal ports even though only one processor would actually service the device. This was necessary to maintain a Cluster-wide naming convention on spooled devices. This idea was quickly rejected since it would dedicate ports on one processor that would never be used, a waste of important resources in this case DZ11's. Also, the concept would leave no room for future expansion unless we dedicated more ports that we actually needed today.

Finally, it occured to us that we could create non-existant terminal ports on both systems that could serve as pseudo-queues. These pseudo-device queues could be mapped to real devices on either of the processors with the "/ON" qualifier in the Queue Manager. A common set of pseudo-devices would be created on each processor in the Cluster and each would be spooled to the actual output device by the Cluster-wide Queue Manager. These devices and queues would exist even if one of the processors was in a failure mode, however the processor to which the device were physically connected must be operational for the output to occur. User programs and procedures would continue to run but their output would be collected in the appropiate device queues for later printing when the processor is brought back on-line.

Terminal pseudo-devices were selected rather than pseudo line printers since both nodes already contained the terminal driver but only one contained the line printer driver. In practice the line printer turned out to be a special case since it is already known to the system, and user, under many different names (LP, LP0, LPA0, SYS$PRINT, etc).

The only problem we have experienced with our queue setup is with programs which conditionalize their output format on the outut device characteristics. For the most part this programming practice has been eliminated from DEC VMS software; the DEFINE FORMS facility is used to define output layouts. This solved the problem some users experienced where their line printer output was sized according to the form style in the printer at the time the output was generated rather than being formatted by the form style to which the output was to be printed on. Unfortunately, some third-party software, namely ROSS/V, still intergoates the output device for formatting information. Since the pseudo-terminal devices "TTQn" are not really present in the system, VMS shows them to be off-line and will therefore not allow you to set any characteristics for them. The actual device characteristics that will be returned to a user program are the default terminal

characteristics defined in SYSGEN. To fake ROSS/V into seeing 132 column printers, we set the default terminal characteristics to be those of an LA120.

Basically the technique requires the use of a Cluster-wide Queue Manager with a common definition of individual queue attributes. Since we run a Common System Disk this part of the Queue startup is contained in a file SYS$COMMON:[SYSMGR]STARTQUE.COM which is invoked by each processor from the SYSTARTUP.COM file on boot. This part of the queue startup:

1) Starts the Cluster-wide Queue Manager for the invoking processor and points to the the Cluster-wide Queue File.

2) Creates all pseudo-terminals that will be mapped to actual spooled devices. These definitions occur on both processors in the Cluster. We chose to name the devices "TTQ0" through "TTQ6" (on our system). There is lots of room for additional definitions for future spooled devices.

3) Each pseduo-terminal is set spooled to itself, i.e.
"SET DEVICE/SPOOLED=TTQ0: TTQ0:"
This was done so that users may use the "TTQ" device names in their programs for transparent output spooling.

4) Each TTQ Queue is tested to see if it exists. If not, it is initialized (but not started) with appropiate attributes such as /FORM, /DEFAULT etc. This is also the point at which we define the actual terminal or printer in the cluster that will process the output requests through the use of the /ON qualifier.

5) Other queue attributes are also set such as queue scheduling.

6) A system-wide logical name is created and assigned to the TTQ pseudo-terminal. We are still trying to encourage our users to use logical names for the output devices rather than physical names, but we must support both.

7) Finally, we also define the batch queues that will be required in the Cluster. This common command procedure then invokes the node specific queue startup procedure which is kept in SYS$SYSROOT:[SYSMGR]QUESTART.COM.

The node specific Queue startup is responsible for setting up the actual physical device characteristics and starting the Queue for that device. This may involve as much as:

1) Loading any custom device drivers for special spooled devices. We have one, a Versatec printer/plotter.

2) Setting terminal characteristics for physical terminal ports that will handle a spooled output device. This includes such things as terminal speed, format control (TAB, FORM, WRAP, WIDTH, etc).

3) Setting the physical device spooled to the corresponding pseudo-terminal device. This is not really required but it intercepts those user requests for output the the "real" physical device.

4) Starting the Queue Manager to process output requests on the pseudo-terminal. This begins the de-queue operation.

5) Starting the node-specific batch queues. On our Cluster we run a SYS$BATCH on both nodes, so a system-wide logical is assigned to the appropiate batch queue at this point.

6) On the system that does not have a line printer we also assign system-wide logicals for SYS$PRINT, LP LP0, and LPA0 to point the the appropiate TTQ pseudo-terminal. This intercepts all line printer output on the system and re-directs it to the proper line printer queue.

All of this may seem to be a long way to go just to get output on a printer, but our technique does have certain advantages.
First, users are assured that the device names "TTQn" are fixed in our configuration. We will not re-define these in the future, but may add additional ones as needed. Those users that write programs to use VMS transparent spooled may continue to do so. This also solves the problem of not being able to open and output file on a Queue, which allows Datatrieve users to output results "directly" to the spooled device. Second, since our logical names for the pseudo-terminals in fact point the the pseudo-terminals themselves, the /DEV and /QUE qualifiers in the PRINT command are identical (as they were in earlier VMS releases). Third, User programs are independent of the processor they run on and may continue to run creating output for physical devices that are currently not available in the Cluster. Their output will be intercepted by the VMS Queue Manager for later printing when the device is available. Finally, we no longer run background DECnet processes just to get output to a physical device. DECnet is rarely used between the Cluster nodes.

One last set of changes was required in ROSS/V V3.4 (RSTS/E-VAX) to permit Basic-Plus program access to all spooled devices. This was done during the ROSS/V Sysgen where we defined fixed keyboards for each of the TTQ devices in the VMS system. The line printer was assigned the VMS name of TTQ0 to create a version of ROSS/V that was node independent, i.e. could be run from either node without change. Basic-Plus programmers can now access the spooled devices as RSTS/E devices "LP0:" and "KB1:" through "KB6:". All other RSTS/E keyboards are left to "float" in the system; we allow ROSS to assign names to the devices as user enter the ROSS emulator from their VMS terminal.

-- File SYS$COMMON:[SYSMGR]STARTQUE.COM --
(This file is common to all nodes in the Cluster.)

```
$ !
$ ! Initialize cluster-wide queues
$ !
$       SET NOON
$       START/QUEUE/MANAGER SYS$COMMON:[SYSEXE]JBCSYSQUE.DAT
$ !
$ ! Set up pseudo terminals for cluster print devices
$ !
$       RUN SYS$SYSTEM:SYSGEN
CONN TTQ0/NOADAPT        ! LPA0 on       BETAX::LPA0:
CONN TTQ1/NOADAPT        ! PLOTTER on    ALFAX::TTA6:
CONN TTQ2/NOADAPT        ! LA100 on      ALFAX::TTB0:
CONN TTQ3/NOADAPT        ! LA120 on      ALFAX::TTE3:
CONN TTQ4/NOADAPT        ! GEOPRINT on   ALFAX::TTE4:
CONN TTQ5/NOADAPT        ! BLDG5 on      ALFAX::TTE5:
CONN TTQ6/NOADAPT        ! VERSATEC on   ALFAX::LWA0:
EXIT
$ !
$ ! Set and spool pseudo print devices, assign system-wide
$ !   logical names, and initialize the cluster-wide queue
$ !   if it doesn't exist
$ ! Define the forms types available to users on the Cluster
$ !
$       DEFINE/FORM DEFAULT 0/MARGIN=(BOTTOM=2)/NOTRUNC/NOWRAP
$       DEFINE/FORM ONE 1/MARGIN=(BOTTOM=2)
$       DEFINE/FORM TWO 2/MARGIN=(BOTTOM=2)
$       DEFINE/FORM THREE 3/MARGIN=(BOTTOM=2)
$       DEFINE/FORM FOUR 4/MARGIN=(BOTTOM=2)
$       DEFINE/FORM FIVE 5/MARGIN=(BOTTOM=2)
$       DEFINE/FORM SIX 6/MARGIN=(BOTTOM=2)
$       DEFINE/FORM SEVEN 7/LENGTH=42/MARGIN=(BOTTOM=0)
$       DEFINE/FORM EIGHT 8/MARGIN=(BOTTOM=2)
$       DEFINE/FORM NINE 9/MARGIN=(BOTTOM=2)
$       DEFINE/FORM COMPRESS 11/MAR=(BOT=2)/WID=220/NOTRUNCATE -
                /DESC="COMPRESSED PRINTINT ON TTA3:"
$       DEFINE/FORM TEST 99/DESC="TEST PRINT QUE"
$ !
$ ! Cluster-wide system line printer (computer-room)
$ !
$       SET DEVICE/SPOOLED=(TTQ0:,DUA1:)     TTQ0:
$       ASSIGN/USER NLA0: SYS$OUTPUT:
$       SHOW QUE TTQ0:
$       IF .NOT $STATUS THEN INIT/QUEUE/DEFAULT=FLAG-
                /ON=BETAX::LPA0:  TTQ0:
$       SET QUEUE/SCH=NOSIZE        TTQ0:
$ !
$ ! HP-Plotter (computer-room)
$ !
$       SET DEVICE/SPOOLED=(TTQ1:,DUA1:)     TTQ1:
$       ASSIGN/USER NLA0: SYS$OUTPUT:
$       SHOW QUE TTQ1:
$       IF .NOT $STATUS INIT/QUEUE/NOENABLE_GENERIC-
                /DEFAULT=(NOFLAG,NOFEED)/TERM-
                /ON=ALFAX::TTA6:            TTQ1:
$       DEFINE/SYSTEM/NOLOG         PLOTTER: TTQ1:
$       SET QUE/SCH=NOSIZE TTQ1:
$ !
$ ! LA100 (across from computer-room)
$ !
$       SET DEVICE/SPOOLED=(TTQ2:,DUA1:)     TTQ2:
$       ASSIGN/USER NLA0: SYS$OUTPUT:
$       SHOW QUE TTQ2:
$       IF .NOT $STATUS INIT/QUEUE/NOENABLE_GENERIC-
                /DEFAULT=NOFLAG/BLOCK_LIM=200/TERM-
                /ON=ALFAX::TTB0:            TTQ2:
$       DEFINE/SYSTEM/NOLOG         LA100:   TTQ2:
$       SET QUE/SCH=NOSIZE TTQ2:
$ !
$ ! LA120 (user terminal room)
$ !
$       SET DEVICE/SPOOLED=(TTQ3:,DUA1:)     TTQ3:
$       ASSIGN/USER NLA0: SYS$OUTPUT:
$       SHOW QUE TTQ3:
$       IF .NOT $STATUS INIT/QUEUE/NOENABLE_GENERIC-
                /DEFAULT=NOFLAG/TERM-
                /ON=ALFAX::TTE3:            TTQ3:
$       DEFINE/SYSTEM/NOLOG         LA120:   TTQ3:
$       SET QUE/SCH=NOSIZE TTQ3:
$ !
$ ! Geophysical system printer
$ !
$       SET DEVICE/SPOOLED=(TTQ4:,DUA1:)     TTQ4:
$       ASSIGN/USER NLA0: SYS$OUTPUT:
$       SHOW QUE TTQ4:
$       IF .NOT $STATUS INIT/QUEUE/NOENABLE_GENERIC-
                /DEFAULT=NOFALG/TERMINAL-
                /ON=ALFAX::TTE4:            TTQ4:
$       DEFINE/SYSTEM/NOLOG    GEOPRINT:     TTQ4:
$       SET QUE/SCH=NOSIZE TTQ4:
$ !
$ ! Industrial controls printer
$ !
$       SET DEVICE/SPOOLED=(TTQ5:,DUA1:)     TTQ5:
$       ASSIGN/USER NLA0: SYS$OUTPUT:
$       SHOW QUE TTQ5:
$       IF .NOT $STATUS INIT/QUEUE/NOENABLE_GENERIC-
                /TERMINAL-
                /ON=ALFAX::TTE5:            TTQ5:
$       DEFINE/SYSTEM/NOLOG         BLDG5:   TTQ5:
$       SET QUE/SCH=NOSIZE TTQ5:
$ !
$ ! VERSATEC Printer/plotter in Geophysical systems area
$ !
$       SET DEVICE/SPOOLED=(TTQ6:,DUA1:)     TTQ6:
$       ASSIGN/USER NLA0: SYS$OUTPUT:
$       SHOW QUE TTQ6:
$       IF .NOT $STATUS INIT/QUEUE/NOENABLE_GENERIC-
                /DEFAULT=(NOFEED,NOFLAG)/PROCESSOR=VRSSMB-
                /ON=ALFAX::LWA0:            TTQ6:
$       DEFINE/SYSTEM/NOLOG    VERSATEC:     TTQ6:
$       SET QUE/SCH=NOSIZE TTQ6:
$ !
$ ! Define Cluster-wide batch queues
$ !
$       ASSIGN/USER NLA0: SYS$OUTPUT:
$       SHOW QUE/BATCH ALFAX_BATCH
$       IF .NOT $STATUS INIT/QUEUE/BATCH/START/JOB_LIM=3-
                /BASE_PRI=4/WSDEFAULT=256/WSQUOTA=512 -
                /WSEXTENT=750/ON=ALFAX::   ALFAX_BATCH
$       ASSIGN/USER NLA0: SYS$OUTPUT:
$       SHOW QUE/BATCH ENGBATCH
$       IF .NOT $STATUS INIT/QUEUE/BATCH/START/JOB_LIM=1-
                /BASE_PRI=4/WSDEFAULT=256/WSQUOTA=512 -
                /WSEXTENT=750/ON=ALFAX::   ENGBATCH
$       ASSIGN/USER NLA0: SYS$OUTPUT:
$       SHOW QUE/BATCH BETAX_BATCH
$       IF .NOT $STATUS INIT/QUEUE/BATCH/START/JOB_LIM=3-
                /BASE_PRI=4/WSDEFAULT=150/WSQUOTA=250 -
                /WSEXTENT=350/ON=BETAX::   BETAX_BATCH
$       ASSIGN/USER NLA0: SYS$OUTPUT:
$       SHOW QUE/BATCH GLBATCH
$       IF .NOT $STATUS INIT/QUEUE/BATCH/START/JOB_LIM=1-
                /BASE_PRI=4/WSDEFAULT=150/WSQUOTA=250 -
                /WSEXTENT=350/ON=BETAX::   GLBATCH
$ !
$ ! Invoke node specific queue startup
$ !
$ @SYS$MANAGER:QUESTART
$       SET NOVERIFY
```

-- File SYS$SYSROOT:[SYSMGR]QUESTART.COM on node A --

```
$ !
$ ! Start queues on ALFAX (VAX-A)
$ !
$       SET NOON
$ !
$ ! Load driver for the VERSATEC Printer/plotter
$ !
$       RUN SYS$SYSTEM:SYSGEN
RELOAD LWDRIVER/DRIVER=SYS$SYSTEM:LWDRIVER.EXE
CONNECT LWA0/ADAPTER=3/VECTOR=%0134/CSR=%0777354/DRIVER=LWDRIVER
EXIT
$ !
$ ! Set and spool local node printers
$ !
$ SET TERM/PERM/SPEED=4800/NOBROAD/NOANSI/NOHOST-
       /NOADV/TAB/NOWRAP/NOBRD/NOEDIT/NODEC/FORM-
       /NOECHO/PASTHRU              TTA6:
$ SET TERM/PERM/DEV=LA100/SPEED=2400/NOBROAD/PAGE=66-
       /WIDTH=220                   TTB0:
$ SET TERM/PERM/DEV=LA120/SPEED=1200/NOBROAD/FORM-
       /PAGE=66/NOTAB               TTE3:
$ SET TERM/PERM/DEV=LA120/SPEED=1200/NOBROAD/PAGE=66-
       /NOTAB/WID=132/FORM          TTE4:
$ SET TERM/PERM/DEV=LA36/WIDTH=132/PAGE=66/SPEED=2400-
       /NOTAB/FORM/NOBROADCAST/NOSET_SPEED/CRFILL=9-
       /LFFILL=9                    TTE5:
$ !
$       SET DEVICE/SPOOLED=(TTQ1:,DUA1:)    TTA6:
$       SET DEVICE/SPOOLED=(TTQ2:,DUA1:)    TTB0:
$       SET DEVICE/SPOOLED=(TTQ3:,DUA1:)    TTE3:
$       SET DEVICE/SPOOLED=(TTQ4:,DUA1:)    TTE4:
$       SET DEVICE/SPOOLED=(TTQ5:,DUA1:)    TTE5:
$       SET DEVICE/SPOOLED=(TTQ6:,DUA1:)    LWA0:
$ !
$ ! Start local printer queues
$ !
$       DEFINE/SYSTEM/NOLOG        SYS$PRINT         TTQ0:
$       DEFINE/SYSTEM/NOLOG        LPA0              TTQ0:
$       DEFINE/SYSTEM/NOLOG        LP0               TTQ0:
$       DEFINE/SYSTEM/NOLOG        LP                TTQ0:
$       START/QUEUE TTQ1:                   !ALFAX::TTA6:
$       START/QUEUE TTQ2:                   !ALFAX::TTB0:
$       START/QUEUE TTQ3:                   !ALFAX::TTE3:
$       START/QUEUE TTQ4:                   !ALFAX::TTE4:
$       START/QUEUE TTQ5:                   !ALFAX::TTE5:
$       START/QUEUE TTQ6:                   !ALFAX::LWA0:
$ !
$ ! Start the batch queues
$ !
$       DEFINE/SYSTEM/NOLOG SYS$BATCH ALFAX_BATCH:
$       START/QUEUE/BATCH ALFAX_BATCH:
$       START/QUEUE/BATCH ENGBATCH:
```

-- File SYS$SYSROOT:[SYSMGR]QUESTART.COM on node B --

```
$ !
$ ! Start queues on BETAX (VAX-B)
$ !
$       SET NOON
$ !
$ ! Set and spool local node printers
$ !
$       SET DEVICE/SPOOLED=(TTQ0:,DUA1:)    LPA0:
$       DEFINE/SYSTEM/NOLOG    SYS$PRINT:    LPA0:
$ !
$ ! Start local printer queues
$ !
$       START/QUEUE LPA0:                   !BETAX::LPA0:
$ !
$ ! Start batch queues
$ !
$       DEFINE/SYSTEM SYS$BATCH BETAX_BATCH:
$       START/QUEUE/BATCH BETAX_BATCH:
$       START/QUEUE/BATCH GLBATCH:
```

## AUTHORS INDEX


| AUTHORS | PAGE |
|---|---|
| Albert, J-N | 225 |
| Baisley, Wayne E. | 275 |
| Baker, June | 281 |
| Barrett, Daniel | 407 |
| Borger, Frank R. | 249, 257 |
| Buford, W. L. Jr. | 213 |
| Carey, Robert | 39 |
| Coker, Michael R. | 375 |
| Cossey, David V. | 181 |
| Darkangelo, D. G. | 557 |
| Davis, Ray | 29 |
| Denison, Marlays | 469 |
| Dennis, John | 399 |
| Dixon W. V. | 349 |
| Draughn, Mark | 165 |
| Duff, Steven G. | 531 |
| Feldner, Pat | 161 |
| Fiedeldey, Joseph W. | 531 |
| Froyd, Stan | 393 |
| Gabriel, Richard P. | 1 |
| Goldstein, Robert B. | 209, 511 |
| Hartwig, Arthur | 357, 365 |
| Heinicke, Peter | 463 |
| Hohmann, Edward C. | 195 |
| Huff, Al | 187 |
| Hughes, Steven | 7 |
| Hurst, Art | 435 |
| Jackson, James B. | 457 |
| Janik, Charles | 219 |
| Kaiser, Peter | 327 |
| Kane, Thomas | 35 |
| Leahy, B. C. | 501 |
| Lederman, Bart Z. | 129, 139 |
| Macko, Glen | 339 |
| Marshall, Ted A. | 531 |
| McGee, Robert E. | 349 |
| McGuigan, David | 39 |
| Merriam, E. William | 517 |
| Merritt, Glen Del | 331 |
| Mistretta, Paul | 21 |
| Molinari, John | 47 |
| Myers, L. M. | 213 |
| Nagy, Frank J. | 489 |
| Nestor, Bob | 565 |
| Nicinski, Tom | 463 |
| Orosz, Michael D. | 545 |
| Peli, Eli | 209 |

| AUTHORS | PAGE |
|---|---|
| Peterson, Ray | 165 |
| Racine, Phil | 21 |
| Rasmussen, Bob | 565 |
| Rafizadeh, Schumann | 541 |
| Rannenberg, Wendy | 493 |
| Rieck, JenniferL. | 405 |
| Robinson, Ann | 207 |
| Rotunni, Lisa M. | 195 |
| Rounds, James A. | 195 |
| Rousseaux, M. | 225 |
| Russ, Roger | 369 |
| Saxe David H. | 83 |
| Scott, Larry D. | 415 |
| Sewell, E. W. | 293 |
| Shehi, Don | 175 |
| Sherwood, Bruce A. | 179 |
| Slavich, A. L. | 287 |
| Smith, Daniel P. B. | 511 |
| Smith, Danny | 365 |
| Smith, Ted | 261 |
| Somes, Richard K. | 51 |
| Stabiner, Rivkah | 511 |
| Stefanek, George | 161, 165 |
| Straub, E. J. | 287 |
| Szep, Steven | 505 |
| Teeter, Brent | 473 |
| Tellis, Winston | 149 |
| Thomas, Lloyd K. | 453 |
| Thomas, J. D. | 191 |
| Thompson, D. E. | 213 |
| Thury, Dennis L. W. | 479 |
| Tibbetts, James | 317 |
| Turano, Thomas | 103 |
| Tyndale, Clyde L. | 229 |
| Tyrrill, Al | 521 |
| Uleski, Robert | 427 |
| Valentine Pamela A. | 113 |
| Vibert, J-F | 95, 225 |
| Wallace, Richard K. | 313 |
| Walthers, Denny | 449 |
| Waltz, Richard B. | 229 |
| Watson, Claude M. | 167 |
| Winter, C. | 287 |
| Wise, Rebecca | 13 |
| Wooledge, Karl | 209 |
| Yardley, John | 553 |
| Yoder, James R. | 123 |