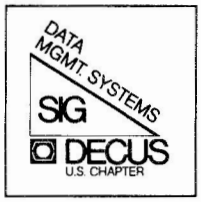


THE DATA MANAGERS

NEWSLETTER OF THE DATA MANAGEMENT SYSTEMS SPECIAL INTEREST GROUP

FEBRUARY Vol. 4 --- No. 2 1982

Please address contributions to:

Paul D. Clayton, Editor
c/o DECUS
One Iron Way, MR2-3/E55
Marlboro, Mass. 01752

CALL FOR MENU INPUT
MENU RESPONSES
SAMPLE PROGRAMS
VAX INFO. ARCH.
WORD PROCESSING

(DETAILS INSIDE)

(NEWS AT 11:00)

DMS SIG Newsletter Cumulative Index
Feb. 1982

Items of Interest from the Newsletter Editor
Paul D. Clayton

The Los Angles Symposium is a memory and a successful one at that. There were a number of events which bear acknowledgement.

First, the DMS SIG has a new SIG Chairman, Sandy Krueger. Sandy's opening letter is enclosed.

Second, VAX now has Information Architecture which is a complete package of data management software. Doug Dickey has the position of Product Group Chairman for the VAX Information Architecture products. If you have any questions, problems, reports, please call Doug. As I said in my first issue, the setup of this newsletter was expandable, and Vax Info. Arch. is now section 7.0 of the index. This issue has a brief overview of the products, with a better report due for the next issue.

Third, we now have a section on word processing interests that can be of value to you. This issue contains some comments I have concerning the DEC software package DX/IAS, and an article from the DECUS Proceedings of Miami, 1981. Some advanced concepts are explained about using DEC's word processing in a day-to-day environment. Word processing will be section 8.0 in the index from this day on.

Fourth, a number of SIG Steering Committee members have moved, assumed new roles or been added to the list. Please note the inclusion of a new membership list and update your notebook accordingly.

***** DMS SIG Newsletter Articles on SIG Tapes *****

As the newsletter editor, I have entered into machine readable media (WS200 & PDP 11/70) most of the articles that have appeared in this newsletter since I took over. What I have done is to take the articles that I have and submit them to all the DEC Operating System SIG's (RSX/IAS, RT, VAX, DEC10 and DEC20) tape copy program which is sponsored at each symposium. They should appear in the L.A. tape with file extensions of '.DMS'. There is a copy of the cummulitive index so that you can tell what articles you want. I hope to continue and expand this in the future so that everyone can get there copies as they need them. As far as I can tell, this is a first for any newsletter!!

***** New Name for This Newletter *****

We have changed our name to the 'The Data Managers'. The title of 'The Schema' is retired as of this issue. It is my hope that this name will continue to cover the broad areas the DMS SIG is responsible for.

Paul D. Clayton

Letter from DMS SIG Chairman
January 25, 1982

The worst part about being DMS SIG Chairman is the constant ache in the back of your head where you have made a mental note that Paul Clayton is expecting a letter for the next SIG Newsletter. So here goes.

The best place to begin is by thanking the past chairman and father of us all, Sat Mohan, for bringing us as far as we've gone and leaving me with the excellent organization that we now have. I'm also pleased to report that even though Sat is moving up into the upper echelon of DECUS, he has consented to continue with us for awhile as Planning Coordinator. For my money, a little Mohan is better than none.

Los Angeles was an important symposia for the SIG, primarily with the annoucement of the VAX Information Architecture product set. I expect to see a great deal of renewed interest in DEC information management software. Special thanks is due to the DEC contingent that attended L.A., led by Anita Moeder and Fred Howell. Some, but I'm sure not all, of the others were Rick Landau, Harri Rantiainen, Jim Donnelly, Tom McIntyre, Bill Noyes and Bill Harrelson.

The SIG Steering Committee was out in full force at L.A. and did a real bang-up job. While Sat and I were busy transferring batons, everything kept running because of their efforts. Thanks again.

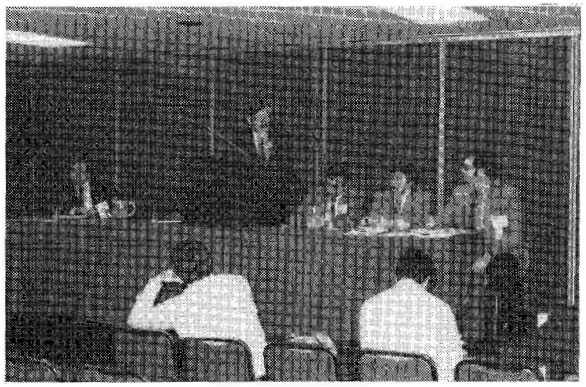
L.A. was important also because we had an opportunity to get together with the VAX and DATATRIEVE Sig's and do some advance "Coordinated" planning for Atlanta. Our three SIG's are jointly sponsoring an Information Management Theme Day at Atlanta where we will have a continuous run of sessions for both technical and managerial types, headed by a keynote speaker. Check the symposia catalog, when it arrives, for more details.

Another first at Atlanta will be half hour business sessions for each of our Product groups and some proposed Working groups where SIG members can get together, discuss needed activities and sessions, and hopefully volunteer to help organize some of the SIG activities in those areas. I hope to make these sessions a regular part of all future symposia.

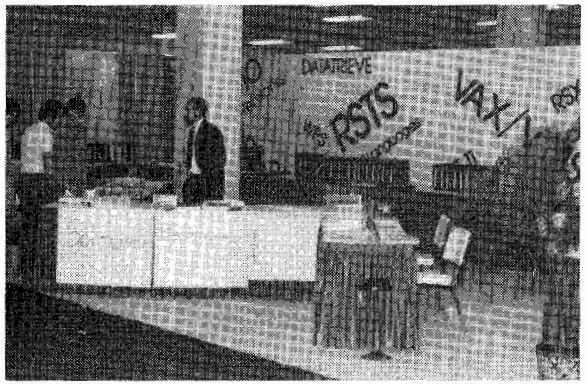
Now some notes about direction changes. The SIG in the past has existed primarily to support DEC products (i.e. DBMS-11, RMS, FMS, etc). We will continue to do so. But more and more we run into people at symposia, looking to discuss issues related to data management (i.e. DBMS with Decision Support Systems, DBMS on Micros, DBMS in a Scientific Environment, etc). In the past we have supported sessions when we could find them. In the future, I hope to have Working groups operating at the same level that our Product Groups operate, even to the extent of having DEC counterparts. The Working Group business sessions mentioned above will be our first attempt at setting them up. I hope to generate some support and alot of ideas for these groups. Please participate. Lets get issue oriented!

Sandy Krueger

(Left to Right)



Doug Dickey, Sandy Krueger,
Sat Mohan, Steve Pacheco
and Chris Wool

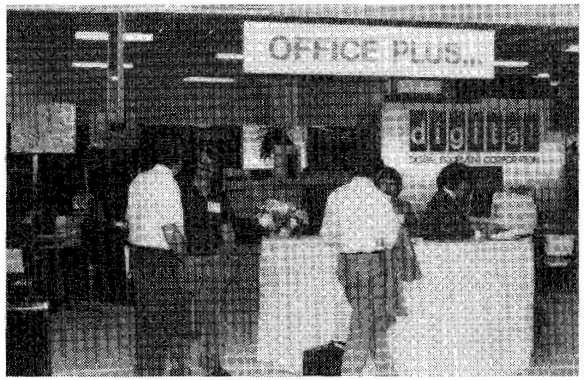


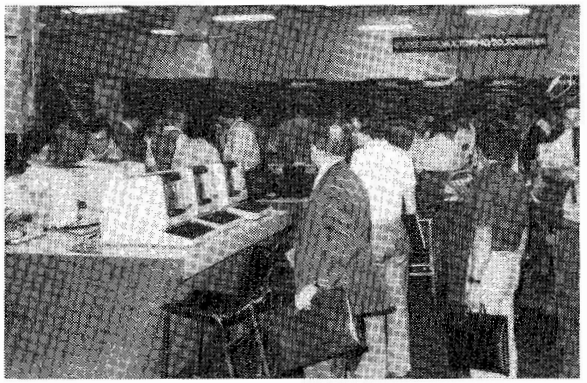
Now if they could only make the
RMS-11 Manual understandable!

Books Anyone?

Anything your heart and wallet
could want.

The new wave for DEC





The only notable operating
system not shown was IAS!

MENU PROCESS

The DMS SIG is changing the format of the MENU Procedure used to pass questions and comments to DEC for formal response.

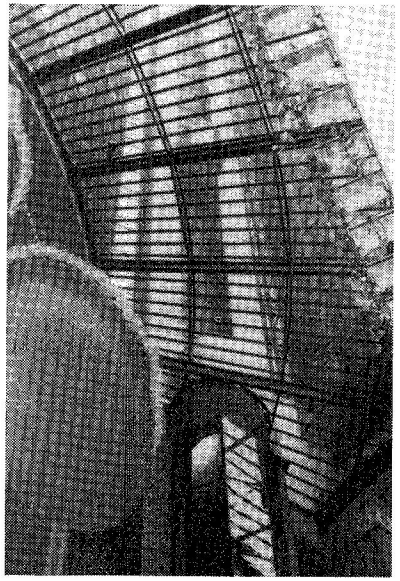
The goal of the new procedure is to obtain a faster response from DEC when their answer is known or obvious. We will ask the various groups within DEC to respond to menu items during the SIG closing session at subsequient symposia. These answers will also appear in the newsletter. Questions which will require analysis effort on DEC's part will also appear in the newsletter allong with their answers.

Newsletter readers are requested to send in MENU items prior to each symposia. These will be combined with the MENU items collected at the symposia and ranked based on frequency. The upper half of each group: DBMS-11, DBMS-32, DBMS-10/20, RMS, FMS, TECO, and any other data management related product produced by DEC and supported on the PDP-11, VAX and 10/20 systems.

This change is being made to allow an immediate response to questions which arise at the symposia. We hope this will benifit the people who are unable to attend symposia.

Please write down your requests for answers to bugs, enhancements or general questions on paper in as easily understood form as possible. Mail all menu items to the address listed below.

Steve Pacheco
Athena Systems Inc.
206 S. Broad Str.
Pawcatuck, CT. 06379



The elevators were not for the weak stomached!

Now you know why it takes so long to get things delivered! DEC uses Blimps!

DMS SIG Steering Committee Members
Feb 8, 1982

If you have any questions or concerns contact the person that has the responsibility for the area you want to discuss.

SIG Chairman
Sany Krueger (201) 541-8347
AMAX Copper Inc.
833 Roosevelt Ave.
Cartaret, N.J., 07008

Symposia Co-ordinator
Barbara Mann (213) 536-4190
TRW Inc.
One Space Park, R3/2030
Redondo Beach, Calif. 90278

Publications Co-ordinator
Stephen Pacheco (203) 599-3061
Athena Systems Inc.
206 S. Broad Str.
Pawcatuck, CT. 06379

Newsletter Editor
Paul D. Clayton (215) 441-2708
Republic Management Systems
One Neshaminy Interplex, Suite 306
Trevose, Pa. 19047

Product Groups Co-ordinator
Robert F. Curley (215) 662-3083
Department of Radiation Therapy
University of Pennsylvania Hospital
3400 Spruce Street
Philadelphia, Pa. 19104

DBMS-11 Product Group Chairman
Michael Antin (617) 684-7308
Polaroid Corp.
1265 Main Street W4-2B
Waltham, MA. 4570

DBMS 10/20 Product Group Chairman
Robert F. Curley
Same as above

RMS Product Group Chairman
Robert F. Curley
Same as above

FMS Product Group Chairman
Judy Kessler (617) 742-3140
Eye Research Institute of Retina Foundation
20 Staniford Street
Boston, Mass. 02114

TECO Product Group Chairman
Carl Marbach (215) 542-7008
RSTS Professional
Box 361
Fort Washington, Pa. 19034

Vax Information Architecture Product Group Chairman
Doug Dickey (703) 556-7400
CTEC Inc.
6862 Elm Str.
McLean, Virg. 22101

Seminars Co-ordinator
Sandy Krueger
Same as above

Performance Working Group Co-ordinator
Burt Weaver (612) 571-1249
Consulting Engineer
Weaver & Associates Inc.
2852 Anthony Lane South
Minneapolis, Minn. 55418

User Support Co-ordinator
Douglas Dickey
Same as above

Planning Co-ordinator
Satish Mohan (416) 461-8111 Ext. 269
TIW Industries Ltd.
Metals Group
629 Eastern Ave.
Toronto, Ontario
M4M 1E4

Projects Co-ordinator
T. Chris Wool (302) 366-4610
E.I. DuPont
Louviers Building
Wilmington, Delaware, 19898

DEC Counterpart
Anita L. Moeder
Digital Equipment Corp.
MK1-2/D03
Continental Blvd.
Merrimack, N.H. 03054

TECO Newsletter Co-ordinator
Greg Steinkuhler (305) 792-5100
TRT Telecommunications
P.O. Box 8876
Ft. Lauderdale, Fl. 33301

DMS SIG Site Profile

The purpose of this questionnaire is to obtain a profile of the sites of the members of the DMS SIG. Please take a few moments fill it out and return it to: T.C. Wool, E.I. DuPont, Engineering Dept. Louviers Bldg., Wilmington, DE 19898.

Name:_____ Address:_____

Company:_____ _____

Phone:_____ _____

Please circle the appropriate items below:

CPU Type: PDP-11 VAX System 10 System 20

Model:_____

Memory Size:_____(Specify Units)

Disk Drives:	RK05	RK06	RK07	RL01	RL02	RM02
	RM03	RM05	RM80	RP02	RP03	RP04
	RP05	RP06	RP07	RP20	Other_____	
Tape Drives:	TE16	TS11	TU10	TU16	TU40	TU45
	TU70	TU72	TU77	TU78	Other_____	
Operating System:	RSX-11M		RSX-11M+		Version_____	
	RSTS/E		RT-11		Version_____	
	VMS				Version_____	
	TOPS-10		TOPS-20		Version_____	
	IAS				Version_____	
Languages:	FORTRAN		COBOL		PASCAL	
	BASIC		PL/1		Other_____	
Layered Products:	RMS		FMS		DATATRIEVE	
	DBMS-11		VAX-11 DBMS		DBMS-10/20	
	DECNET		TECO			

Note - Respondees will receive a copy of all responses.

RMS-11 Workshop Session
Spring 1981 DECUS Symposium
Miami, Florida

Speaker: Tim Day - RMS Development Group

Prepared Questions Answered:

Q: An initial sequential GET by KRF field for an index file was done. The person then did a GET by RFA and RMS ignored the KRF field, the retrieve got the next record on the primary key. Why?
R: This is a user error in that it is well documented that a GET by RFA has no meaning except in the context of the primary key. When you have done a GET by RFA you have set your context into the primary key. Any values placed into the KRF field are meaningless and ignored.

Q: Why not place the index of a file on a separate device? This could be done by keeping the name of the index file in the prologue of the data file and opening it on a separate channel.
R: This will probably not come to life because of the large amount of problems that can arise in maintaining, tracking, backing-up and recovering that form of a file. If someone wanted to be really tricky and modify the RMS code on their own, it MIGHT be doable. Doing this with a "shared file" would cause problems!! The reason for doing this is really for a performance increase, DEC's overall committment to performance is by other means such as buckets, areas and placement.

Q: What is the best way to load an index file, last or first record first?
R: If a file is loaded in descending order you will end up with a file that is longer than necessary with records not packed to the best they could be. The smallest, densest file will result from loading the file in ascending order.

Q: Someone wanted a utility to compress a file on-line and in-place.
R: No committment to this from DEC, although a show of hands showed an interest in it.

Q: Another question was raised about "hashed indexes" for a file and its possible implementation in RMS.
R: This is just another way to retrieve data from a file in a much faster way than by using indices as RMS does. The proper use of hashed indices should get ANY record at random in two (2) I/O's. In the current use of indices, it COULD take many more I/O's to a device.

Questions From Attendees:

Q: What about placing the indices into a temporary file when an OPEN is executed, possibly on a separate disk to increase performance?
R: This might work for a file in a single-user per file environment. In a file openned for sharing, the indices would be extremely hard to maintain without corruption. If RMS were to become part of an operating system and have knowledge of everything going on, this might be feasible. Right now though, there are too many drawbacks to this idea.

Q: What is the true story about reclaiming space left from deleted records in an index file?

R: There are a lot of different types of deletes that take place. The worst case being that for a shared file with duplicate keys. The problem arises because RMS has to leave enough of the record to describe the primary key, which oddly enough is the only way RMS can tell that a record has been deleted. This is needed because another person may have your current record as his next record. Generally speaking, the remains (or fossils) of any deleted records are compressed in the buckets to leave a contiguous space which is then available to hold a record IF the record length PLUS any RMS overhead is smaller than the space available. The remains of deleted records will stay in the file for the life of the file or until it is reloaded, using IFL for example.

Q: What methods are available for optimizing the structure of a file such as that found on the VAX?

R: DEC is exploring the possibility of developing a utility to aid in defining a file with performance kept in mind.

POINTS OF INTEREST:

Concerning ODL Files and Resident Libraries.

An ODL file is a way to do disk overlaying and under the RMS implementation, if you use a non-overlayed version, you are going to get between 7 and 44KB of RMS code/buffers added to your task space. In an overlayed environment, RMS will get down to about 10KB of your task space in some situations. This may not apply when using RMS from a High Leval Langauge, such as Fortran IV+. The above numbers were gotten from using MACRO-11 assembly language. The disadvantages of disk overlays are many. First, the program execution speed can be affected depending on the sequence of operations and the overlay structure itself. This is because overlay implies I/O. Second, optimizing an ODL file is plain hard work. Third, the task image on disk will grow depending on the tree structure defined. This may cause problems for those systems cramped for disk room. Fourth, the time needed to task build an overlayed program increases depending on the tree structure.

The cure for many of these problems is the use of resident libraries. In terms of RMS, this means that you have a single, shared copy of RMS code placed into physical memory which all tasks are linked to at task build time. The first advantage is that there are no overlay I/O operations as all the library is resident together. Second, you don't need complex overlay files for RMS code. The ODL file that you would use is 3 lines long, which account for approx. 1.5KB of RMS code in your task image. Third, the task images on disk become smaller and the program should execute faster because of the lack of overlay I/O operations. The disadvantage to using libraries is that you give up 2 APR's (or 8KW of your task space), plus the little extra RMS code in your 24KW (remember you "lost" 2 APR's) task space.

The breakeven point at which to change from having separate copies of RMS code in everyone's task space and installing a resident RMS library which will consume approx. 46KB of physical memory is four (4) simultaneous memory resident tasks using RMS. This would decrease your OVERALL memory requirements with a possible increase in execution speed.

Goals of RMS.

The greatest goal of RMS is reliable tracking of a user's data. Considerable code was put into RMS to insure that data will not be lost or corrupted. This was done at the expense of execution time. Second, is a content addressable capability which generated the index file organization. Third, is the ability to have multiple indices. Fourth, good sequential access performance on the primary key. The overall structure of RMS was towards this goal. Fifth was fair to good access on alternate keys. This can be done by using bucket size, areas and placement. Sixth is Relative File Address (RFA) access which is guaranteed to be the same for the life of the file. Seventh is good space utilization within the file structure. An area for improvement is reclaimation of space left by deleted records.

Areas.

Even if you specify no areas, you are given an area zero (0), but you as a user don't know it. It won't even show when you "DSP /FU". The best conditions are if your file and, hence your areas, are contiguous. For the most part, the use of areas is a trial and error thing in search of the best performance. Proper use of areas will boost sequential access with little increase in random retrieval. The use of areas does not impact the user program, this frees you to try many schemes.

DEF Utility.

Is there any way to back up to previous entry in the case of errors?

There is a better (?) problem area in that DEF does not attach the terminal being used for input, which on a heavily loaded system, if DEF is checkpointed, any user input typed while DEF is checkpointed will be sent to MCR NOT DEF. Naturally, MCR does NOT know what a bucket is, you let your imagine take over from here. There is a patch in the works for this. In answer to re-entering previous parameters, a future release of DEF will confront this issue. CTRL-Z is the universal input to terminate an RMS utility.

Odds and Ends.

If a user has a contigious file with a bucket size that spans physical disk blocks, RMS will issue a multi-block read/write to the disk ACP. This increases the performance of your program.

DEC is looking at a means to zero out a file in preparation to re-populating it. This would allow a user to maintain the physical location of a file on a volume. This would be more controlled than deleteing and re-creating the file on a multi-user system.

RMS-11 LA DECUS Symposium Menu Response

1. Menu #1 / RMS / Enhancement / There should be a common syntax for all access methods and languages for the specification of RMS files.

   As the name implies, Record Management Services (RMS) is a set of subroutines that interface at a level between the language application program and the operating system. RMS itself has one interface (syntax) defined, which is documented in the RMS-11 Macro-11 Reference Manual. Any application, and high level languages ARE an application from the RMS point of view, determines its own syntax for using the RMS Macro interface. For most languages, I/O is described by a standard for the language. Therefore, the syntax is defined by the language, not by the underlying service.

2. Menu #2 / RMS-11 / Enhancement / Provide a new utility or change "RMSDEF" to calculate area sizes and recommend file factors given information such as expected number of record, length of record, fixed or variable length, key attributes, etc.

   RMS-11 has developed such a utility. Since we have not yet field tested it, we are not announcing this as part of the product. We do anticipate that it will be part of our next release.

3. Menu #4 / RMS-11 / Enhancement / Support for RMS in DIBOL.

   This comes as a surprise to us for two reasons. First, due to the manner in which DEC develops its products, it is not the responsibility of the RMS-11 group to require languages to support RMS. Second, and probably most confusing, is that DIBOL HAS supported RMS for a number of years. Possibly there is another question that is really being asked here, and we would be happy to respond to it.

4. Menu #5 / RMS-11 / Enhancement / Provide a utility to compress an indexed or relative file.

   Relative file compression is currently a feature of RMSCNV. When you specify a relative file as output and a relative file as input, CNV reads the input file sequentially, bypassing cells in the file without a valid record. It writes the output file sequentially also, beginning with cell #1 and continuing in ascending cell number order until end of file.

   Indexed compression is also accomplished by either RMSCNV or RMSIFL in the sense that buckets from the input file that are currently unused and unreclaimable by RMS will be "compressed" to free space in the output file, depending of course upon the output file's allocation.

   As to other definitions that the word compress could imply, we are researching methods to better reclaim space within a file that current processing algorithms cannot reclaim. We cannot predict what the final solution, if any, will be for future space reclamation functions.

   Another meaning of compress is data compression. This is also being investigated as a future enhancement to RMS-11. Again, we can't predict specifics on what solution would be chosen.

5. Menu #7 / RMS-11 / Enhancement / Provide record locking rather than bucket locking.

   Bucket locking is implemented via device - logical block contention lists kept by the operating system. Since RMS has context only within a task, it has no means of recording what activity is occuring on a file (device), and must depend upon the operating system to enforce the locking at a device level. To provide true record locking, RMS itself would have to be significantly changed to manage buffers globally among other things. Since it is our opinion that record locking is not dramatically more efficient than bucket locking, we would probably choose to put our development resources into other activities (i.e. compression and space reclamation).

6. Menu #10 / RMS-11 / Question / There appears to be problems associated with transferring files on DECnet using RMS. RMS has a much lower transfer rate than using qio operations. In addition, qio allows the sending of blocks larger than 512 bytes and RMS does not.

   Again, I believe there is some confusion. RMS-11 does not do network operations in its current release. If this is an RMS-32 question, then your assumptions are correct. For block I/O, RMS-32 does transfer one block at a time consisting of 512 bytes. Using qio, it is possible to select a larger transfer size consisting of multiple blocks.

OMSI Pascal Interface to RMS-11

The following is the beginning of the users manual for a package of routines that provide a means for using RMS-11 from OMSI Pascal. The complete package was submitted to the RSX/IAS SIG Tape for distribution on the the Los Angles, 1981 tape. Anyone can get a copy through there Local Users Group (LUG) so PLEASE DO NOT CALL Me OR Ken for a copy. The inclusion of the following excerpt is to wet your appetite and give you an idea of its capabilities. Our thanks go to Ken & Co. for giving it to DECUS !!!

PRM-11 USER'S GUIDE
OMSI PASCAL INTERFACE TO RMS-11
Kenneth G. Tibesar
3M Engineering Systems and Technology Labs
St. Paul, Minnesota 55144

Pascal Record Management User's Manual

Table Of Contents

Preface

Record Management Services (RMS-11) is a software package written and supported by DEC. The package is a data management tool that supports sequential, relative and indexed (ISAM) files.

Pascal Record Management (PRM-11) is a non-supported interface developed by 3M Engineering and Systems Technology Labs. PRM was developed as an interim until the OMSI Pascal Compiler directly supports RMS-11.

PRM was developed and tested using V1.2G of OMSI Pascal and DEC RSX-11M+ operating system. Known problems will occur when using OMSI Pascal V1.2H due to changes in initialization procedures. Version 2 of OMSI Pascal can be used only if the Version 1 switch is used.

Comments and questions concerning PRM should be directed in writing to:

Kenneth Tibesar
3M Center, Bldg. 518-1
St. Paul, Minnesota 55144

PLEASE - NO PHONE CALLS

Introduction to PRM

PRM-11 is a set of routines written in OMSI Pascal with in-line MACRO-11 code to interface to RMS-11. DEC supported high level languages interface to RMS-11 by using key words supported by the language compiler. The present versions of OMSI Pascal for the PDP-11 do not support RMS-11 interface. PRM provides the interface via a set of external procedures that are similiar to the key words used by DEC languages. PRM routines are called by the user code when RMS-11 operations are required. The PRM routines are linked to the user code during the task build.

PRM routines use DEC supported MACRO-11 routines to perform RMS-11 operations. RMS-11 requires attribute blocks describing the file (FAB) and record (RAB). PRM defines fields within the attribute blocks. This allows Pascal to directly control the contents of the files RABs and FABs. The contents of the RABs and FABs are indirectly controlled by the PRM user by loading pre-defined buffers and using one of the PRM external procedures.

User buffers required by PRM routines are allocated in the user space. The user defines only the number of buffers required for the application. The buffers are loaded by the PRM user to indicate file or record access parameters. The buffers are also used by PRM as working space. By using buffers allocated globally (from the heap), PRM can preserve variables between PRM calls.

Normally, RMS-11 requires the user to reserve POOL space. The POOL space is used by RMS internal code as control structures known as BDB's (Buffer Descriptor Blocks). The POOL space has to be allocated based on the number of simultaneously opened files. This design would waste user space that cannot be reclaimed even though the file is closed. To avoid this loss of valuable user space, PRM dynamically allocates and de-allocates the space when required by RMS-11. RMS-11 allows the MACRO-11 programmer to specify a GSA (Get Space) routine. The PRM GSA routine simply calls the Pascal Run-Time-System NEW and DISPOSE routines to manage HEAP space. RMS-11 uses the declared GSA routine when it requires or releases space. Space is not used until a file is opened and returned to the Pascal heap when the file is closed.

PRM was modeled after syntax used by existing DEC supported languages that interface to RMS-11. The following is a list of PRM and high level language comparisons:

DEC - Compilers support commands that perform RMS-11 file and record operations.

PRM - External procedures linked to the Pascal program perform RMS-11 file and record operations.

DEC- Descriptive key words are used with RMS-11 commands to specify attributes of the file and record operations.

PRM - Descriptive field names are used to load records passed to the external procedures. The record fields specify attributes of the file or record operation.

DEC - A completion code indicating the success or failure of each RMS operation is made available to the user. The completion code is available through the use of a reserved word.

PRM - A completion code is returned with each PRM call. The code is the standard RMS-11 completion code. In addition, PRM provides debugging completion codes for the PRM programmer.

The PRM interface to OMSI Pascal uses the Pascal internal record structure to pass data. Pre-defined records are included in the TYPE declaration of the User Pascal program. The Pascal programmer determines the operation to be performed (open, close, get record, etc.) and loads the appropriate record buffer. Once loaded, the record address is passed to the specific PRM routine that performs the file or record operation. The PRM routines load the RMS attribute blocks (RAB and/or FAB) with the control codes necessary to satisfy the user request. If data is to be stored or retrieved, the User Pascal program is also required to pass the address of a data buffer. The data buffer is loaded with data retrieved or the contents stored.

In addition to control and data buffer addresses, an address of a status buffer is required with all PRM calls. The status buffer is loaded with a completion code. The code is a standard MACRO-11 RMS success/error code (refer to RMS-11 User's Manual, Appendix B). In addition PRM will return debugging error codes for the PRM user. PRM error codes are only generated due to PRM user program procedural errors. Refer to Appendix D for detail on PRM error codes.

The user should check the completion code following each PRM call. If the file is not opened properly, no record operations should be performed. If the user continues after an error in a file open or creation, PRM will catch the error. All record operations on an improperly opened or created file are prohibited. Any call that attempts access to an improperly opened file will be rejected and a PRM error code will be returned.

DEAR DECUS,

THE ATTACHED PROGRAM IS MY LAST CONTRIBUTION TO THE CAUSE AS FMS' PRODUCT MANAGER. I HAVE SINCE MOVED ON TO OTHER THINGS, BUT FMS WILL BE ABLY CARED FOR BY JIM DONNELLY, A FORMER FMS USER WHO HAS JOINED DIGITAL.

THIS PROGRAM WAS INITIALLY DEVELOPED AS A GENERAL-PURPOSE CAI PROGRAM TO DEMONSTRATE HOW EASY IT IS WITH FMS TO WRITE GENERIC APPLICATIONS THAT ARE SHAPED COMPLETELY BY THE TEXT ON THE SCREEN AND BY THE NAMED DATA PARAMETERS STORED WITH THE FORM. IT TURNS OUT TO BE EVEN MORE GENERAL THAN THAT -- IT IS A GENERAL-PURPOSE QUESTION-ASKER AND ANSWER-LOGGER, WITH ANSWER VALIDATION LIMITED PRIMARILY BY THE FMS V1 LIMIT OF 16 NAMED DATA PARAMETERS PER FORM. I AM NOW USING IT TO GATHER MARKET RESEARCH INFORMATION IN MY NEW JOB IN THE CSS PRODUCT LINE.

IT IS MY GIFT TO THE MORE THAN A THOUSAND LOYAL FMS USERS WHO ARE ALREADY OUT THERE AND TO THE THOUSANDS MORE OF YOU WHO HAVE NOT YET DISCOVERED THE JOYS OF THIS FLEXIBLE, STIMULATING, EASY-TO-USE, AND FUN PRODUCT. I HOPE THAT IT WILL NOT ONLY BE USEFUL TO YOU IN ITS CURRENT FORM, BUT ALSO WILL PROVOKE SOME THINKING ON HOW TO USE SOME OF THE MORE OBSCURE FEATURES OF THE PRODUCT.

BOB NUSBAUM
DIGITAL EQUIPMENT CORPORATION


```
5  REM AUTHOR: ROBERT NUSBAUM - DIGITAL EQUIPMENT CORP. - FMS PRODUCT MANAGER (RETIRED)
10 REM QUICK AND DIRTY CAI DEMO
20 REM DESIGNED TO SHOW HOW EASY CAI BECOMES
30 REM WITH FMS-11 AND VAX-11 FMS
40 REM
50 REM THIS PROGRAM IS A GENERAL PROGRAM TO ASK QUESTIONS
60 REM VIA FMS FORMS AND LOG THE ANSWERS IN A FILE.
70 REM ALL INFORMATION OTHER THAN THE SCREEN DISPLAYS
80 REM IS STORED WITH THE FORMS USING THE "NAMED DATA"
90 REM FACILITY OF FMS.
100 REM
110 REM THIS PROGRAM WAS DEVELOPED USING BASIC+2 AND FMS/RSTS,
120 REM THEN MOVED TO VAX, AND THENCE TO A PDT USING RT-11 AND
130 REM BASIC-11.  IF ANYONE IS INTERESTED, IT COULD ALSO BE
140 REM RUN ON RSX AND IAS.
150 REM
160 REM THE PROGRAM TAKES ALL THE INPUT FROM THE USER AND
170 REM CONCATENATES IT INTO A SINGLE STRING (FGETAL WAS NOT
180 REM USED IN THIS CASE BECAUSE THE USER HAD A STRONG
190 REM OBJECTION TO THE FMS CONVENTION OF USING TAB FOR THE
200 REM NEXT FIELD KEY.).  IF A NAMED DATA ITEM NAMED "LOG"
202 REM IS PRESENT AND HAS A VALUE OF "Y", IT WRITES A RECORD
204 REM TO DISK CONTAINING THE FORM NAME AND THE DATA FROM THE
205 REM FORM.  IT THEN ATTEMPTS TO MATCH THE DATA ENTERED AGAINST
210 REM THE SET OF PRE-DEFINED (EXPECTED) ANSWERS FOR THIS
220 REM FORM, WHICH BY CONVENTION HAVE BEEN STORED AS NAMED
230 REM DATA WITH NAMES OF THE FORM
240 REM                                  ANSN.
250 REM IF NO MATCH IS FOUND, N IS SET EQUAL TO ZERO.
260 REM
270 REM THE PROGRAM THEN RESPONDS TO THE USER IN ONE OF THREE
280 REM WAYS.  THE PROGRAM RETRIEVES THE NAMED DATA ITEM WITH
282 REM THE NAME RSPN, USING THE N DETERMINED ABOVE.  THEY ARE
284 REM INTERPRETED AS FOLLOWS:
290 REM
300 REM   B=MESSAGE                    WRITE MESSAGE TO BOTTOM OF SCREEN
310 REM   F=FORMNAME                   OVERLAY FORM ON SCREEN WITH RESPONSE
320 REM   X                            DO NOTHING FOR RESPONSE
330 REM   NO ITEM NAMED RSPN EXISTS    DO NOTHING FOR RESPONSE
340 REM
350 REM
360 REM THE PROGRAM THEN TAKES ITS FLOW DIRECTIONS FROM NAMED
370 REM DATA PARAMETERS OF THE FORM ANSN.  THE FIVE OPTIONS
380 REM DEFINED ARE:
390 REM
400 REM   R                      RE-ASK THE QUESTION
410 REM   N=FORMNAME             GO TO NEXT FORM (SAME LIBRARY)
420 REM   L=LIBSPEC(FORMNAME)    GO TO NEXT FORM (CHANGE LIBRARIES)
430 REM   C                      ASK USER TO PRESS <CR> TO CONTINUE
435 REM                            (NEXT ACTION IS IN N.D. ITEM ACTNA)
440 REM   X                      EXIT FROM PROGRAM
450 REM
990 REM
1000 REM
1001 REM PGM BEGINS HERE
1002 REM
1010 CALL FINIT(3000)
1020 CALL FLOPEN('PD0:CAI.FLB')
1030 LET M$='FIRST '
```

```
1040 OPEN "PD1:CUSTO" FOR OUTPUT AS FILE #2%
1050 REM
1051 REM TOP OF ANSWER ANALYSIS ROUTINE
1052 REM
1060 LET C = RCTRLC
1070 REM DISABLE CONTROL-C ABORT
1100 CALL FCLRSH(M$)
1104 CALL FGETAF(V$,T%,F$,I%)
1106 REM CODE TO TREAT <CR> LIKE <TAB> UNLESS AT END OF FORM
1108 IF T%=0% THEN LET T%=1%
1110 CALL FPFT(T%)
1120 CALL FSTAT(S%)
1130 IF S%=-19% THEN  GO TO 1170
1160 GO TO 1104
1170 REM AT END OF FORM
1175 CALL FRETAL(X$)
1180 REM CHECK NAMED DATA WHETHER TO LOG ANSWERS ON THIS FORM
1190 CALL FNDATA('LOG    ',L$)
1200 CALL FSTAT(S%) @ IF S%=-15 THEN  GO TO 1220
1205 IF L$<>'Y' THEN  GO TO 1220
1210 PRINT #2%,M$&SEG$('       ',1%,6-LEN(M$));X$
1220 LET I%=1%
1222 LET B$='
1230 LET P$=B$
1240 LET I$=STR$(I%)
1245 CALL FNDATA('ANS'&I$&'  ',P$)
1250 CALL FSTAT(S%)
1260 IF S%=-15% THEN LET I$="0" @ GO TO 1300
1270 IF X$=P$ THEN  GO TO 1300
1280 LET I%=I%+1%
1290 GO TO 1240
1300 REM
1301 REM RESPOND TO STUDENT HERE
1302 REM
1310 CALL FNDATA('RSP'&I$&'  ',R$)
1315 CALL FSTAT(S%)
1320 IF S%=-15% THEN  GO TO 1360
1321 REM TREAT NO RESPONSE DEFINED AS 'X'
1330 LET R1$=SEG$(R$,1%,1%)
1340 IF R1$<>'B' THEN  GO TO 1350
1342 CALL FPUTL(TRM$(SEG$(R$,3,LEN(R$))))
1344 CALL FGET
1346 GO TO 1400
1350 IF R1$<>'F' THEN  GO TO 1360
1352 CALL FSHOW(SEG$(R$,3%,8%))
1354 CALL FPUTL('PRESS RETURN WHEN YOU ARE READY TO CONTINUE')
1356 CALL FGET
1357 CALL FCLRSH(M$)
1358 CALL FPUTAL(X$)
1359 GO TO 1400
1360 IF R1$='X' THEN  GO TO 1400
1399 REM
1400 REM
1401 REM ROUTINE TO GUIDE LESSON FLOW
1402 REM
1410 LET A$='
1415 LET A9$='ACT'&I$&'  '
1420 CALL FNDATA(A9$,A$)
1430 CALL FSTAT(S%)
1440 IF S%=-15% THEN  GO TO 1600
1450 LET A1$=SEG$(A$,1%,1%)
1460 IF A1$<>'R' THEN  GO TO 1470
1462 REM RE-ASK THE QUESTION
1464 CALL FPUTAL
1466 GO TO 1210
1470 IF A1$<>'C' THEN  GO TO 1500
1471 REM REQUEST 'RETURN' TO CONTINUE
1474 CALL FPUTL('PRESS RETURN WHEN YOU ARE READY TO CONTINUE')
1478 CALL FGET
1482 LET A9$='ACT'&I$&'A '
1486 GO TO 1420
1500 IF A1$<>'N' THEN  GO TO 1530
1501 REM 'N' MEANS GO TO NEXT FORM, SAME LIBRARY
1510 LET M$=SEG$(A$,3%,8%)
1520 GO TO 1100
1530 IF A1$<>'L' THEN  GO TO 1590
1531 REM 'L' MEAMS GO TO NEXT FORM, CHANGE LIBRARIES
1538 LET L$=SEG$(L$,3%,LEN(A$))
1542 LET L2%=POS(L$,'(',1%)
1546 IF L2%=0% THEN  GO TO 1800
1550 LET L1$=SEG$(L$,L2%-1%)
1551 REM GET LIBRARY FILESPEC
1554 CALL FLCLOS
1558 CALL FLOPEN(L1$)
1562 CALL FSTAT(S1%,S2%)
1566 IF S1%<0 THEN CALL FPUTL('STATUS 1 = '&STR$(S1%)&', STATUS 2 = '&STR$(S2%&'
       WHILE OPENING '&L1$)
1570 LET R2%=POS(L$,')',L2%+1%)
1574 IF R2%=0% THEN  GO TO 1800
1578 LET M$=SEG$(L$,L2%+1%,R2%-1%)
1582 GO TO 1210
1590 IF A1$='X' THEN  GO TO 5000
1600 CALL FPUTL('MISSING ACTION CODE IN FORM, EXITING LESSON')
1610 GO TO 5000
1800 REM BAD ACTION IN MAMED DATA
1810 CALL FPUTL('BAD ACTION: '&A$)
1820 STOP
5000 REM SHUTDOWN ROUTINE
5005 CLOSE #2%
5010 CALL FLCLOS
5020 CALL FLOPEN('CAI.FLB')
5030 CALL FCLRSH('BLANK ')
5040 CALL FLCLOS
5050 LET C = CTRLC
5060 GO TO 9999
9999 END
```

FMS PASCAL EXAMPLE

```
PROGRAM PasDem(DataFile);
{
*       Same program as extended example in Appendix B of the FMS/RSX Software
*       reference manual. Adapted directly from the FORTRAN program on
*       ps B-16, without all the checking for errors from the FMS routines.
*       Written for OMSI Pascal version 1.26.
*       Uses the FORTRAN callable FMS HLL routines.
*
*       Requires the following task-build command:
*
*           >TKB PasDem/CP/FP=PasDem,[300,4]HLLFOR,FDVLIB/LB,[300,4]PASLIB/LB
*
*       Author:
*
*               Kurt W. Parke
*               Spectronix, Inc.
*               24590 Glen Rd.
*               Shorewood, MN 55331
*               Tel. (612) 474-0831
*
*       Edit:   10/21/81 Modify for OMSI Pascal 2.0H
}

CONST
  ImpureSize = 1000;            { Size of impure area }
  FmsLun = 4;                   { LUN for FMS library I/O }
  MaxString = 7;                { Maximum string length }
  LineLength = 15;              { Maximum "fputl" line length }
  MaxFileName = 30;             { Maximum length of file names }
  MaxDataLength = 255;          { Maximum data (bytes) for "fsetal" }

TYPE
  ImpureType = PACKED ARRAY [1..ImpureSize] OF char;
  String = PACKED ARRAY [1..MaxString] OF char;
  ResponseType = PACKED ARRAY [1..3] OF char;
  FileNameType = PACKED ARRAY [1..MaxFileName] OF char;

VAR
  Null : char;
  DataFile : text;
  ImpureArea : ImpureType;
  Size : integer;
  Lun : integer;
  LibName : String;
  Response : ResponseType;
  Form, Form1 : String;
  ExitForm : String;
  FileName : FileNameType;

{
*       Globally accessible FMS routines from HLLFOR
}

PROCEDURE Fclrsh(VAR Name : String); NonPascal;
PROCEDURE Finit(VAR Imp : ImpureType; VAR Siz : integer); NonPascal;
PROCEDURE FlChan(VAR Chan : integer); NonPascal;
PROCEDURE FlOpen(VAR Name : String); NonPascal;
PROCEDURE FnData(VAR res : ResponseType; VAR Fname : FileNameType);
  NonPascal;
PROCEDURE FlClos; NonPascal;
```

```
{
*       Main program code
}

BEGIN { PasDem }
  Null := chr(0);
  Asciz('.EXIT. ',ExitForm);
  Size := ImpureSize;
  Finit(ImpureArea,Size);
  Lun := FmsLun;
  FlChan(Lun);
  Asciz('DEMLIB ',LibName);
  FlOpen(LibName);
  REPEAT
    GetMenu('FIRST  ','CHOICE ',Response,Form1);
    IF Form1 <> ExitForm THEN
      BEGIN
      Response[2] := 'F'; Response[3] := Null;
      FnData(Response,FileName);
      OpenFile(DataFile,FileName);
      REPEAT
        Form := Form1;
        FormSeries(Form,DataFile);
        GetMenu('LAST   ','CHOICE ',Response,Form)
      UNTIL Response[1] <> '1';
      close(DataFile)
      END
  UNTIL (Form = ExitForm) OR (Form1 = ExitForm);
  Flclos
END. { PasDem }
```

```
{
*       Routine to open a file given an ASCIZ file name.  The file name must
*       be blank filled to avoid a syntax error from the Pascal run-time
*       system.
}

PROCEDURE OpenFile(VAR FileVar : text; FileName : FileNameType);

  VAR NullPos,i : integer;

  BEGIN { OpenFile }
    NullPos := 1;
    WHILE FileName[NullPos] <> Null DO NullPos := NullPos + 1;
    FOR i := NullPos TO MaxFileName DO FileName[i] := ' ';
    rewrite(FileVar,FileName)
  END; { OpenFile }
```

```
{
*       Routine to follow a series of data entry forms.  After each form
*       is displayed and the data stored in a file, the routine looks in
*       the Name Data for the next form to process.  If the next form is
*       '.NONE.' the end of the series has been hit and the routine exits.
}

PROCEDURE FormSeries(VAR Form : String;          { Starting form name }
                     VAR FileVar : text);        { File variable for I/O }

  TYPE DataBuffer = ARRAY [1..MaxDataLength] OF char;

  VAR
    Count,DataPos : integer;
    Data : DataBuffer;
    DataName,NoneForm : String;

  PROCEDURE Fgetal(VAR buf : DataBuffer); NonPascal;
  PROCEDURE FnData(VAR Inp : String; VAR Data : String); NonPascal;

  BEGIN { FormSeries }
    Asciz('.NONE. ',NoneForm);
    REPEAT
      Fclrsh(Form);
      Fgetal(Data);
      DataPos := 1;
      Count := 1;
      WHILE Data[DataPos] <> Null DO
        BEGIN
        write(FileVar,Data[DataPos]);
        IF Count = 78 THEN
          BEGIN
          writeln(FileVar);
          Count := 1
          END;
        DataPos := DataPos + 1
        END; { WHILE }
      writeln(FileVar);
      Asciz('NXTFRM ',DataName);
      FnData(DataName,Form);
    UNTIL Form = NoneForm;
  END; { FormSeries }
```



```
{
*       Routine to produce an ASCIZ string from a blank-filled string.
*       This is often necessary since all the FMS FORTRAN routines require
*       ASCIZ strings, and the Pascal compiler does not insert a null.
}

PROCEDURE Asciz(    InString : String;           { Input string }
                VAR OutString : String);         { Output string }
  BEGIN
    OutString := InString;
    OutString[MaxString] := Null
  END; { Asciz }

{
*       Routine to get a menu choice from the operator.  Displays the menu
*       and loops until the operator types in a valid response.
}

PROCEDURE GetMenu(     Menu : String;            { Name of menu to display }
                       Data : String;            { Response field name }
                VAR Response : ResponseType;     { Response value }
                VAR Namedata : String);          { Name Data of response }

  TYPE LineType = PACKED ARRAY [1..LineLength] OF char;

  VAR
    Terminator, Status, Stat2 : integer;
    ErrorLine : LineType;

  PROCEDURE FnData(VAR res : ResponseType; VAR Data : String);
    NonPascal;
  PROCEDURE Fget(VAR res : ResponseType; VAR Trm : integer; VAR Nam : String);
    NonPascal;
  PROCEDURE Fstat(VAR Stt : integer; VAR stt2 : integer); NonPascal;
  PROCEDURE Fputl(VAR line : linetype); NonPascal;

  BEGIN { GetMenu }
    ErrorLine := 'Illegal Choice ' ErrorLine[LineLength] := Null;
    Asciz(Menu,Menu);
    Fclrsh(Menu);
    Asciz(Data,Data);
    REPEAT
      Fget(Response,Terminator,Data);
      FnData(Response,NameData);
      Fstat(Status,Stat2);
      IF Status <= 0 THEN Fputl(ErrorLine)
    UNTIL Status > 0
  END; { GetMenu }
```

FMS Application Programming Handout
Miami '81 Symposium

********** FORM DRIVER ********

1. RUN-TIME COMPONENT OF RMS.
2. SET OF SUBROUTINES TO ALLOW AN APPLICATION PROGRAM TO ACCESS FORMS AND CONTROL OPERATOR INTERACTION.
3. LINKED WITH APPLICATION PROGRAM.

******** FORM DRIVER FEATURES ********

1. FLEXIBLE SET OF CALLS.
2. INTERFACES TO COMMON PROGRAMMING LANGUAGES.
3. SUPPORT FOR FORM AND FIELD ATTRIBUTES SUPPORT FOR OPERATOR HELP.

******** FORMS CONTAIN ********

1. CONSTANT BACKGROUND TEXT.
2. VARIABLE FIELDS.
3. ASSOCIATED NAMED DATA.

******** FORM ATTRIBUTES ********

1. FORM NAME.
2. SCREEN WIDTH (80 OR 132 COLUMNS).
3. SCREEN AREA TO CLEAR.
4. SCREEN BACKGROUND.
5. HELP FORM NAME.

******** BACKGROUND TEXT ********

1. CONSTANT INFORMATION.
2. CANNOT BE MODIFIED BY OPERATOR.
3. PROMPTS FOR FIELDS OR EXPLANATORY TEXT VT100 VIDEO ATTRIBUTES.

******** FIELDS ********

1. IDENTIFIED BY SIX CHARACTER NAME AND INDEX.
2. MODIFIABLE BY APPLICATION AND/OR TERMINAL OPERATOR.
3. ASSOCIATION BETWEEN PROGRAM AND FIELDS MADE AT RUN TIME.

******** FIELD ATTRIBUTES ********

1. VT100 VIDEO ATTRIBUTES.
2. DEFAULT VALUE.
3. HELP TEXT.
4. EMBEDDED TEXT CHARACTERS.
5. RIGHT OR LEFT JUSTIFICATION.
6. AUTOTAB TO NEXT FIELD.
7. DISPLAY-ONLY.
8. CLEAR CHARACTER FOR SCREEN DISPLAY.
9. BLANK OR ZERO FILL.
10. FIXED DECIMAL FOR MONEY.
11. INDEXED.
12. NO-ECHO SUPERVISOR PROTECT.

******** DATA VALIDATION ********

1. CHARACTER VALIDATION BASED ON FIELD PICTURE.
2. RESPONSE REQUIRED IN FIELD.
3. MUST FILL FIELD IF ANY DATA ENTERED.

******** HELP ********

1. HELP TEXT FOR EACH FIELD.
2. HELP FORM FOR EACH FORM.
3. HELP FORMS CAN BE CHAINED TRANSPARENT TO CALLING PROGRAM.

******** NAMED DATA ********

1. ASSOCIATES ASCII DATA WITH FORM.
2. NOT DISPLAYED AS PART OF FORM.
3. CAN BE USED TO EMBED APPLICATION LOGIC IN FORMS.
4. UP TO 60 BYTES OF TEXT PER ENTRY.
5. UP TO 16 ENTRIES PER FORM.
6. ACCESSED BY NAME OR INDEX.

******** FMS INITIALIZATION ********

1. SET UP FORM DRIVER IMPURE AREA.
2. OPEN FORM LIBRARY.

FMS CALLS TO USE:

```
      CALL FINIT (IMPURE, 2000)          ; INITIALIZE IMPURE AREA
  *   CALL FLCHAN (1)                    ; SET FORM LIBRARY
                                         ; CHANNEL
  *   CALL FLOPEN ('FRMLIB.FLB')         ; OPEN FORM LIBRARY
```

* NOT REQUIRED FOR MEMORY RESIDENT FORM SUPPORT

******** BASIC FORM PROCESSING CALLS ********

CASE 1. DISPLAY A FORM.
CASE 2. WRITE DATA TO A FORM.
CASE 3. GET INPUT FROM THE TERMINAL OPERATOR.
CASE 4. RETURN FIELD VALUES FROM A FORM.
CASE 5. DISPLAY APPLICATION MESSAGES.

******** DISPLAY A FORM ********

CLEAR ENTIRE SCREEN AND DISPLAY FORM. - (FCLRSH)
CLEAR SPECIFIED AREA AND DISPLAY FORM. - (FSHOW)

WRITE DATA TO A FORM.
    WRITE DATA TO ALL FIELDS. - (FPUTAL)
    WRITE DATA TO A SPECIFIED FIELD. - (FPUT)

GET INPUT FROM THE TERMINAL OPERATOR
    GET ALL FIELDS. - (FGETAL)
    GET A SPECIFIED FIELD. - (FGET)
    GET ANY FIELD. - (FGETAF)

CASE 1: NO APPLICATION PROCESSING OF INPUT IS NECESSARY AT THE FIELD LEVEL.

FMS CALLS TO USE:
WRITE DATA TO ALL FIELDS. - (FPUTAL)
GET ALL FIELDS. - (FGETAL)

```
        CALL FCLRSH ('FORM')                ; DISPLAY FORM
        READ RECORD INTO DATA               ; GET RECORD FROM DATA BASE
        CALL FPUTAL (DATA)                  ; WRITE DATA TO FORM
        CALL FGETAL (DATA)                  ; LET OPERATOR MODIFY DATA
        WRITE RECORD FROM DATA              ; WRITE UPDATED RECORD
```

ADVANTAGES:
1. APPLICATION REQUIRES NO KNOWLEDGE TO FORM CONTENT OR LAYOUT.
2. FORM DRIVER HANDLES ALL INTERACTION WITH TERMINAL OPERATOR.
3. TERMINAL OPERATOR CAN MOVE BETWEEN FIELDS AT WILL UNTIL SATISFIED.

DISVANTAGES:
1. APPLICATION CANNOT PROCESS INPUT UNITL ENTIRE FORM IS COMPLETE.
2. ORDER OF FIELDS IN A RECORD MUST BE THE SAME AS THE ORDER IN A FORM.

CASE 2: APPLICATION PROCESSING IS NECESSARY AT THE FIELD LEVEL AND/OR THE APPLICATION PROGRAM REQUIRES CONTROL OVER THE ORDER OF FIELD ENTRY.

FMS CALL TO USE:
GET A SPECIFIED FIELD. - (FGET)

```
        CALL FCLRSH ('ORDER')               ; DISPLAY FORM
10      CALL FGET (ACCT, TERM, 'ACCT')      ; GET ACCOUNT NUMBER
        PROCESS INPUT                       ; PROCESS
        IF ACCT NOT VALID GOTO 10           ; IF INVALID, GET FIELD AGAIN
20      CALL FGET (PARTNO, TERM, 'PARTNO')  ; ELSE GET PARY NUMBER
        PROCESS INPUT                       ; PROCESS
        IF PARTNO NOT VALID GOTO 20         ; IF INVALID, GET FIELD AGAIN
        CALL FPUT (DESC, 'DESC')            ; ELSE WRITE DESCRIPTION FIELD
        CALL FGET (QUANT, TERM, 'QUANT')    ; GET QUANTITY TO ORDER
        .
        .
        .
```

ADVANTAGES:
1. APPLICATION PROGRAM GETS IMMEDIATE CONTROL AFTER EACH FIELD IS ENTERED TO VALIDATE INPUT, RESPOND TO ERRORS, AND UPDATE THE FORM.
2. APPLICATION PROGRAM HAS ABSOLUTE CONTROL OVER ORDER IN WHICH FIELDS ARE ACCESSED

DISAVANTAGES:
1. TERMINAL OPERATOR HAS NO CONTROL OVER MOVEMENT BETWEEN FIELDS.
2. APPLICATION PROGRAM MUST ACCESS ALL FIELDS IN A FORM BY NAME.

CASE 3: APPLICATION PROCESSING IS NECESSARY AT THE FIELD LEVEL AND THE TERMINAL OPERATOR SHOULD CONTROL MOVEMENT BETWEEN FIELDS.

FMS CALLS TO USE:
GET A SPECIFIED FIELD. - (FGET)
PROCESS A FIELD TERMINATOR. - (FPFT)

```
10      CALL FGET (RESP, TERM, FIELD)       ; GET INPUT FOR A FIELD
        PROCESS INPUT FOR THE FIELD         ; PROCESS INPUT
        IF TERM EQUAL 1 GOTO 20             ; IF TERMINATOR ENTER, FORM
                                            ; COMPLETE
        CALL FPFT (TERM)                    ; ELSE DETERMINE NEXT FIELD TO
                                            ; GET
        CALL FGCF (FIELD)                   ; GET FIELD NAME
        GOTO 10                             ; GET INPUT FOR FIELD
20      .
        .
        .
```

USE OF FGET/FPFT CALLS:

```
        CALL FCLRSH ('FORM')                ; DISPLAY FORM
        CALL FGET (RESP, TERM, '*')         ; GET INPUT FOR FIRST FIELD
        CALL FGCF (FIELD)                   ; GET FIELD NAME
        GOTO 20                             ; PROCESS INPUT
10      CALL FGET (RESP, TERM, FIELD)       ; GET INPUT FOR A FIELD
20      PROCESS INPUT FOR FIELD             ; PROCESS INPUT
        CALL FPFT (TERM)                    ; DETERMINE NEXT FIELD TO GET
        CALL FGCF (FIELD)                   ; GET FIELD NAME
        IF TERM NOT EQUAL 1 GOTO 10         ; IF TERMINATOR NOT ENTER, GET
                                            ; THE FIELD
        CALL FSTAT (STAT)                   ; ELSE CHECK FOR INCOMPLETE
                                            ; FORM
        IF STAT EQUAL 2 GOTO 10             ; GET INPUT FOR INCOMPLETE
                                            ; FIELD
                                            ; ELSE DONE
        .
        .
        .
```

ADVANTAGES:
1. TERMINAL OPERATOR CAN MOVE BETWEEN FIELDS AT WILL UNITL SATISIFED.
2. APPLICATION PROGRAM GETS IMMEDIATE CONTROL AFTER EACH FIELD IS ENTERED TO VALIDATE INPUT, RESPOND TO ERRORS, AND UPDATE THE FORM.
3. APPLICATION PROGRAM REQUIRES NO KNOWLEDGE OF FORM LAYOUT.

DISAVANTAGES:
1. MORE COMPLEX TO IMPLEMENT THAT THE SINGLE 'GET ALL FIELDS' CALL.

CASE 4: INPUT IS REQUIRED IN ANY ONE FIELD IN A FORM.

FMS CALL TO USE:
GET ANY FIELD. - (FGETAF)

```
        CALL FCLRSH ('FORM1')                   ; DISPLAY FORM WITH TWO FIELDS
                                                ; NAME AND BADGE NUMBER
        CALL FGETAF (RESP, TERM, FIELD)         ; GET INPUT FOR EITHER FIELD
        IF FIELD EQUAL 'BADGE' GOTO 10          ; BRANCH IF BADGE NUMBER
        ELSE FIND BADGE NUMBER CORRESPONDING TO NAME ENTERED
        CALL FPUT (BADGE, 'BADGE')              ; DISPLAY BADGE NUMBER
        GOTO 20
10      FIND NAME CORRESPONDING TO BADGE NUMBER ENTERED
        CALL FPUT (NAME, 'NAME')                ; ELSE DISPLAY NAME
20      .
        .
        .
```

CASE 5: APPLICATION PROGRAM HAS TO SYNCHRONIZE WITH THE TERMINAL OPERATOR

FMS CALL TO USE:
GET A SPEICIFIED FIELD. - (FGET)

```
        CALL FCLRSH ('FORM1')                   ; DISPLAY FORM
        CALL FGETAL (DATA)                      ; GET INPUT FOR FORM
        PROCESS INPUT                           ; PROCESS
        CALL FPUT (VALUE1,'FIELD1')             ; WRITE DATA
        CALL FPUT (VALUE2,'FIELD2')             ; WRITE DATA
        CALL FGET                               ; WAIT FOR TERMINAL OPERATOR TO
                                                ; ACKNOWLDEGE
        CALL FCLRSH ('FORM2')                   ; DISPLAY NEXT FORM
        .
        .
        .
```


******** RETURN FIELD VALUES FROM FORM ********
(NO OPERATOR INPUT ALLOWED)

CASE 1. RETURN A SPECIFIED FIELD VALUE
CASE 2. RETURN ALL FIELD VALUES

CASE 1: ACCESS INDIVIDUAL FIELDS IN A FORM AFTER THE FORM IS COMPLETED BY THE TERMINAL OPERATOR IN ORDER TO: VALIDATE INPUT FOR INDIVIDUAL FIELDS. CREATE A RECORD WITH FIELDS IN A DIFFERENT ORDER THAN THEY ARE IN THE FORM. CREATE A RECORD CONTAINING FIELDS FORM SEVERAL SOURCES, INCLUDING THIS FORM.

FMS CALL TO USE:
RETURN A SPECIFIED FIELD. - (FRETN)

```
        CALL FCLRSH ('FORM')                    ; DISPLAY FORM
        CALL FGETAL                             ; ALLOW OPERATOR TO COMPLETE
                                                ; FORM BUT DON'T RETURN DATA
        CALL FRETN (RESPA, 'FIELDA')            ; RETURN VALUE FORM FIELD A
10      PROCESS FIELD VALUE                     ; PROCESS DATA
        IF RESPA VALID GOTO 20                  ; IF VALID CONTINUE
        CALL FGET (RESPA, TERM, 'FIELDA')       ; ELSE GET INPUT FOR FIELD
        GOTO 10                                 ; PROCESS
20      CALL FRETN (RESPB, 'FIELDB')            ; RETURN VALUE FOR FIELD B
        PRECESS FIELD VALUE                     ; PROCESS
        .
        .
        .
```

CASE 2: RETURN THE DATA FOR AN ENTIRE FORM AS A SINGLE RECORD AFTER GETTING OPERATOR INPUT FOR EACH FIELD INDIVIDUALLY.

FMS CALL TO USE:
RETURN ALL FIELDS. - (FRETAL)

```
        CALL FCLRSH ('FORM')                    ; DISPLAY FORM
10      CALL FGET (RESP, TERM, FIELD)           ; GET INPUT FOR A FIELD
        PROCESS INPUT FOR FIELD                 ; PROCESS INPUT
        IF TERM EQUAL 1 GOTO 20                 ; IF TERMINATOR ENTER, FORM
                                                ; COMPLETE
        CALL FPET (TERM)                        ; ELSE DETERMINE NEXT FIELD TO
                                                ; GET
        CALL FGCF (FIELD)                       ; GET FIELD NAME
        GOTO 10                                 ; GET INPUT FOR FIELD
20      CALL FRETAL (DATA)                      ; RETURN ALL FIELD VALUES
        WRITE RECORD FORM DATA                  ; WRITE RECORD
        .
        .
        .
```

******** DISPLAY APPLICATION MESSAGES ********

1. ERROR MESSAGES
2. INFORMATIONAL MESSAGES

FMS CALL TO USE:
WRITE MESSAGE TO LAST LINE OF SCREEN. - (FPUTL)

```
10      CALL FGET (RESP, TERM, 'CHOICE')        ; GET FIELD
        IF RESP VALID GOTO 20                   ; IF RESPONSE VALID CONTINUE
                                                ; ELSE DISPLAY ERROR MESSAGE
        CALL FPUTL ('INVALID RESPONSE. VALID CHOICES ARE 1, 2, OR 3.')
        GOTO 10                                 ; GET THE FIELD AGAIN
20      .
        .
        .
```

```
        ******** ADDITIONAL FORM DRIVER CALLS SUPPORT ********

1.      NAMED DATA
2.      SUPERVISOR ONLY MODE
3.      SCROLLED AREAS
4.      NAMED DATA USAGE
        A.      RANGE CHECKING
        B.      TABLE LOOKUPS
        C.      FORM LINKAGE
        D.      FORM SPECIFIC INFORMATION
5.      ACCESS NAMED DATA
        A.      BY NAME (FNDATA)
        B.      BY INDEX (FIDATA)

        ******** USE OF FNDATA FOR FORM LINKAGE ********


10      CALL FCLRSH (FORM)                  ; DISPLAY FORM
        CALL FGETAL (DATA)                  ; GET INPUT FOR FORM
        PROCESS INPUT                       ; PROCESS
        CALL FNDATA ('NXTFRM, FORM')        ; GET NAME OF NEXT FORM
        CALL FSTAT (STAT)                   ; CHECK FOR NO NAMED DATA
        IF STAT > 0 GOTO 10                 ; IF NAMED DATA FOUND, DISPLAY
                                            ; NEXT FORM
        .
        .
        .

        ******** USE OF FNDATA FOR MENUS ********


        CALL FCLRSH ('MENU')                ; DISPLAY MENU FORM
10      CALL FGET (RESP, TERM, 'CHOICE')    ; GET MENU SELECTION
        CALL FNDATA (RESP,FORM)             ; CHECK FOR NO NAMED DATA
        IF STAT > 0 GOTO 20                 ; IF FOUND OK
        CALL FPUTL ('INVALID SELECTION')    ; ELSE INVALID RESPONSE
        GOTO 10                             ; GET ANOTHER SELECTION
20      CALL FCLRSH (FORM)                  ; DISPLAY FORM CORRESPONDING
                                            ; TO SELECTION SUPERVISOR ONLY
                                            ; MODE
        .
        .
        .

        ******** SUPERVISOR ONLY MODE ********

1.      PROVIDES FIELD PROTECTION FOR DATA RETRIEVAL AND MODIFICATION CONTROL
2.      ALLOW OPERATOR ACCESS TO SUPERVISOR ONLY FIELDS (FSPOFF)
3.      RESTRICT OPERATOR ACCESS TO SUPERVISOR ONLY FIELDS (FSPON)

USE OF FSPOFF/FSPON:

        CALL FCLRSH ('ACCESS')              ; DISPLAY FORM
        CALL FGET (PASSWD, TERM, 'PASSED')  ; GET PASSWORD
        IF PASSWD EQUAL 'XYZZY'             ; IF CORRECT PASSWORD
        CALL FSPOFF                         ; ALLOW OPERATOAR ACCESS TO
                                            ; SUPERVISOR ONLY FIELDS
```



```
PROCESS FORMS

        CALL FSPON                          ; RESET SUPERVISOR ONLY
                                            ; MODE TO RESTRICT
                                            ; ACCESS TO FIELDS ACCESS
                                            ; SCROLLED AREAS

WRITE DATA TO A LINE IN A SCROLLED AREA (FOUTLN)
GET INPUT FOR A LINE IN A SCROLLED AREA (FINLN)
USE OF FOUTLN:

;
; INITIALIZE A FIVE LINE SCROLLED AREA FROM
; A DATA BUFFER.
;
        CALL FCLRSH ('FORM')                ; DISPLAY FORM
        CALL FOUTLN ('SCRFLD' , A(1)        ; WRITE DATA TO FIRST LINE OF
                                            ; SCROLLED AREA
        REPEAT FOR I = 2 TO 5               ; INITIALIZE THE REST
        CALL FPFT (8, 'SCRFLD')             ; MOVE DOWN ONE LINE
        CALL FOUTLN ('SCRFLE' , A(1)        ; WRITE DATA TO NEW LINE
        END REPEAT                          ;
        REPEAT 4 TIMES                      ; GO BACK TO FIRST LINE
        CALL FPFT (9, 'SCRFLE')             ; MOVE UP ONE LINE
        END REPEAT

USE OF FINLN/FOUTLN:
;
; ALLOW ENTRY AND REVIEW OF 50 LINES OF
; DATA IN A SCROLLED AREA.
;
        CALL FCLRSH ('FORM')                ; DISPLAY FORM
        I = 1                               ; INITIALIZE BUFFER INDEX
10      CALL FINLN ('SCRFLD' , A(1), TERM)  ; GET INPUT FOR SCROLLED LINE
        IF I = 1 AND TERM = 7 OR 9          ; CHECK FOR BEGINNING OF DATA
                                            ; AND SCROLL BACKWARD
        CALL FPUTL ('BEGINNING OF DATA')    ; IF SO, DISPLAY MESSAGE
        GOTO 10                             ; GET LINE AGAIN
        IF I = 50 AND TERM = 6 OR 8         ; CHECK FOR END OF DATA AND
                                            ; SCROLL FORWARD
        CALL FPUTL ('END OF DATA')          ; IF SO, DISPLAY MESSAGE
        GOTO 10                             ; GET LINE AGAIN
        IF TERM = 7 OR 9                    ; IF SCROLL BACKWARD
        I = I = 1                           ; DECREMENT BUFFER INDEX
        IF TERM = 6 OR 8                    ; IF SCROLL FORWARD
        I = I + 1                           ; INCREMENT BUFFER INDEXZ
20      CALL FPFT (TERM, 'SCRFLD')          ; PRECESS TERMINATOR
        CALL FOUTLN ('SCRFLD', A(I))        ; WRITE DATA TO NEW LINE
        GOTO 10                             ; GET INPUT
        .
        .
        .
```

DBMS 10/20 COORDINATOR

Having held this post (DBMS 10/20 Coordinator) since the week or two prior to the Spring DECUS Symposium in Miami Beach, and having been involved with the DECUS Organization and DMS SIG only slightly longer, I find it a mite difficult to address the newsletter with knowledge, confidence, and authority. I am not currently blessed with the "Black Book" so carefully developed by my predecessor so my resource list is limited to those people I have met at the two Symposia I have attended and those who have been mentioned in passing during informal discussions and at get-acquainted-cocktail-party sessions. Now, after baring my soul and off-loading the responsibility for performance in this office, let me offer some ways of using the DBMS 10/20 Coordinator in your site.

My understanding of the position with the structure of the DMS SIG is one of providing a central information and query point through which more information should flow than at the present time. I can not help 10/20 sites who do not request advice, counsel, or assistance in the selection of a DBMS or in the solving of vexing problems. To date my inventory of requests is none received and therefore, 100% successful solutions. I am not offering myself as a DBMS 10/20 guru, though I have slept many nights on the snow covered rocky peaks, but rather as a focal point for information about the product, user experiences (both good and bad), and a distribution point for solutions to problems which more experienced hands have already solved.

My fervent hope is that the paucity of DBMS 10/20 queries I have intercepted means that the software is bullet-proof and running well everywhere. I know better. The installation here at the Nashville Gas Company has experienced some extremely vexing disruptions. Out of humanity and concern for my fellow users I would be happy to share the solutions, such as they are, with others facing similar situations. Perhaps the inaction I witness is because you all do not know I exist. My raucous behavior in the"computerized service lines" during the Symposia and at other functions makes me doubt this excuse, however. So here I stand, ready to serve, DBMS 10/20 Coordinator for the DMS SIG, with little to do. Try me. Not all at once, but a few at a time and let's try to generate more involvement with software problems, successes, and solutions at the user level.

I am trying to retreive the "Black Book". Anyone who would like to serve as a resource for DBMS 10/20 questions, just send me a business card or a note to that effect, and I will happily add you to the too-short list.

A final comment about my tenure in this position. My company management is now pursuing a different path for Data Processing, so that this cheery greeting is also the beginning of the end. After the Fall DECUS in Los Angeles, there will probably be a new DBMS 10/20 Coordinator to deal with. I would like all the DBMS 10/20 users to help me to help the new Coordinator off to a good start with a backlog of requests, and offers of assistance, so that my brief tenure will have had some positive effect.

DBMS 10/20 Coordination and choreography provided by:

Jack Hill
Director, Data Services
Nashville Gas Company
814 Church Street
Nashville, Tennessee 37203
(615) 244-7080 8AM to 5PM Central prevailing Time

DBMS-11: THE LOS ANGELES SYMPOSIUM REPORT

Michael Antin
DBMS-11 Coordinator
for the DMS SIG

Polaroid Corporation
1265 Main Street
Waltham, MA. 02254
(617) 684-5674

Since the old year has passed, and now that Atlanta is only four months away it seems like an appropriate time to think back on the past symposium held in Los Angeles at the beginning of December, 1981.

For those DBMS-11 users who did not attend, I have passed along via the newsletter an excerpt of the handouts from the DBMS-11 Technical Tutorial. The presentation was made by Ann Harrison of Digital. She discussed some of the new features of the next release which should be available for all three operating systems (RSX-11M,RSX-11M PLUS, and IAS) by the time this report comes out in the newsletter. Several of the items that were submitted through the DMS SIG menu in the Spring of 1981 were incorporated into the latest releases. Several others were discussed at the Technical Tutorial and are summarized in the handout.

The primary topic of the session was the structure of the data dictionary. Ann Harrison provided insights about the dictionary contents and illustrated a straight-forward technique using DBQ for accessing the DBMS-11 data dictionary. It requires a little work: a subschema compile and a few manipulations of the redirection table.

If you have doubts about what you would gain from peeking into the data dictionary with DBQ, perhaps you would like to know what the structure of the dictionary looks like. You can see for yourself ! I have included six diagrams from the handout which show all the record and set relationships for the data dictionary. VOILA ! Note that in the interest of saving DECUS some money, I have not included the 30 page data dictionary schema source listing which was also part of the handout. I will be glad, however to send a copy to anyone who would like to receive it.

As always, I am eager to hear from all DBMS-11 users. If you have a problem, I will do my best to put you in contact with other users who may have experienced similar difficulties. Even if you do not have a problem or a question. I would like to know who is using DBMS-11 out there in the cruel world.

DBMS-11

TECHNICAL SESSION

Ann Harrison
Digital Equipment Corp

I. Version 2.0 Announcement

II. DMS SIG menu response

III. Data Dictionary Topics

I. DBMS-11 Version 2.0 Annoucement

Available in mid-January

For RSX-11M, RSX-11M-PLUS, IAS

Replaces Versions 1.6, 1.7, & 1.8

→ Major Features

Multiple databases

Datatrieve-11

Journalling control

→ New features since Version 1.8

DBQ - datatypes
repeating groups

DBO - New volume on journal

Exit status from all utilities

Subschemas and CALC records

Documentation

Security

DMS SIG Menu response

→ Fall 1981 List Response:

- Alias *
  FDML translates names
- Performance *
  Timer and documentation
- Restructure Utility *
- Data Dictionary *
  Covered later this session

* -> I will be in the camp ground
to talk about these questions

DMS SIG Menu response

Fall 1981 List Response:

→ Describe the implications of using a single thread system.

Two attributes that make it single threaded:

Lack of concurrency control

All access is done through a single program (DBM)

→ IAS user information:

Operator/DBA interface changed

DML programs are restricted to
56 KB

Version 2.0 uses more node pool
than Version 1.7 even for
a single database system

II. DMS SIG Menu response

→ Good news from last session:

Datatypes: DBQ supports them all

Performance measurement:
DBQ timer
Described use of statistics

Journalling: End of Volume

DMS SIG Menu response

Concurrency Problems:

Recovery
Undetected interference can corrupt data and pointers

Implications:

Data integrity is reduced in concurrent update mode
Programs which don't share data can run safely

All requests channel through DBM

Problems:

Buffer Contention
Other requests wait for I/O to complete
Context switch time

Implications:

Programs that run with others that are I/O intensive suffer

All solutions are very costly in address space and physical memory

DMS SIG menu response

Data Dictionary

Expand the data dictionary to include comments from the schema description subschema, DMCL, and the DBA. Provide a utility to extract this information.

Some of it is there now:

Comment entries from Schema

DBREPS subschema report

Data Dictionary Reports:

DMCL report:
- all areas defined in the schema
- every DMCL and what it contains

Range map:
- page range of each area in schema
- page range of each record within the area

Schema record description:
- all declared attributes plus
- actual DBKey positions used
- length including prefix
- starting position of each field
- and more

Schema set description
- All sets defined in the schema
- And the CALC set
- Owner, member, and set characteristics

Schema procedure report
Schema protocol report
- Information stored by DBCLUC

Subschema data dictionary listing
- Records from the subschema with
- Data type, offset, byte length, character length, picture
- and COMMENT

Even more Data Dictionary Reports

Subschema Record Description
Subschema Set Description

- Appropriate subsets of the equivalent schema reports

- Plus privacy lock information

III. Data Dictionary Topics

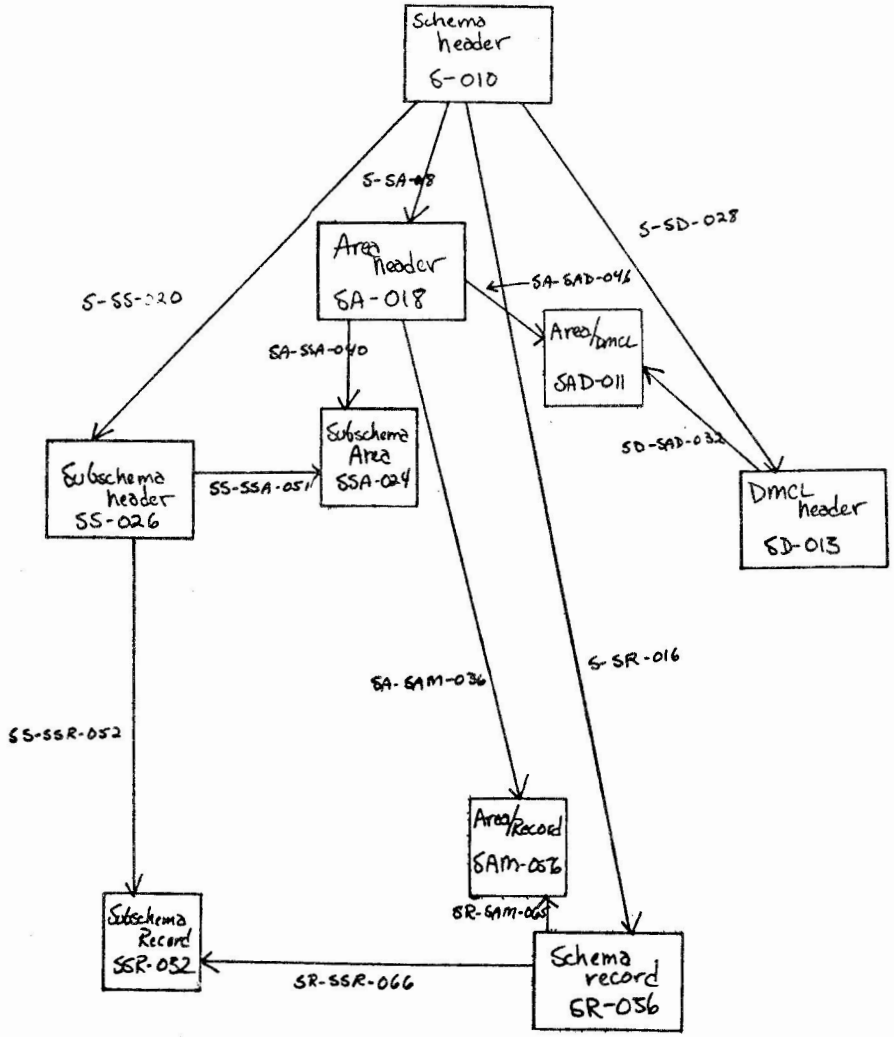
Interactive Dictionary Inquiry Tool

Your old friend DBQ can do it!

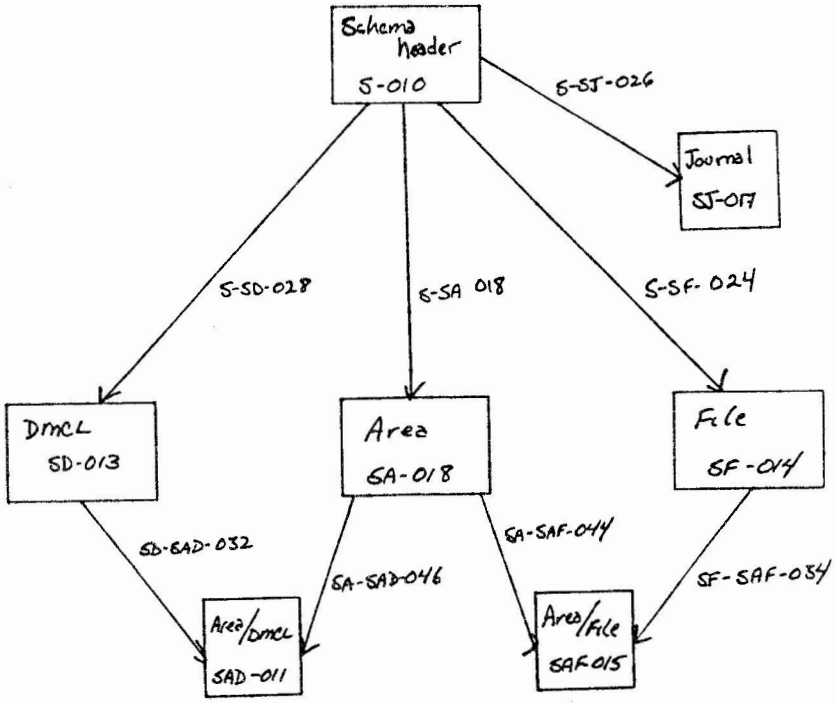
Procedure:

- Define a new redirection table entry
- Create a dictionary identical in size to the target
- Compile NETSC.SCH from the kit
- Create a subschema NETSS.SSC which includes all records and sets
- Compile that subschema
- DBQ> DB NETSS
- Remove the new redirection table entry, map to target database
- DBQ> READY

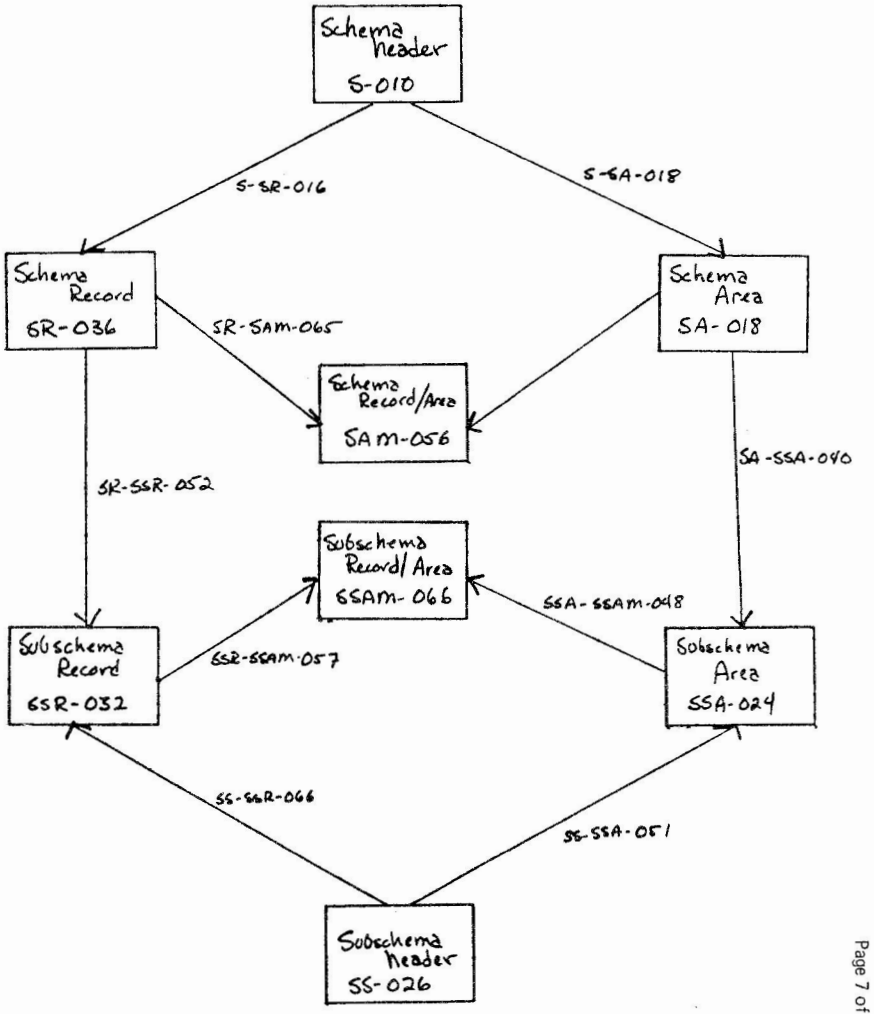
Central Dictionary Structure

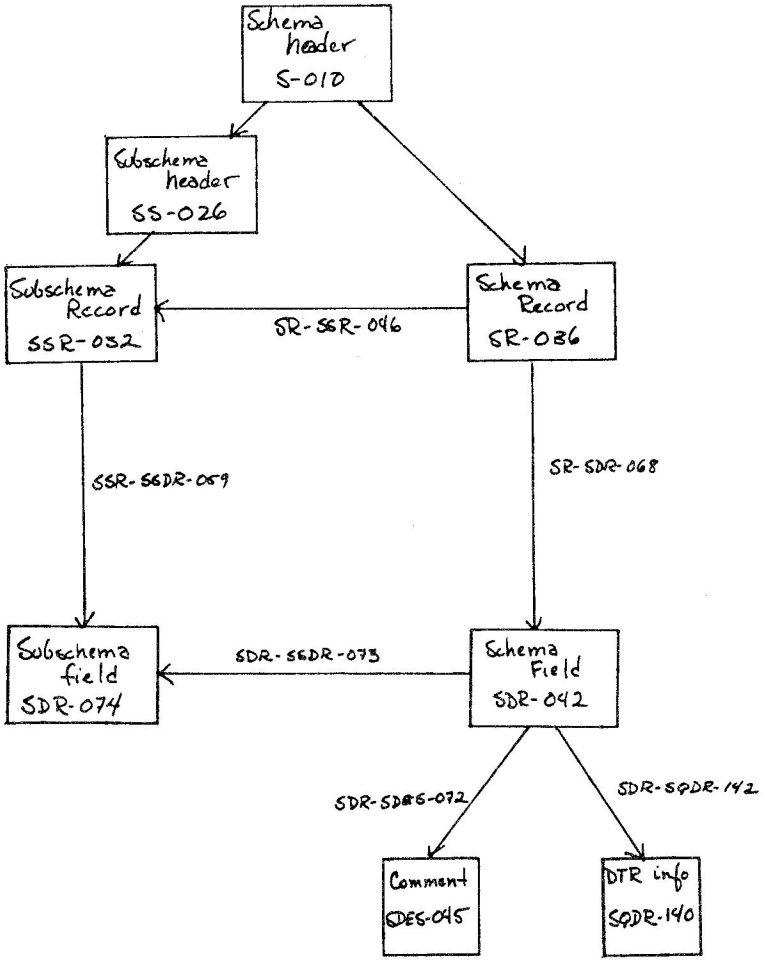


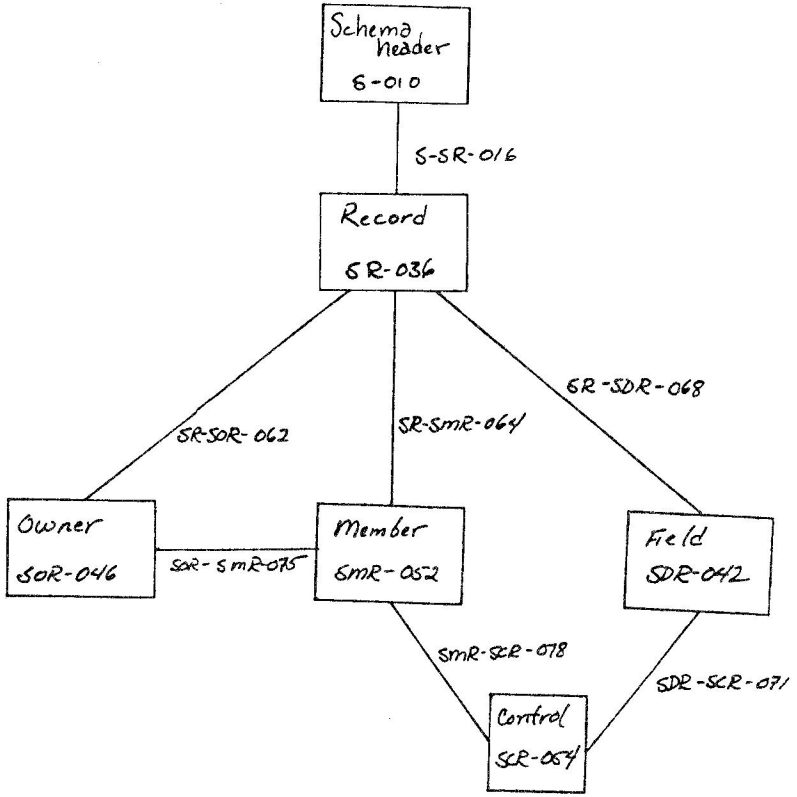
Areas, Files and Dmcl's

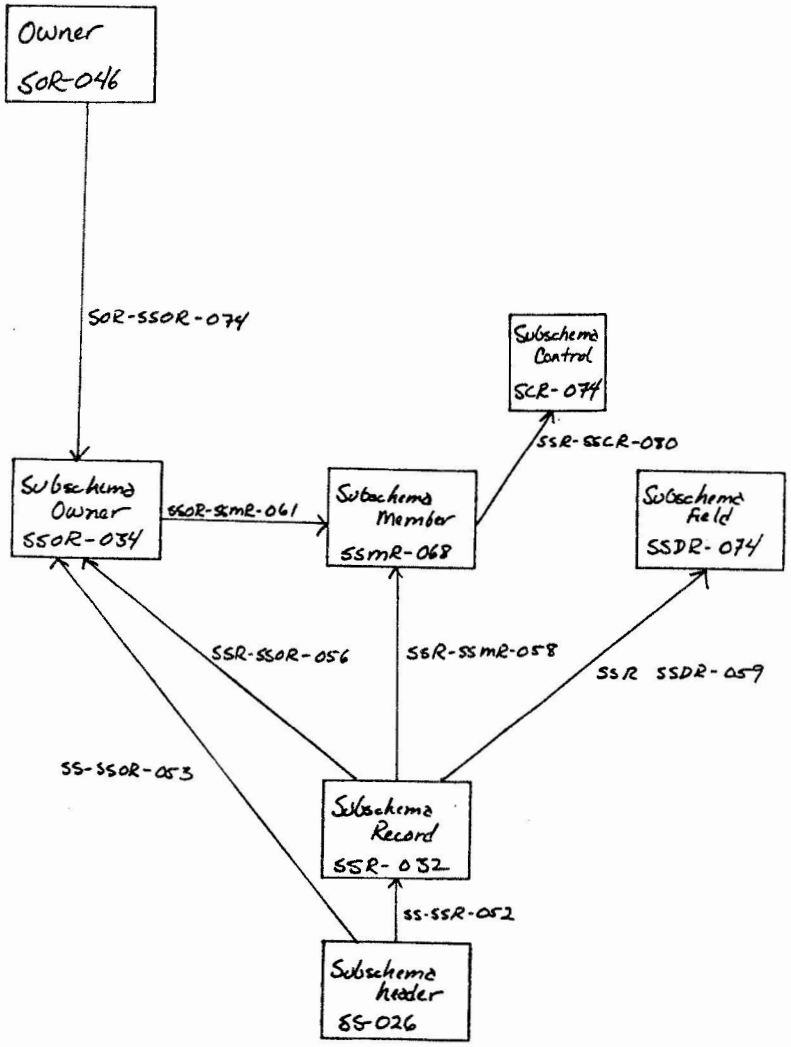


Areas and Records




Schema header S-010
Subschema header SS-026
Subschema Record SSR-032
SR-SSR-046
Schema Record SR-036
SSR-SSDR-059
SR-SDR-068
Subschema field SDR-074
SDR-SSDR-073
Schema Field SDR-042
SDR-SDES-072
SDR-SDDR-142
Comment SDES-045
DTR info SDDR-140


Schema header S-010
S-SR-016
Record SR-036
SR-SDR-068
SR-SOR-062
SR-SMR-064
Owner SOR-046
SOR-SMR-075
Member SMR-052
Field SDR-042
SMR-SCR-078
SDR-SCR-071
Control SCR-054



SUBSCHEMA SET REPRESENTATION

MEMO FROM THE NEW TECO NEWSLETTER EDITOR
GREG STEINKUHLER
SEPTEMBER 30, 1981

The TECO SIG is attempting a comeback. We're doing this by establishing communications with current TECO users and interested future TECO users. Expertise is definately not a requirement for participation.

We need your help and your contributions. Do you have questions? Send them to us anonomously if desired. We'll answer everything we can, and what we can't, we'll throw out to the user community. On all correspondance include the operatiing system and the TECO version number at your site (CTRL V=$$ or EO=$$ will give the version number).

Do you have a favorite macro? Let's have it and we'll all share the wealth.

By the way, did you know that TECO is up to version 36 (for PDP-11, DEC-10 is up to version 3 and PDP-8 is up to version 7) and that it comes with a page editor macro called VTEDIT?

VTEDIT requires no knowledge of TECO commands for its usage!!! But, if you do know TECO, you now have character editing, macro editing, and page editing at your fingertips!!!

With version 36, TECO has, once again, proved itself to be 'The Programmable Editor'. Here are some highlights:

-the ' ' statement now gives TECO the ELSE capability within the 'if' block;
-the 'F' statement provides for greater FLOW control within 'loop' and 'if' blocks.

TECO has advanced to the point that someday a TECO guru will emerge who, in his madness, will produce the ultimate --- the TECO operating system (maybe someone already has and called it UNIX!). To his honor we dedicate the following:

TECO GURU OF THE MONTH

by Franklin Reynolds
of TRT Telecommunications

Send all 'TECO GURU OF THE MONTH' drawings, questions, and macros to:

GREG STEINKUHLER
TRT TELECOMMUNICATIONS
P.O. BOX 8876
FT. LAUDERDALE, FL.
33310

DATAGUARD CORPORATION
15 SPINNING WHEEL ROAD
HINSDALE, ILLINOIS 60521
312-789-2277

September 21, 1981

Robert F. Curley
P.O. Box 332
Flourtowm. PA 19031

Dear Mr. Curley:

I have enclosed a sample run and a listing of a TECO utility which has proven quite useful to our BASIC+2 RSTS programmers.

Since we are consulting on several different systems, it is convenient to insure our BASIC+2 modules are compiled with the desired switches. For instance, it is quite frustrating to discover that one of your modules was compiled with the wrong math package after a lengthy task build.

In order to eliminate these sources of errors and improve morale at the same time, I have developed the enclosed TECO utility. This utility will compile the last file editted with TECO, provided it has an extension of ".B2S".

As the utility is well commented, it can be easily modified to do any set sequence of commands to the last file editted with TECO.

Sincerely,
Mark J. Diaz
Mark J. Diaz

Sample run (all input actually typed is underlined)

```
TECO A.B2S/72
*EX$$
Ready
MUNG BP2
Ready
BP2
PDP-11 BASIC-PLUS-2 V1.6 BL-01.60
BASIC2
OLD A.B2S
BASIC2
COMPILE/NOLINE/NOCHAIN/OBJECT/DOUBLE
BASIC2
EXIT
Ready
Ready
```

```
!* TECO utility: BP2.TEC
   This utility will execute the ATPK CCL, which will compile the
   last file editted if it has the extension ".B2S".
   It is executed with a MUNG BP2 monitor command or an "EGBP2$$" command
   from TECO.
*!
                        !* Build a 2 digit leading zero filled Job number *!
(EJ-10)"L               !* Is the Job number less than ten? *!
        I0$             !* Yes?, then we need a leading zero *!
       '
EJ\$                    !* Insert the Job number *!
(.-2,.)XJ               !* Store the Job number in Q-reg. J *!
(.-2,.)D                !* Delete the Job number from the text buffer *!

                        !* Build the TECFJJ.TMP file open *!
I:ERTECF$               !* Open read, "TECF" + *!
GJ                      !* + Job number + *!
@I\.TMP$\               !* + ".TMP$" *!
IUO$                    !* Return the open status in Q-reg 0 *!
HXO                     !* Store the TECO "memory" file open *!
HK                      !* Clear the text buffer *!
MO$                     !* Attempt the TECO "memory" file open *!
GO"U                    !* If the open was unsuccessful, report this and abort *!

   ^ACan't find TECO memory file - Aborting^A^C^C
  '
Y                       !* Read in the TECO memory file *!
J                       !* Get to the beginning of the buffer *!
:S.B2S$"U               !* If the last file editted does not have *!
                        !* an extension of ".B2S", report this and abort *!

        ^ABASIC+2 not last file editted - Aborting^A^C^C
       '
J                       !* Get to the beginning of the text buffer *!
:S/$"S                  !* If there are any switches delete them *!
     (.-1,Z)D
    '
HXF                     !* Store the BASIC + 2 file specification *!
HK                      !* Clear the text buffer *!
                        !* Build the indirect command file open *!
I:EWBP2F$               !* Open for output, "BP2F" + *!
GJ                      !* + Job number + *!
@I\.TMP/MO:1536$\       !* + ".TMP"; Create at top of UFD *!
IUO$                    !* Return open status in Q-reg 0 *!
HXO                     !* Store indirect command file open *!
HK                      !* Clear the text buffer *!
MO                      !* Attempt the indirect command file open *!
GO"U                    !* If the open was unsuccessful, report this fact and abort *!

   ^AUnable to create indirect command file - Aborting^A^C^C
  '
                        !* Build the CCL command stream for ATPK *!
IEG$                    !* The exit to moniter and issue moniter command *!
I@$                     !* The ATPK CCL (could vary by installation) *!
I BP2F$                 !* "BP2F" + *!
GJ                      !* + Job number + *!
@I\.TMP$\               !* + ".TMP" *!
HXG                     !* Store the ATPK CCL command line (@ BP2FJJ.TMP) *!
HK                      !* Clear the buffer *!

                        !* Build the indirect command file *!
IBP2
OLD $ GF I@
          COMPILE/NOLINE/NOCHAIN/OBJECT/DOUBLE
EXIT
$
MG                      !* Exit to monitor, forcing ATPK CCL command line *!
$$
```

1

VAX INFORMATION ARCHITECTURE OVERVIEW

The VAX information architecture is made up of a highly integrated set of information management products that are supported by the VAX/VMS operating system. These products were developed on the principle that a typical user needs a variety of approaches to meet all of his or her information management needs.

With VAX information architecture, different users, different departments, different applications, and different VAX systems can have different file structures. Yet these different file structures can all be accessed through a single set of consistent commands.

Because the components of the architecture are arranged in layers above the operating system, it is possible for the components on one level to use the facilities of the other components.

On the top level, the VAX languages and VAX-11 FMS (Forms Management System) provide a user interface for interactive and language-callable video forms. VAX-11 DATATRIEVE supports English-like queries, hard-copy reports, and graphics. On the next level is the VAX-11 Common Data Dictionary, which integrates the other components of the architecture, and the VAX-11 DATATRIEVE high-level and distributed data access facilities. The lowest level consists of two multi-user, data management facilities: VAX-11 RMS (Record Management Services) and VAX-11 DBMS (Database Management System).

Figure 1 shows these components as a series of interlocking building blocks that fit into a well-defined software structure.

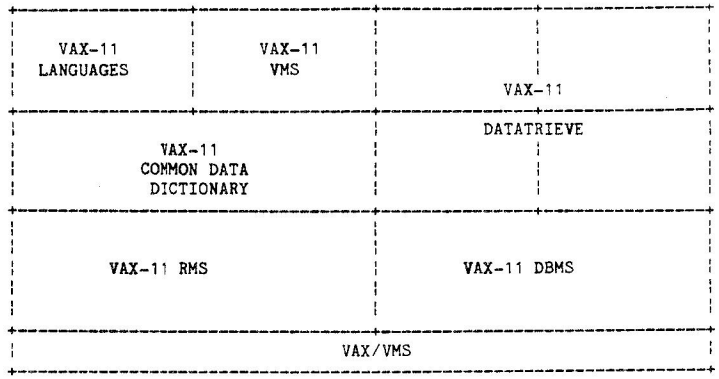


Figure 1: The VAX Information Architecture

This modular design offers several benefits. You have the flexibility to choose the appropriate solutions for both central and departmental applications. You gain an increase in control over your information, which in turn provides increased integrity and security. More efficient data processing tools mean increased programmer productivity. Finally, you can protect your current software investments while building a foundation for future growth.

The following sections focus on the features and benefits of three components of the VAX information architecture.

VAX-11 COMMON DATA DICTIONARY (CDD)

When data is managed with conventional file or record management methods, the programmer must include a description of the data and how it will be used in the logic of the program. To use the data effectively, the data processor must understand how the program is written and how the data is stored. In addition, storing data definitions in programs can lead to different definitions being created for the same data, which increases the likelihood of data redundancy and inconsistency.

The Common Data Dictionary, or CDD, provides a major step towards eliminating redundant data definitions by serving as the central VAX/VMS storage facility for data descriptions shared by VAX-11

DBMS and DATATRIEVE. The CDD stores only the data descriptions, not the data itself.

The CDD is organized as a hierarchy of **dictionary directories** and **dictionary objects**, similar to the VMS directory system. This structure allows different users to organize their portions of the dictionary according to their needs. Figure 2 shows how an organization might set up its directories under the topmost directory, CDD$TOP.

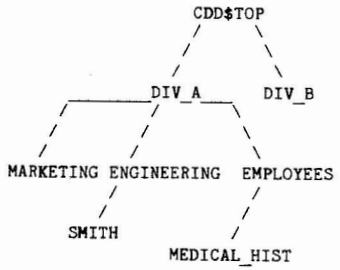


Figure 2: Example of CDD Hierarchy

In this example, Division A and Division B do not have access to, or even awareness of, the other's portion of the CDD. The departments within Division A -- marketing and engineering -- have their own sections of CDD space and share access to some directories, such as "EMPLOYEES". Smith, an engineer in Division A, would have access to those portions of the CDD storing the data descriptions he needs to do his job. The directory hierarchy allows him to arrange those definitions in whatever way he finds most useful. Each group using the CDD can use access control lists to protect directories and prevent unauthorized users from reading, updating, or deleting stored definitions.

The CDD structure, therefore, permits flexibility on an organizational, departmental, and individual level. In addition, because programmers no longer need to embed data definitions in their programs, the CDD provides data independence and data integrity.

You manage the CDD with the **Dictionary Management Utility (DMU)**. You can use DMU to backup and restore the CDD, to create and delete CDD objects, to create and delete dictionary directories, and to create and delete control lists.

VAX-11 DBMS

VAX-11 DBMS is a CODASYL-compliant general purpose database management system based on the March 1981 Working Document of the ANSI Data Definition Language Committee. It provides multi-user support with data security and performance features needed for large-scale applications. At the same time, its ease-of-use features make it suitable for developing small- and medium-scale databases.

A database consists of **database storage files** and the **database root file**. The storage files are for storing database records. The root files contain data definitions used by the **Database Control System (DBCS)**, the run-time controller of VAX-11 DBMS. The major functions of the DBCS are to monitor database usage, act as an intermediary between VAX-11 DBMS and VAX/VMS, and manipulate database records on behalf of user programs. A separate VMS process, the **DBMS Monitor**, controls access to the database.

A major feature of database programming is that **records** can be directly linked in meaningful relationships called **sets**. If you were designing an inventory application, for example, you would want the relationship between the parts you keep on hand and the supplier who provides those parts to be accurately represented in the database. Because VAX-11 DBMS is a network model database management system, you can define set relationships between any records in the database, not just those sequentially or hierarchically above and below the record.

The **schema**, **storage schema**, and **subschema** contain the definitions for records and relationships between those records. The schema and subschema describe the data characteristics and relationships of the database; the storage schema defines the physical structure. You use **data definition languages**, or **DDLs**, to write the three schemas. These are then compiled and stored in the CDD.

There are several ways to process data stored in VAX-11 DBMS databases:

- The interactive DML (data manipulation language) utility, DBQ
- Callable DBQ, which allows all languages that conform to the VAX/VMS Calling Standard to access a VAX-11 DBMS database
- VAX-11 DATATRIEVE

Figure 3 shows how these various components interact at run time with the database and the CDD. (DBCS is the VAX-11 DBMS Database Control System; DBQ is Database Query.)

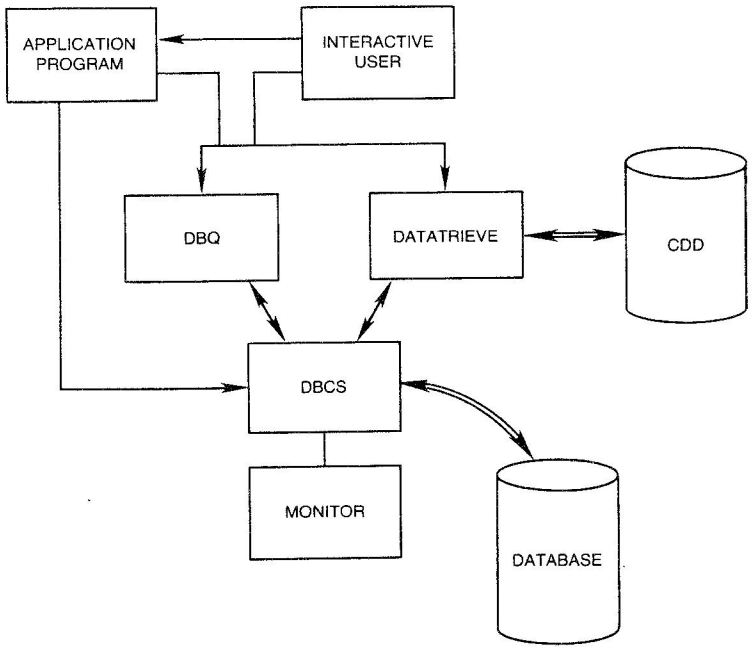


Figure 3: Run-time Interaction of CDD, DATATRIEVE, and DBMS

The **DBO (Database Operator) utilities** provide a variety of system procedures to help operate and maintain databases. These procedures include database alteration, verification, reporting statistics, and backing up and restoring databases. DBJ and DBR are separate processes that handle journaling and recovery functions. Certain DBO operations start these processes automatically.

VAX-11 DATATRIEVE

VAX-11 DATATRIEVE is a data management tool for inquiry, update,

and maintenance of information stored in databases. It has the flexibility to meet the needs of casual users and professional data processors. DATATRIEVE can serve as a single, high-level interface to RMS files and VAX-11 DBMS databases. As a result, if an application changes or a new one is added to the system, data processors do not need to learn new skills.

Figure 4 focuses on the DATATRIEVE section of the illustration used in Figure 1 to highlight DATATRIEVE's role in the VAX information architecture.

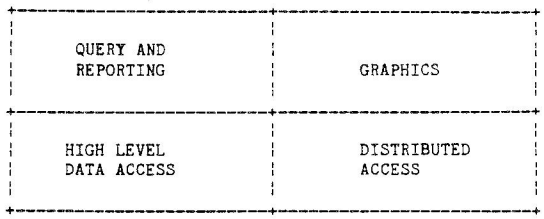


Figure 4: VAX-11 DATATRIEVE Components

Entire applications can be written in DATATRIEVE, or programs can use the VAX-11 DATATRIEVE Call Interface to access data and to produce reports. A program accesses DATATRIEVE similarly to the way an interactive user does. Programs pass command lines to DATATRIEVE and receive back records, print lines, and messages. Once the programs have retrieved records from DATATRIEVE, they can perform complex statistical analyses, other varieties of large-scale computations, and complicated report formatting; they can also produce data and use it to modify and store DATATRIEVE domains. The Call Interface, therefore, allows you to combine the data manipulation power of DATATRIEVE with the computational strengths of programming languages such as COBOL, BASIC, and FORTRAN.

The VAX-11 DATATRIEVE interface to VAX-11 DBMS lets you query, report, and manipulate data in CODASYL databases created and managed by VAX-11 DBMS. The DATATRIEVE **record selection expression** contains optional clauses that let you work with database set relationships. To allow database-specific operations, several DATATRIEVE verbs are reserved for use with VAX-11 DBMS. In addition, the DATATRIEVE **Context Searcher** helps you simplify your record selection expressions when you work with complex set relationships.

The DATATRIEVE **Report Writer** helps you display data in easy-to-read formats. You can display a report on your terminal,

print it on a hard-copy printing device, or store it in a VAX-11 RMS file for display or printing at a later time. The DATATRIEVE Editor closely resembles the VAX-11 EDT Editor. It lets you edit in either line or character mode and use keypad or nokeypad commands. With the Editor, you can change the CDD definitions of procedures, domains, records, and tables, and you can correct errors that result from faulty typing, syntax, or logic in your DATATRIEVE commands and statements.

The DATATRIEVE interface to VAX-11 FMS, a forms management utility, lets you display, modify, and store records with a video terminal whose screen is controlled by FMS forms definition. If a form name is used as part of a DATATRIEVE domain definition, DATATRIEVE automatically uses the form to collect or display the associated data.

With DATATRIEVE **Graphics**, you can generate several different types of graphs using a VT125 terminal. The graphics include pie charts, histograms, scatter plots, and time series. In these charts and graphs, you can plot field values against other field values, against frequency of occurrence, and against dates.

The **Distributed Data Manipulation Facility (DDMF)** allows you to retrieve data from remote VAX-11 DATATRIEVE nodes through DECnet communications software. This capability makes it possible to use DATATRIEVE to retrieve data on remote VAX systems, whether that data is stored in RMS files or DBMS databases.

DOCUMENTATION FOR THE VAX INFORMATION ARCHITECTURE

Documenting as diverse and complex a product set as the VAX information architecture offers many challenges, in particular because each component can be used alone or in combination with other parts of the architecture. The conventional user's guide/reference manual approach seemed too limited for products with a wide range of capabilities that would be used by professional data processors, programmers, and casual users alike.

Hence, the documentation sets described in the next three sections not only document software features, they address the tasks you may want to carry out with the software. In addition, each documentation set offers comprehensive explanations of data management concepts, complete descriptions of command syntax, and cross-references to sources of further information.

The documentation directory that follows can guide you to the manuals that describe the VAX information architecture components that interest you most. Each summary explains the purpose of the manual, describes its intended audience, and lists the manual's

concepts and features. Order numbers are listed at the beginning of each section; to order manuals, see the last page of this booklet.

Word Processing Passages

Paul D. Clayton

We are using a WS248 in conjunction with a PDP 11/70 for mass storage of all documents. The software package we use is DX/IAS which was written by DEC. In using the software we have encountered a number of problems that when reported to DEC fall on deaf ears. I am using this newsletter as a means of spreading the word and providing a forum for future passages by any interested parties. Below is a list of the changes we have made to the DX/IAS software here at Naval Air Development Center, Pa.. If anyone is interested in how we did what we did, feel free to call and we will see what we can work out to pass information back and forth.

FILES MODIFIED:

WP8DIR - 1) ADD IN CODE FOR PAGE HEADERS TO BE PRINTED ON THE TOP OF EACH SUCCESSIVE PAGE.

WP8PIP - 1) MODIFY CODE TO STORE/RETRIEVE ALL THE PRINT SETTINGS OF A A DOCUMENT. THE REPLACEMENT CHARACTERS WERE BEING LOST IN TRANSFERRING A DOCUMENT BACK AND FORTH.

WP8LPT - 1) ADD IN COMMENTS FOR DEFINING WHAT THE POSITIONS IN THE DOCUMENT HEADER ARE.
2) MOVE THE POSITION OF THE 'LN' (DIAGNOSTIC LINE NUMBERS) IN THE HEADER. THIS ELIMINATES THE CONFLICT WITH THE 'SE' (STOP BEFORE EACH PAGE) OPTIION.
IMPACTED MODULES: WP8LPT,NEWSET,DISPLA
3) PUT IN CODE TO INITIALIZE THE PRINT DARK VARIABLE 'DA'.

FILES ADDED:

WP8CHG - 1) REWORK OF 'WDE' TO PROVIDE THE CAPABILITY TO CHANGE A DOCUMENT NAME. UP TO 64 CHARACTERS CAN BE ENTERED. NO CHANGE TO ANY OTHER DATA IN A DOCUMENT HEADER.
IMPACTED MODULES: WP8CHG

???TKB.CMD - ARE NEW TASK BUILD COMMAND FILES FOR ALL THE DX PROGRAMS WHICH MAKE USE OF THE FORTRAN IV RESIDENT LIBRARIES. THIS SAVES APPROX. 10KW OF CORE FOR **EACH** COPY OF A DX PROGRAM WHEN IT EXECUTES. CORRESPONDING CHANGES HAVE BEEN MADE TO 'DXINS.CMD' TO USE THESE FILES.
IMPACTED MODULES: DXINS.CMD

WP8SPE - 1) This is a spelling DETECTION program that will take a WPS file or a standard host text file and check it for unknown words. A listing is generated with all unknown words underlined for easy locating of errors. The dictionary we have is approx. forty-four (44) thousand words long. There is a means to add words to the dictionary as new ones are found and verified. The host operating system must have RMS-11K avaible for this program to run.

WORD PROCESSING WITH DEC COMPUTERS
ADVANCED PRACTICAL HINTS AND TECHNIQUES

Lawrence H. Eisenberg
17141 Nance Street
Encino, California 91316

ABSTRACT

This paper discusses various handy hints and kinks associated with the use of Word Processing Systems on the PDP-8 and PDP-11 systems. While developed primarily with WPS-8, the routines and hints offered generally are applicable with WPS-11 and other commercially available systems currently utilizing the WPS-8 format. The discussion presented below consists of the various matters presented at the Spring 1981 Symposium in a Panel Discussion with Vicki Ann Rose, Manager, Marketing Development and Research -- Word Processing and Scott Shulga, Manager, Market Planning and Development -- Word Processing of Digital Equipment Corporation, Merrimack, N.H. and Lawrence H. Eisenberg.

"USING YOUR INDEX" APPLICATION

Will Allow You To:

* CREATE OPERATOR STATISTICS
* LOG FILE/DOCUMENT LOCATION
* GENERATE TABLE OF CONTENTS

INTRODUCTION

The following is a "super" application for developing a file (or document) locating system, to generate Table of Contents, or record operator statistics. It is based on "adding fields" when first creating your documents (i.e., by adding the "field" names to the name of your document, you create fields which may be referenced in your index). It has numerous uses!!! It will allow users to build upon their index. Information such as author <a>kac, date <d>10/10/80, typist <t>db, location of document <loc>RL2:SECT10, can be included along with the title. The C denotes Create from the MAIN MENU.

Example: C "Using Your Index" <a>kac <d>10/10/80 <t>db <loc>RL2:SECT10

[When you view the index of your documents, everything following the left angle bracket will not be displayed, but may be referenced using list processing. Read on ---]

Another use may be to generate Table of Contents by typing in titles, page numbers, etc. The index then can be used as a list document in list processing and the form can be whatever the you require. A field can be included to indicate that this is a title line <tl> and used in list processing, those not containing <tl> will not print (if you don't define it to be printed).

The really neat thing is that the system automatically assigns the fields <n>document name, <#>document number and <>end of record. Other data needed can be created within the document name. The limit for titles is 68 characters. The other feature of this procedure is that when "I" for index is typed, only the information to the first "<" is displayed.

The other information is stored in document 1, the system index. Please note that it is not necessary for the list document to be vertical. The system looks only to the next "<" for the next field.

APPLICATION PROCEDURE

The main idea of modifying the index file generated by Digital's word processing system is to create a more detailed file indexing scheme. This is accomplished by including parameters in a list processing format in each record of the diskette index file.

The term "list processing format" means a variable name enclosed in brackets "<" and ">" usually with a value following the closing bracket.

e.g., <n>manual A.

These fields or parameters can be edited directly into the system generated index file, document file one (1), or when a new file is created on a diskette or RL0, the parameter can be included in the file name. For example, in creating a normal non-modified file the format would be:

C filename

But with the modifications it could look like:

C filename <field1>value1 <field2>value2 <field3>value3

An example of its use would be:

C Letter to John Jones <a>kac <d>10/10/80 <t>db

This would translate as:

C	=	Create Letter to John Jones
<a>	=	author, kac
<d>	=	date, 10/10/80
<t>	=	typist, db

BENEFIT

Modifying the file index without altering the normal system created values does not hamper the original

intent of the diskette index. When the file is deleted the parameters added to the index file are also deleted. The only requirement when passing these parameters to the index file is that one space must follow the filename. Spacing between the other added fields is not necessary.

Because our index is a document, fields can be added or deleted at any time. If it is necessary to exceed the 68 character limit - keep in mind that this restraint is only during CREATING titles, once document one is called up for editing, the one line limit no longer applies. That is, you may add as much as you feel is necessary to the description in document one, so long as you do not disturb the required sequence following your descriptive matter.

On Digital's word processing system the index is created automatically and maintained by the system for each diskette with one index file per diskette. When a file name is created an associated document number is assigned by the system and that newly created file can be accessed by either the file name or document number. Upon inspection of this system index file the information was stored in a list processing format of:

<n> TITLE OF EACH DOCUMENT <#>2<>

Recognizing the fact that the '<n>' is the name of the document and the '<#>' is the associated document position on each diskette.

An overview of list processing is a method of processing variable names and the data associated with each name against a standard form letter and producing a separate letter for each grouping of variable names. A common application of list processing is where a form letter is created for each address of a mailing list. Sorting and selection criteria may be performed on the list processing file. In the example of the mailing list the addresses could be sorted on zip code and then only one state could be selected to produce the form letter.

By modifing the system created index, which by DEC design is always document number one, more variable names and values may be added without affecting the indexing concept of the word processor. Thus, the following variable names (or parameters) and values could be entered for each document in the index file.

Parameters	Values
<flpy>	Diskette Number
<b>	Manual Number
<s>	Section number of the manual
<SS>	Sub-section number

Now, a file exists that contains the desired information and can be copied, combined with other indexes to produce a master index and printed using list processing.

To illustrate how these parameters are entered for a new file for Manual 1, Section 3, Sub-section 5 on diskette number 12 the command would be:

C NEW FILENAME <flpy>12<b>1<s>3<SS>5

This command with a space after the file name establishes the parameters in the diskette index for each file created by this procedure. Several functions now may be performed on this modified index file. Producing sorted indexes according to the desired formats and printing selected file information is the biggest advantage. For example, an alphabetical list of file names for one diskette of a documentation manual may be produced indicating the diskette number; manual number, section and sub-section; and document position on the diskette.

ADVANTAGE

All of the above procedures have been designed in order to ease the task of indexing, editing, printing, cross-referencing, and filing of word processing documentation.

Using your index in this manner allows the user to:

** Record Operator Statistics (author, date, typist, etc.)

C Mechanical Procedures Manual <a>kac <d>10/10/80<t>db

** Record File Locations (diskette #, RL#, etc.)

C Mechanical Procedures Manual <d>12A or <rl>2<sec>10

** Generate Table of Contents (code in titles, page numbers, etc.)

C MPM <tl>Mechanical Procedures Manual <pg>12 <chart>III

** Break down documents for ease of editing by section, page, etc.

C MPM/1 <pg>10 4

** Keep track of correspondence by topic, case, KEYWORDS.

C J.Jones <c>#12568-80 <re>litigation <k>drunken driving

** Use the word processor for a database in KEY areas by coding in keywords and then sorting on those needed.

** If a document changes - <r>revised dates can be added.

** If it is a master <m> document - it can be denoted.

** Using Document 1 - Allows for index flexibility because it can be edited. Information can be added deleted or changed.

Adding <fields> + values to document one (1) can give the user a lot of flexibility in Table of Contents generation, archival information, allow to sort on <kw>key words, <re>topics, <gc>general correspondence, add <r>revised dates, sort on date for deletion or archiving, and in conjunction with list processing and the wild card <*> feature, opens a lot of areas. TRY IT!!!!. All that is needed are a few fields that they require and can be adapted to their needs. It is NOT necessary to have the fields identical in the index, the system will bypass those it does not need and use only the ones specified. If you are looking for the same information, it must be coded in that matter. If <re> is used for regarding, it must always be <re> and not <RE> or <R>.

All the the normal list processing rules are in effect.

!!!!!!!!! IDEAS !!!!!!!!!!

Have you ever been asked if you could track any of the following? Using this procedure, NOW YOU CAN.

<t>typist	<tl>title line
<a>author	<flpy>floppy number
<d>date	<sec>section number
division	<dod>due out date
<di>date in	<p>page number
<l>letter	<ta>turn-around time
<w>what	<m>memo
<c>ase title	<re>regarding: topic
<at>attorney	<c#>case number
<r>revised date	<rep>response to
<kw>keywords	<info>sent information
<dist.>district	<srep>sales rep

Use a little imagination and you have indexing by page, section, table of contents generation, tracking of correspondence, turn-around times, revised dates, logon information, storing of keywords, a list of who to send literature to, product announcements, etc.

PLEASE NOTE: If this procedure is going to be used, especially by new operators, do a "GO GET DOCUMENT" and make a back-up of the index document (#1) to protect file information. Remember that document 1 is the system generated document and cannot be deleted or the Index will be lost. To recreate a lost Index, however, you may edit each document by document number and pick up the document name. You then may recreate the Index, using the field identifiers normally created in the Index.

LIST PROCESSING HINTS

FIELD IDENTIFIERS AND DATA PROCESSING

The "lists" which are developed in list processing often are useful for data processing activities as well as many of the Word Processing and List Processing purposes. For PDP-11 users, many of the data files developed under Word Processing may be addressed directly by data processing. However, for PDP-8 users the WPS-8 files (which are saved in a format similar to COS-310) cannot be addressed directly by COS-310 or OS/8. While the WS-200 series originally was disigned to provide for direct communication between Word Processing and COS-310, this feature no longer is supported and the WS-200, as with all other WPS-8 systems, requires conversion to utilize the files. (Conversion utilities for WPS-8 to both COS-310 and OS/8 are available through the DECUS LIBRARY. These utilities transfer list processing type files between the various systems. The conversion procedures are not discussed in this paper.)

It is most helpful, therefore, to maintain the LIST FIELD IDENTIFIERS as upper case characters. While the DEC WPS manuals show the field identifiers (e.g. - <field1>) as lower case fields (indeed, the entire article, above, uses lower case), such was not meant to be a required form for identifying the fields. The use of lower case by DEC was a throwback to computer manuals which used lower case to indicate operator decisions, as opposed to upper case which indicated mandatory acts.

Since each of the WPS-8 systems utilizes special characters to indicate lower (and upper) case shifts, any conversion program is going to require considerable additional (and wasted) time in order to perform the conversion, as each of the special characters will have to be stripped from the field before the data can be used by the data processing system.

If there is even the remotest possibility that your list files will be used in data processing, it is important to avoid the use of hard [the RETURN key] returns except at the end of a field identifier. In other words use one identifier for every line of text. For example:

DO NOT USE

<NAME>John Doe
<ADDRES>123 Any Street
Our Town, U.S.A.
00123

DO USE

<NAME>John Doe
<ADDS1>123 Any Street
<ADDS2>Our Town, U.S.A.
<ADDS3>
<ADDS4>
<ZIP>00123

In many conversion programs, and nearly all data processing programs, the carrier returns within a field may be read as a terminator, and the information following the return may be lost during the conversion or use by the program.

While the use of several fields may appear somewhat cumbersome at first, the benefits soon become very apparent. Also, the more available fields, the easier it is to edit and to SORT!

SELECTION SPECIFICATION - TO SELECT ONLY IF SOME CHARACTER EXISTS

The DEC manuals fail to disclose the selection specification which can be used to select a record only if a field has information. The wild card specifications presented by DEC are <?> and <*>. The <?> is used to replace a letter (i.e., it must be preceded or followed by some character other than a <?>). The <*> is used to define a field as containing ANY OR NO characters.

From time to time it is necessary to select a record ONLY IF A GIVEN FIELD HAS SOME INFORMATION. E.g.:

(1) if<field5> =<?><*>
then process record

(2) not if<field5> =
then process record

Of the examples given, each should result in the record being processed only if the information is present. Example 1 is believed to be more reliable.

DELETING UNUSED LINES FROM FINAL OUTPUT WHERE THERE IS NO DATA

This is a repeat of an article dealing with this same subject in the last Symposium Papers. It does include some updated information which may prove helpful to you.

DEFINING THE PROBLEM: EMPTY FIELDS ON LINES WHICH SHOULD NOT BE PRINTED. The problem which often is encountered is how to eliminate blank lines which are printed when there is a field which is empty, but which has been defined in the form. We will use an address block as an example.

```
<NAME>
<TITLE>
<COMPANY>
<APT/SUITE#>
<ADDR1>
<ADDR2>
<ADDR3>
<CI/ST/ZP>
<DROP>
```

In the example presented it is obvious that several of the fields might not be present in the final printout. The individual may have no title; s/he may not be associated with a company; there may only be a single address line. However, if the FORM is created in the manner indicated, which, in the example (and only by way of illustration) would be the same as the LIST, the final output would be printed with blank lines for each line on which there is missing data.

There is a solution. It takes a little planning, but once understood, it is simple to apply to every situation. (Just keep in mind, however, that this solution will cause each missing field to disappear and to bring the following line up one line feed! You must remember to allow for this, if the missing lines could affect other line-count features of your form.)

The first step is in the creation of a FORM. To accomplish the desired result for any set of circumstances it is necessary to create two FORMS. [NOTE: If you are positive that there are no spaces following any blank field, then the FIRST FORM is not required, i.e., you may proceed directly from your list as with the second form described below.] The first FORM should include only the variable information, and will, itself, become the LIST which then will be used to create the actual FORM or PRINTOUT. THERE CAN BE NO SPACES OR TABS ON ANY LINE WHICH MAY "DISAPPEAR", EITHER IN THE ORIGINAL LIST OR ON THE FORM. (Adjust the Left Ruler in lieu of a single tab, if indentation is desired.)

The FIRST FORM is created to determine which, if any, fields are not present and automatically to create a "wrap", as opposed to a HARD RETURN, for each such field. It also is used to create the second LIST. To accomplish this, it is necessary to create "soft" returns on each line which may not have information upon a field. This is done by using dummy rulers after each line which reasonably is expected to "wrap". Using the LIST above, and assuming that EVERY LINE may possibly have a missing field, we could create a form as follows [NOTE THE RULERS!]:

```
L----------------------------------------------R-
.  1  .  2  .  3  .  4  .  5
.....0.....0.....0.....0.....0..
L----------------------------------------------R
<<NAME><NAME>
LT---------------------------------------------R-
<<TITLE><TITLE>
L----------------------------------------------R-
<<COMPANY><COMPANY>
LT---------------------------------------------R-
<<APT/SUITE#><APT/SUITE#>
L----------------------------------------------R-
<<ADDR1><ADDR1>
LT---------------------------------------------R-
<<ADDR2><ADDR2>
L----------------------------------------------R-
<<ADDR3><ADDR3>
LT---------------------------------------------R-
<<CI/ST/ZP><CI/ST/ZP>
----L----------------------------------------R-
    <<DROP><DROP>
LT---------------------------------------------R-
<<>
```

Note that each of the rulers is identical, except for the dummy tab which follows every alternate ruler. The only purpose for the tab is to create a new ruler which can be imbedded. (If the rulers were identical, they would all disappear, and the method described could not be used.) Also note that the last line, DROP, has been indented by changing the left margin. The "indent" feature may be used on any line and is used to avoid the insertion of tabs or spaces which necessarily will defeat this utility. Also note the "<<>" identifier to create the new list! (Down arrows indicate hard returns which may be observed with GOLD VIEW.)

Using the blue EDIT keys, proceed to the beginning of each line AFTER A LINE WHICH MIGHT RESULT IN AN EMPTY FIELD. Use the Blue LINE key to travel from line to line. With the cursor on the left margin, strike the RUB CHAR OUT key ONCE. (This will delete the hard return and, upon a GOLD VIEW, will disclose a funny looking circle at the end of the sentence, instead of a down arrow.) Repeat for each line which might reasonably be expected to have an empty field. [If it becomes necessary to edit the last letter, back the cursor to the end of the line -- this will place it under the last letter -- and insert the new characters. The last letter will continue to travel and, if undesired, must be deleted.] Run the List Processing feature, creating a document. The document created by this feature will, itself, become the LIST for the second part of the program.

Upon completion, you will have created a form which, when operated with the List Processing Feature, will result in a new LIST which will have "wraps" in each empty field, between a ruler. YOU MUST BE CAREFUL TO AVOID TABS OR SPACES IN EMPTY FIELDS AND IN THE FORMS or this utility will not work properly.

The SECOND STEP is to create another FORM, which is identical to the first, except for the special field indentifiers. REMEMBER — if your original list is free of imbedded spaces following empty fields, this step may be your first step.

From the following illustration, note that that extra field identifiers have been removed. This will be the final list and will eliminate the spaces between lines which otherwise would have been created as a result of unwanted fields.

If you should find spaces between lines, the problem most likely will be that tabs or spaces were imbedded in either the FORMS or the original LISTS. Check them carefully.

The following form is such an illustration:

```
.  1  .  2  .  3  .  4  .  5
.....0.....0.....0.....0.....0..
L----------------------------------------------R-

<NAME>
LT---------------------------------------------R-
<TITLE>
L----------------------------------------------R-
COMPANY>
LT---------------------------------------------R-
<APT/SUITE#>
L----------------------------------------------R-
<ADDR1>
LT---------------------------------------------R-
<ADDR2>
L----------------------------------------------R-
<ADDR3>
LT---------------------------------------------R-
<CI/ST/ZP>
--------L----------------------------------R-
        <DROP>
LT---------------------------------------------R-
```

As with the first FORM, line feed to the beginning of each line AFTER the field which may not be present; and enter a RUB CHAR OUT to delete the hard return. (If you merely copied the document, be careful, as you may delete a character from the preceding line. To edit this problem, BACK UP to the preceding line (you will be on the last character). Re-type the last character (the one which is above the cursor) and any character which was deleted. Finish with a hard return. Delete the remaining character above the cursor, which should remove the hard return, also. (Check with GOLD VIEW.)

Now, USING THE NEW LIST CREATED BY THE LAST FORM AS YOUR LIST DOCUMENT, run the list processing again. This time, the new document (which also can be a direct PRINT) will cause all of the empty fields to "fold" upon themselves, so that all of the rulers with "soft" returns will collapse and final output will be without lines between information. While all rulers will appear on the screen, there will be no returns within them; the printer will skip to the next line of text without printing the "empty" lines.

If it appears that there is a space between rulers on which there was no data, check to see if there had been a space or tab on either of the FORMS or LISTS used for the procedure. Check your original LIST with GOLD VIEW. Each empty field's right arrow should be followed by an immediate down arrow (without a space).

Remember, you only have to create the two forms ONCE. They can be used for every processing run. (Actually, you need create the form only once, and then add the extra field identifiers to one of the forms. If you should get a line wrap, because of the extra space required by the new field identifiers, don't worry. The program automatically will adjust.)

PROGRAMMING NOTE: Although you can use the same selection specification for both forms, you also can use the simple specification of "process record" for the second run, as you already have specified the records to be used.

COMFORT NOTE: Although this may appear somewhat clumsy, it actually is rather easy and once you get the hang of it, you will find the procedure very useful!

USING LIST PROCESSING TO CREATE AN INDEX OR TABLE OF CONTENTS

NOTE: The following discussion provides a method for creating an Index or Table of Contents within a document. It is not the same as, nor an alternative to, the preceding discussion which uses the system index for a similar result.

Presently there is little ease with which to create an index or a table of contents with the existing WPS-8 or WPS-11 systems. While 11-based systems automatically can create an index and Table of Contents, and other dedicated word processing systems do the same, some ingenuity is required to acomplish this with DEC's systems (although we are assured that this, too, will change some day!).

For the time being, a fairly long document can become a LIST document using the following procedure.

First, copy the document over to another location (or on another diskette), as you are going to alter it (i.e., destroy it) considerably.

Second, decide on some easy shorthand for the categories you are going to use with your index or table of contents. For example, you might wish to use <H> for headers; <N> for names, etc. Choose a character to be used as a dummy field identifier, e.g. <X>.

Enter a terminator and the dummy field identifier in the PASTE buffer, as you will be using it quite a bit during this exercise. (To enter it in the paste buffer, type it and then cut it.)

E.g.: <><X>

Start the document with the dummy field (e.g., <X>) and proceed to the first data which is to

be used in the Table of Contents or Index. Let's suppose the first data is a header, which will use the <H> identifier. Enter a terminator <> and field identifier <H> immediately preceding the header and then enter the PASTE immediately after the header. Thus, the document would appear something like this:

```
|---------- t o p  o f  p a g e ----------|
|<X>                                       |
|                                          |
|  (miscellaneous data)                    |
|                                          |
|   <><H>TITLE OF DOCUMENT<><X>            |
|                                          |
|                                          |
|<><H>First Subheading<><X>                |
|                                          |
|  (miscellaneous data)                    |
|      <><X><N>(desired name)<><X>         |
|  (miscellaneous data)                    |
|                                          |
|<> [entered as last character in document]|
|-------- b o t t o m  o f  p a g e --------|
```

In the same manner, identify the different titles throughout the document, such as names, subtitles, books, etc., until you have identified each item which will be used in your index or Table of Contents.

CAUTION: As you proceed through the document, enter the PASTE in a random manner (i.e., insert the dummy field identifier <><X>) about every 2/3 screen, or more often. This is necessary as no field may contain more than 1500 characters, and to avoid an error message you will have to insert the dummy field every so often. It doesn't matter how often you use the dummy field, as it never will be referenced during list processing.

At the very end of the document, be sure to enter a terminator <> or an error message will occur (it won't affect your program, but no error is more comforting than some buzz error which might leave some doubt).

After proceeding through the entire document, you can create a very simple FORM and SELECTION SPECIFICATION. The FORM may consist of a single entry (e.g., <H>). The selection specification may be "process record". Operating the List Processing, then, will transfer each datum identified with the <H>, and will skip all of the rest. (If you have to format the output, it will be much easier to do so after running the list processing.)

Also, if you have the type of document which might require some form of sorting, such as alphabetical listings, you can perform some minimal alphabetical sorting by use of the wild cards in your selection specification. (This will require several runs through the list processing; e.g.: if <N>=A* then process record, will pick up every name starting with an upper case A, etc.) If there are only a few records, then use of the cut and paste feature will probably result in an easier, as well as faster, alphabetical processing.

Another feature, which will result in much faster operation if several field identifiers are being used, is to utilize the double LIST feature (i.e., create a new LIST with a single pass). To create a new LIST, set up your FORM (for the above example) as follows:

```
<<H><H>
<<N><N>
<<>
```

Processing the entire document will fill a new document with each field, in a random manner, and you then can run a second pass which will be more selective as to the order in which you want the items to appear. All of the dummy <X> field data will be omitted from the new LIST.

COMBINING TABULATED & CENTERED TEXT

Flush left, tabulated, or flush right text can print on the same line with text which has been centered (using GOLD CENTER), as follows:

Using an "F" ruler, type two lines; one line is used for the centered text and the other line is used for the tabulated or flush date. Either line may be entered first, provded that the SECOND line is input as a superscripted line. For example:

```
F-------------------------------------------------R-
                    TITLE [GOLD C]
FLUSH LEFT DATA                    FLUSH RIGHT DATA
qqqqq qqqq qqqq                    qqqqq qqqqq qqqq
L-------------------------------------------------R-
```

Return to your normal ruler.

Upon printing, the half spaced ruler, combined with the superscripting (half space raise) will print everything on the same line - e.g.:

FLUSH LEFT DATA TITLE FLUSH RIGHT DATA

MAINTAINING REQUIRED SPACES ON PRINTED OUTPUT

Often it is necessary to maintain required spaces between words, especially with dates and names. One way to accomplish this is available if you do not use BOLDING in your document. Since the printer will allow BOLD to indicate a two wheel print command, you can insert any character where your space otherwise would appear, being certain to bold that character. (E.g., press the select key, enter the character, press the white bold key.)

When printing your document, indicate TW (two wheel) printing. The bolded character will not print, but the space will be fixed. Oh, yes. Don't forget to stop the printer after you have printed your document. It will be waiting for the second pass.

APPLICATION NOTE: This same procedure may be used to print out a Table of Contents, Index, or other special purpose excerpts. You can't edit it, as such, but you can view it.

LINE NUMBERING USING WORD PROCESSING

Presently there is no easy way to number the lines on a document under WPS-8. Perhaps some day the powers to be will provide us with this feature, but for the time being it is necessary to use some planning in order to accomplish line numbering.

At the moment, one way to number lines, whether starting with 1 and proceeding to nnnn, or repeating the same number of lines per page, is to do it by brute force.

Create your document in the normal manner, but allow sufficient extra space on the left margin ruler for the numbers to be used plus at least two spaces. Thus, if you ordinarily would use the left margin for your left ruler and expect to use three digits for the numbers, set your left ruler, initially, five spaces to the right. (NOTE: There will be a slight variance in this procedure for inside paragraphs. This is discussed below.)

Upon completion of the document, AND AFTER FINAL EDITING, the line numbers can be added by re-setting the left margin on the ruler to its normal location AND INSERTING A TAB AT THE FORMER LEFT MARGIN LOCATION. While this would ordinarily cause the text to "re-wrap", it will make no difference. Proceed to the beginning of each line, using the BLUE LINE editor key.

Enter the line number and then TAB. Repeat this for each line to be numbered. Since the text already has been edited, the new line numbers will not affect your prior formatting, as you are using all of the extra space with the line numbers and tabs.

The use of the line numbers and tabs will not affect right justification, as each line number will follow a soft return. HOWEVER, SUBSEQUENT EDITING WILL BE VERY DIFFICULT. Therefore, try to avoid numbering the lines until the document is ready for final output.

INSIDE PARAGRAPHS

To use the line numbering feature on inside paragraphs, where the numbering is to remain on the left margin, use a W (wrap) in the ruler instead of the L for Left Margin. The first line of each inside paragraph will have to be double tabbed, but you will find it fairly easy to master after a few attempts. When you are ready to insert the line numbers, it will be necessary to remove the W from the ruler, and to replace it with a T (tab). When tabbing over from the number insertions, the text will remain formatted in the same location as with the W, and, as before, right justification will remain undisturbed.

If further editing may be expected, it may be easier to retain a copy of the document before line number inputting, especially where the editing may be extensive. The procedure indicated is not intended as a solution, but, rather, as a procedure which may make life somewhat easier for you.

INSERTION OF PORTIONS OF LONG DOCUMENTS, TOO LONG FOR "CUT AND PASTE", AND/OR WHERE ALL IMBEDDED MATERIAL IS DESIRED, AND/OR WHERE A "GO GET" ROUTINE IS NOT AVAILABLE BECAUSE THERE IS INSUFFICIENT ROOM REMAINING ON THE DISKETTE

It is not at all unusual to have the need to use a portion of a long document in a document presently being created. Quite often, also, the size of the required material exceeds the buffer space allowed with the "cut and paste" method (which often deletes a lot of the material you wanted); the remaining space on the diskette is insufficient to allow you to GO GET the old document, and then cut out the unwanted portions (even if all you want is in the first few pages) or you want to retain imbedded materials, such as rulers and page markers, and the cut and paste method won't retain them. Do not lose hope, there is a fairly simple remedy.

SOLUTION: Edit the old document to the portions desired. Enter a "boilerplate library" type of indicator at the beginning of the text to be copied, and a <> terminator. E.g.:

<<COPY1>>text material (may be as long as needed) <>

Use the same procedure for each section to be copied, but identify each portion with different names, e.g.: <<COPY1>>, <<COPY2>>, etc. (These identifiers can be removed, later, quite easily by using the blue <> key to advance through the document and rubbing out the identifiers.)

Note the drive and document number of the old document. Return to your new document and, with the Gold Menu (i.e., the editor menu) feature, change the boilerplate library to the drive and document number containing the old document.

Proceed to the portion of the new document which is to receive the old document's information, enter GOLD LIBRARY and the name (e.g., COPY1, COPY2, etc.). The information will be transferred, including all imbedded materials, such as rulers.

After using this method, be certain to reset the boilerplate library, in the editor menu, for its usual document location.

ABBREVIATION AND BOILERPLATE LIBRARIES

There is no end to which the system libraries may be utilized by the Word Processing operator. These features not only are among the most important individual assets of the entire system, they have the added benefit of providing some fun and relief from what otherwise might be a boring day.

Naturally, the needs of each user will be different. We believe that the following hints will be of interest to most users.

UPPER VS. LOWER CASE FOR FIELD IDENTIFIERS.

Again, as with List Processing, there is no requirement that you use lower case field identifiers for the libraries. In fact, upper case identifiers generally are much preferable, as reference to the library document may be made in upper or lower case and still retrieve the document, whereas if the library field identifier is in lower case, only a lower case identifier will retrieve it. This especially can be annoying if you are seeking an abbreviation library document (which does not echo the input on the screen) and you happen to have the caps lock activated.

LOCATION OF LIBRARIES

The Word Processing manuals and the self-paced teaching manuals for WPS-8 identify SYSTEM 2 and SYSTEM 3 as the location for the abbreviation and

boilerplate libraries. Indeed, all the software for Word Processing comes with SYSTEM 2 and SYSTEM 3 initiated as the respective libraries.

There is no magic in the assignment of locations for the libraries and your own particular needs should dictate where these libraries are located, and even whether you might wish to change libraries during different operations (a very helpful and powerful feature).

In a client or job oriented operation, where each client or job is assigned an individual data diskette (or RL01 allocation) it might be most helpful to always have the boilerplate library as the first document to be created on that data diskette (which always will be document #2, as #1 is reserved for the diskette's index). If this is done, data which is repetitious for each client or job easily may be recalled by using the same abbreviations or identifiers for each diskette. For example, in our own operations we would identify the name and address block of our client with a field identifier of <<CLIENT>>. Since this information resides only on the diskette in use, every time the library identifier of CLIENT is used, the name and address of that particular client is displayed in the document.

In this manner, the SYSTEM diskette's space is reserved for other needs, and many other libraries.

ALTERNATING LIBRARIES

There is no particular requirement that the library document always be in the same location. On the other hand, it often is helpful to be able to have several documents available on a given diskette which can be utilized as a library document for a particular purpose. This especially is helpful in creating new documents where there is going to be repetitious use of some phrases. A new abbreviation library can be created, for these phrases only, and the phrases called with short entries (and no delays). When completed, the library contents can be deleted (or retained, if desired) and the library document changed to the standard document.

The use of such a "temporary" library especially is appreciated when one no longer has to search through the current document for specific phrases to be "cut and pasted" at a specific location.

If a library document becomes too lengthy, then it takes a considerable period of time for the computer to find the phrases you need. To avoid this problem, you often can break your library documents into categories, and, knowing the category desired, assign that document as the library (abbreviation or boiler plate) document for the current assignment.

USE OF THE HELP COMMAND FOR LIBRARY CONTENTS

As use of library documents increases it becomes increasingly difficult to remember field identifier assignments, and hard copy reminders become antiquated, misplaced, or unhandy. There is, however, an on-line solution, and that is a HELP COMMAND.

When creating a library document, the first field identifier should be <<HE>> for the abbreviation library and <<HELP>> for the boilerplate library. (Entering "help" will call the field in both cases, although the extra letters ("lp") will appear on the screen after an abbreviation library call.)

Prepare a Table of Contents which identifies each field identifier and its meaning, which can be called by the HELP command. As each new abbreviation is added to the library, the HELP section also is updated with the new command information. E.g.:

<<HELP>>CLIENT, OPPONENT, CAPTION, ENVELOPES<<

To seek and examine the HELP information, which only can be accomplished while editing a document, the operator simply (1) enters the SELect key; (2) enters GOLD ABBREVIATION or GOLD LIBRARY and the word HELP (although only HE is required for an abbreviation); and the HELP information is displayed upon the screen. [Reference to "sub-help" libraries may be followed with another GOLD LIBRARY command.] After examining the displayed information, the operator (3) strikes the CUT key and all the displayed information is removed from the screen to the position where the SELect was inserted and the library may be accessed for the desired field.

By no means is the information provided here exhaustive of the potential for the HELP library. One may use HELP as a key to provide the operator with special instructions with respect to procedures to be followed with specific routines or documents. In fact, one can have a separate library entitled HELP. (To access the library, the library document is changed to [diskette/area].HELP, from the Editor Menu, which automatically will change the library document.)

SETTING UP THE LIBRARY DOCUMENTS TO DELETE THE HARD RETURN

There are two methods available to avoid hard returns following a Library Document call. (I.e., where special formatting is required the formatting must follow the library field identifier in order to be imbedded.)

The first method, of course, is to have the library information begin immediately after the field identifier (<<field>>Data xxx).

The second utilizes the soft return described in preceding sections. The following is an example:

```
L-------------------------------------------R--
<<DOCUMENT>>
LT------------------------------------------R--
                    TITLE

                    (data)
<<
```

Without modification, if DOCUMENT is referenced by the library an extra return will result, as creation of the document necessarily required a hard return after the field identifier.

However, with the insertion of a new ruler (a dummy ruler is indicated, but it should represent a required format) the hard return can be changed to a soft return (by moving to the beginning of the line immediately after the field identifier and striking the RUB CHAR OUT key) and, when referenced, the soft return will be ignored. The imbedded ruler also will appear. The same result can be attained by using a PRINT CONTROL (but be sure to include both the START and the END PRINT CONTROLS). Go to the beginning of the line on which the BEGIN PRINT CONTROL appears and enter your RUB CHAR OUT key. The hard return will dissolve into a soft return.

USE OF LIBRARY DOCUMENTS FOR EXTRA RULER STORAGE

Quite often the ten ruler storage availability of the Word Processing System is inadequate, either because more rulers are required or because it is difficult to remember which is which. There is an alternative.

Using the same technique for removing the hard return described in the foregoing sections, rulers can be saved in a library and can be called by document type. This especially can be helpful for unusual documents, but also is helpful for general documents. The following are examples of two rulers. Expansion of the concept is quite unlimited and, obviously, up to the individual user.

```
<<LETTER>>
L---P----T-----------------H------J----------
<<

<<SCHEDULE>>
L---T----T----<------.---------.---------.--->R
<<
```

Naturally, a schedule of all of the rulers can be part of the HELP library.

SPREADING A TITLE OR HEADING

One additional note. The soft return also can be used to spread a word. or series of words, accross an entire page. Provided that the RIGHT MARGIN is set with a J (justification), everything on the soft return line will be spread across the page if the next line commences with any type of an imbedded command (e.g., ruler, print control, page marker, etc.). Example:

```
L---P-----T-----T----------------H-----J--------
(Miscellaneous text to the next line)

                    T I T L E
L---T-----T-----T----------------H-----J--------
```

By creating a soft return (i.e., using the RUB CHAR OUT at the beginning of the first line after "TITLE" in the above example, the word "TITLE" will be spread accross the page as follows:

T I T L E

In order to create the "soft" return, it only was necessary to modify some portion of the ruler, imbed it and then delete the hard return with the RUB CHAR OUT from the beginning of the line.

SOME ADVANCED FEATURES FOR INDEXES (INDEXI?)

In a preceding section of this paper, we have discussed the manner in which an index or a Table of Contents can be created, using the List Processing features of the Word Processing Systems. Here we will discuss, briefly, two enhanced features — including page numbers and sorting.

To include page numbers it is, of course, necessary to have a "finished" copy of the document, as you will have to insert the page numbers after each word to be referenced.

Using a "finished" copy, we mark each word which is to become part of the index. Using the techniques described above, we enter the field identifier before the word, or phrase, and follow it with a space and the page number. Then, the terminating left/right arrow and dummy field identifier follow the number. For example, we are indexing on the phrase "List Processing":

[Text ... <><I>List Processing 22<><X> ...]

In a long document, we do not attempt to identify each page where the word or phrase appears, but merely repeat the process.

Depending upon how involved the index may be, we will either run the List Processor on the <I> field and pick up every instance using a "sub list" which will be SORTed, or we will use a conditional run, based upon the first letter of the field creating a semi alphabetical listing.

[The "sub-list" is quite simple. It is identified as <<I><I>.]

If we create a long index, we then SORT the "sub-list" index using the SORT package which is available through your DEC WPS representative.

What's this? You say your rep doesn't know what you are talking about? If you have any difficulty, send this author two blank diskettes, a returnable mailer AND RETURN POSTAGE, and we will send you a copy of the SORT package and its instructions. Two diskettes are required as the SORT is in DIBOL and operates on COS-310, whereas the instructions are on a WPS diskette. One caveat: you must be licensed for DIBOL (although most word processing users are).

Upon completion of the SORT, it is a fairly simple matter to "cut" the repetitions and allow their page numbers to flow onto the first use of the term. (Using a RIGHT ARROW ">" as the right margin — before the R or J — will allow these numbers to flow backwards so you can have several numbers, with commas, on the same line.)

PERMANENT CUTTING OF LONG SECTIONS OF A DOCUMENT

As you may realize, it is not possible to "cut" more than about 2-1/2 pages of a document at one time. If you want to delete several pages, and do not have to save them for any other purpose, then this can be accomplished with a single operation without concern over the actual length of the material being deleted.

The procedure requires that you proceed to the starting point of the "cut" (which can be the end or the beginning of the "cut") and press the white SEL key. Then, immediately press the red CUT key. This will remove all data from the paste buffer.

Press the white SEL key again and proceed to the point where you wish to end the cut. Press the GOLD REPLC keys (GOLD and '). This will replace all of

the data between the SELect position and your current place with a single null. The cut data is not replaceable so do not use this as a "cut and paste" routine. [For long "CUT AND PASTE" routines, refer to the section above which discusses changing your document to a library document.]

TRANSPORTING RULER AND PRINT SETTINGS TO NEW SYSTEMS

Often it becomes necessary to transport your ruler and print settings from one system diskette to another. There is a fairly easy way to accomplish this.

For the ruler settings, you merely create a single document using your old system diskette and then begin to enter ruler settings separated by some meaningful code so you can identify the settings when used on the new system. E.g.:

```
0
L---P---T----->----- ... -R
1
D---P---T-----T----- ... -R
```

etc.

Placing this document diskette under control of the new system diskette (and the document may, of course, reside on the new system diskette) you merely advance below each ruler, enter GOLD RULER and then SHIFT [number] to preserve the ruler. Continue with each ruler until all 10 have been transferred over (or as many as are needed).

NOTE: The same procedure can be used in your LIBRARY to save more than ten rulers, or to call rulers by document name. Just enter the RULER NAME between the arrows (e.g. <<LETTER>>) and follow the identifier with a return. Place the desired ruler under the identifier, then enter a single RUB CHAR OUT to remove the hard return and enter your end of field marker (<<). Now, when you call the RULER from the LIBRARY, it will appear where you want it without extra returns.

For PRINTING COMMANDS, create up to ten separate documents and merely file them away. Go into the PRINT MENU for each document (either as you create them and afterward) and call up your PRINT COMMANDS from your old SYSTEM diskette. Save the commands on the document by using the GOLD MENU. Placing the diskette with your "new" SYSTEM diskette, you merely begin to print each of the documents. When in the PRINT MENU enter SS nn to save your PRINT COMMANDS as reflected on the particular document.

The procedure also is available for special printing requirements on documents where there are insufficient printer commands (i.e. — 10) available. Just identify a document with the name of the particular type of printer commands you want to save, and you always can assign that to any of the numbers you wish for special purposes. (We find that reserving control number 9 for this purpose works out just fine.)

*** CREATING A TELEPHONE LOG**
(CAN BE USED FOR CREATING A MAIL LOGGING SYSTEM)

INTRODUCTION: The following application is for creating a telephone logging system on the DIGITAL word processor. It can be used by a secretary to record all messages coming into an office. It is assumed that the operator is very familiar with the word processing software and its operation.

By doing a "GLOBAL SEARCH & REPLACE" on the TELEPHONE LOG Application Notes, you would create "MAIL LOG" Application Notes. This memo serves as a brief explanation of both procedures.

Primarily, both applications call for an area on a system or diskette to be dedicated to the storage of telephone messages and a separate area for mail logging. It is necessary for the user to be familiar with list processing, creating forms, and the use of document 1 (the INDEX). The codes in the following application are suggested and can be "added to" or "deleted from" depending on the company or customer wishing to utilize this as a procedure. The method for the telephone log and mail log is identical in implementation. The difference is in the "list" code used. One being:

<Code>Telephone Message

and the other,

<Code>Mail Message

The reports generated would have title changes and the other information would be the same. As with the telephone log, the asterisk indicates "new" information. The date may be incorporated as part of the title as well as for ease of chronological input.

Some coding during creating to designate mail type could be:

- c *l/John Jones - indicating that the mail is a (l)etter.
- c *m/John Jones - indicating that the mail is a (m)emo.
- c *p/John Jones - indicating that the mail is a (p)ackage.
- c *i/John Jones - indicating that the mail is (i)nformation.
- c *cc/John Jones - indicating that this is a (c)arbon (c)opy from.
- c *pr/John Jones - indicating that the mail is (p)er your (r)equest.
- c *b/DEC WP - indicating that the mail is a (b)rochure.
- c *a/Sears - indicating that the mail is an (a)dvertisement.

The other codes listed in the telephone log procedure also can be used designating "A" for Action, "RR" reply requested, or whatever is required. Using the mail log requires that the secretary summarize all incoming mail for quick reference by the reader. Both applications can be implemented easily and can be designed to meet the user's needs.

APPLICATION FOR TELEPHONE LOG

THIS IS A PROCESS WHICH ALLOWS A SECRETARY:

- to take phone messages,
- enter messages into the system in a telephone log area,

and

- to accomplish the above in as few steps as necessary by using User Define Keys and List Processing.

Each message is entered by creating a separate document and using a user define key (UDK) to call up an empty list processing record. These field identifiers will enable the user to organize the information into specific categories.

The reader (the person the call is directed to) then is able to review the list of messages by looking at the designated area's index and then proceed to take action. By recording each message in a separate document, the reader can be acting on one message while the secretary can be entering new messages (assuming that the reader has his/her own terminal). Also, a message can be incorporated into another document or sent to someone else via communications by using the Gold:Get Document or the DX feature. When creating the document which will hold a message, a code is incorporated into the title of the document which will communicate between reader and secretary exactly what has to be looked at, by whom, and what has been handled and no longer needs attention.

By using field identifiers, a secretary can obtain a print out of all calls which need to be returned or acted upon through the list processing method. This is especially helpful if the reader is away on a business trip and the secretary would like to give him a hard copy summary of calls to scan upon the his/her return or to brief the reader in his/her absence.

PROCEDURE FOR SETTING UP THE TELEPHONE LOG

1. ASSIGN AN AREA.

   Determine which area on the system will be devoted to telephone messages. For the purpose of this application, the area chosen is four (4) on a 200 System.

2. DETERMINE FIELD IDENTIFIERS.

   Design field identifiers that will be most helpful to organizing the telephone information. An example of the identifiers to use are as follows:

   ```
   <Date>10/17/80 Fri 12:13  - Use GOLD Date/Time
   <To>
   <From>
   <Company>
   <Subject>
   <Tele No>
   <Request>
   <Code>Telephone Message
   <Action>
   <Follow Up>
   <>
   ```

3. RETRIEVE FIELD IDENTIFIERS VIA UDK.

   Access the FIELD IDENTIFIERS via a user define key (UDK) either by typing the entire identifiers or by placing them in an abbreviation library and typing the command assigned in the abbreviation library into your definition in the UDK. The following are examples of both methods of setting a user define key to accomplish this.

   Type dk 1 command at main menu. Any number may be chosen from 0 to 9 to store the definition, i.e. dk 2, dk 3, etc. For purposes of this example, DK 1 is used.

   TYPE DK 1 Return

   ENTER:

   <Date> Return <To> Return <From> Return <Company> Return <Subject> Return <Tele No> Return <Request> Return <Code> Telephone Message Return <Action> Return <Follow Up> Return <> Backup Para Advance <> Gold:\ <> Gold Halt

   RESULT:

   In typing Return, Backup, Para, etc. make sure you press the key labeled keys Return, Backup, Para, etc. and do not spell out the actual word. Notice, that by typing Backup Para Advance <> Gold:\ <> the system will automatically print the identifiers, backup to the beginning of the paragraph, advance to the enter sign ">" in <Date>, insert Gold Date/Time, and advance to the next enter sign ">". The cursor will position itself for entry and where to begin typing the information at <To>.

   or define DK 1 as follows:

   Gold:=tl Backup Para Advance <> Gold:\ <> Gold Halt

   (tl = telephone log identifiers from abbreviation library.)

Accessing your abbreviation library gives more flexibility in recalling telephone log information. If the fields change or more than one person is using this procedure, they can have their own list stored.

** Please note that in both definitions that typed text (words or characters) will appear to be one space apart. They are stored this way within the UDK but are not entered this way in the document.

4. ESTABLISH A CODE to be incorporated in the title of the document which will hold a telephone message for identifying what calls should be looked at, what calls have been handled and require action by the (R)eader or (S)ecretary, and what calls have been completed.

****

Coding should relate the needs of a particular office and should be as simple as possible. Using familiar terminology would be best. If many people are using the telephone log, it will be necessary to specify people, departments, etc. Coding can be as simple as putting the person's initials in the code to designate that their action is required if the call is to be referred to someone else for action or some scheme, such as the following example, may be adopted:

*/ - denotes new call/information.
*A/ - denotes a new call requiring (A)ction
R/ - denotes call was (R)ead. This is entered by (R)eader.
SR/ - denotes call was (R)ead and requires action by (S)ecretary.
RN/ - denotes call was (R)ead and (N)o action required.
RA/ - denotes call was (R)ead and (A)ction is required by reader.
RD/ - denotes call was (R)ead and can be (D)eleted.
SRS/ - denotes call was acted on by (S)ecretary after being (R)ead.

UTILIZING THE TELEPHONE LOG TO RECORD MESSAGES.

The procedure is as follows:

1. When a telephone message is received, create a document which will record the message in the designated area for the telephone log.

Example: c 4.*/John Smith 617-555-1212

The above example creates a document entitled John Smith in area 4 (designated telephone log area). The asterisk (*) in the title is part of the code which tells the reader when looking at the area 4 index that a call has been received from John Smith. The "*" designates that it is a new call and should be read in the telephone file.

2. When the new telephone message file is created, type Gold:1 to place the field identifiers in the document. Then type in the information corresponding to each identifier and file the document. The message is now recorded and can be referred to. As the list of message documents grows with each new call, the asterisk in the title quickly identifies the calls that require action.

3. When a call is acted upon, it is noted by the reader in the <Action> line by logging in the date using the Gold: Date/Time next to the <Action>. The <Follow Up> identifier provides an area for the reader to enter a summary of the call and action taken. The reader then edits the index (Document 1) and codes the call by adding either with an "RN" denoting that the message was read and no further action necessary, "RS" denoting instructions for further action by the secretary, or "RA" denoting he has reviewed the call but has not as yet taken action. When further action required by the secretary (all calls coded RS) is completed, she then edits the index and codes the call by adding another "S" or her initials. (SRS). If a summary of the action taken by the secreatry should again be brought to the attention of the reader it will noted with another "*" designating that it should be re-read.

An example of a coded index (Document 1 of an area) is as follows:

<n>*/John Smith 617-555-1212 <#>7<>
<n>*A/Karen Black 883-1111 <#>6<>
<n>*SRS/Bill Jones 888-5000 <#>5<>
<n>RN/Wendy Turner 4-5555 <#>4<>
<n>RA/Jane Doe 223-5555 <#>3<>
<n>RD/Mary Brown 212-555-1212 <#>2<>
<n>Area 4 Index <#>1<>

4. GENERATING TELEPHONE MESSAGE REPORTS

When someone is out of the office for a period of time and it is necessary to print a list of messages received, which must be acted upon, etc., it can be done by by using the list processing feature.

The list used is the actual index (Document 1) of the designated area for the telephone log. In this example, document 4.1.

Then a specification is created to capture and print only those calls on the list which require action, i.e. all calls coded with an A*. For example:

If <n>=*A/<*>
then process record

Using the <*> wild card feature of list processing, all messages regardless of content but requiring (A)action will be processed.

NEXT create a form which will print out the list required. The following is an example of a form:

```
                TELEPHONE LOG

    FOR SALES REPRESENTATIVE:  JENNIFER JOHNSON


  DATE  NAME & TELE NO.                          File #
-------------------------------------------------------
<!S>
  <Date>      <n>                 4.<#>
<!E>
```

The form can contain as many or as few of the fields required. It could be just a listing as shown above, or give all information entered by creating a form calling for all fields to be filled in.

An example of the result:

```
                TELEPHONE LOG

  DATE        NAME & TELE NUMBER                File #
-------------------------------------------------------
  10/14/80 Tue   */JohnSmith 617-555-1212        4.7
  10/14/80 Tue   A*/Karen Black 883-1111         4.6
  10/14/80 Tue   *SRS/Bill Jones 888-5000        4.5
  10/14/80 Tue   RN/Wendy Turner 4-5555          4.4
  10/14/80 Tue   RA/Jane Doe 223-5555            4.3
  10/14/80 Tue   RD/Mary Brown 212-555-1212      4.2
```

This process for handling messages can be very effective. Also, as a means of recording all messages and documenting all action taken on each is very efficient. You will discover that there are many variations that can be employed to make this procedure even more efficient for your particular needs, such as entering more information into the title of your document, using different identifiers, setting up a form which can arrange the information obtained in your list to suit your needs, etc.

This process proves to be time saving when someone is out of the office but has access to a terminal which is remotely connected to the main system in an office. A caller, salesperson, etc., can review all calls, communicate with the secretary regarding what action has or should be taken without having to talk with the secretary directly. This is particularly helpful if the manager is visiting a different time zone or is reviewing calls after office hours.

On occasion, it also can be used as a reminder to do something or call someone (tickler file) by creating a document entitled *Notes and then typing several notes serving as a constant reminder of issues which should be handle along with the calls. Or, by creating a document A=Jim McDonald 4-5556 to remind the reader that they wanted to call this person.

There are many ways in which to use this log and many variations you can make with a little imagination and practice. Try it, you'll like it!!!!

"CREATING A CALENDAR" APPLICATION

This feature was developed as a time management tool to efficiently handle the scheduling of meetings and appointments, whereby more detailed information on the meetings can be presented. A list of things that must be done each week or month can also be incorporated into this calendar.

The setting up of a calendar on a word processing system is a fairly simple procedure. The steps are as follows:

1. Choose an area (or diskette) which will contain the calendar for a full year.

2. Each month should be created in a separate document within the same area or diskette.

3. Choose the format you wish to use and create the first document entitled January or the current month. Type the calendar in the format you have chosen and file the document.

At the top of the calendar is a date and time notation. Each time a new item is entered in the calendar, the date and time should be entered at the top of the calendar to let the manager know when the last notation was made.

After the area provided for Friday, you will notice angle brackets "<>". This will enable you to advance rapidly to this symbol and give you an entire work week at a glance by pressing the "<>" key.

The format below should continue on to include the 2nd, 3rd, and 4th week of the month with a "new page" command between each week. All you need do once the calendar is formed is type in the meetings or appointments in the appropriate places.

Note that at the bottom of each week, an area has been provided for notes. This area allows more detail when required on any meeting scheduled in that week. The angle brackets at the end of the note allows you to advance rapidly to the end of the last note and continue on easily with any additional notes as you edit the calendar during the month. EACH WEEK SHOULD BE ON A SEPARATE PAGE.

After filing the document entitled "January", you are ready to create "February". Simply create a document entitled "February" and then use the GOLD:GET DOCUMENT feature to call up the calendar for January in the February document. Then all you need do is edit this document by changing the January dates to the appropriate February dates. Continue this procedure for each of the following months through December.

Another helpful idea which can be incorporated into this calendar is to create a "do list" which can be referred to readily. Simply make a list of all the things that must be done for the month at the top of the calendar with a new page command after the list and before the calendar. We have chosen to code each item in the list to designate what has to be done, what is presently being worked on, and what has been completed.

An example of the "do" list which has been used in creating the following calendars is:

"Do" List

<> Call Sally Smith (617) 223-5555.
<x> Find out if Staff Meeting is Wednesday.
<x> Meet with John Smith on presentation.
<-> Meet with Bill Brown subj: product application.
<> Get Print Requistition for WP Manuals.
<> Process paperwork for expenses.

When each item in the list has been entered, it is preceded with angle brackets <> as a symbol designating an item to be performed; when the item is in the process of being completed, we add a hyphen between the angle brackets <->; and when it has been completed, we add an x between the angle brackets <x>. You can also create a list on a week by week basis by placing the list at the top of each week separated by page markers.

The following is an example of a basic calendar on the word processor. Other examples to follow show how the "do" list is incorporated. As you examine these calendars, you might consider further automation by use of your list processing functions.

```
1st Week  | 1/30/80 Wed 9:11
Jan       |8   9   10  11  12  13  14  15  16  17  18  19  20
----------+-----------------------------------------------------
Mon       |
12/31     |
----------+-----------------------------------------------------
Tues      |
1/1       |
----------+-----------------------------------------------------
Wed       |
1/2       |
----------+-----------------------------------------------------
Thur      |
1/3       |
----------+-----------------------------------------------------
Fri       |
1/4       |
          |
<>        |
----------+-----------------------------------------------------
Sat       |
1/5       |
----------+-----------------------------------------------------
Sun       |
1/6       |
----------+-----------------------------------------------------
Notes:<>
```

An example of a calendar which has been filled in and incorporates the "do" list is below.

MONTH: AUGUST

```
1st Week  |
Aug       |8   9   10  11  12  13  14  15  16  17  18  19  20
----------+-----------------------------------------------------
Mon       |New York trip ..................................................
8/4       |
----------+-----------------------------------------------------
Tues      |    John Brown                  [.................]
8/5       |    9:00 - 11:00                Strategy  Mtg.
          |                                Dartmouth Room
----------+-----------------------------------------------------
Wed       |              Call to John Smith
8/6       |              231-2294 re:ABC
          |              (See Notes)
----------+-----------------------------------------------------
Thur      |
8/7       |
----------+-----------------------------------------------------
Fri       |              Peter Franklin Mtg.
8/8       |              Your Office
          |              11:30 - lunch
          |              (See Notes)
<>        |
----------+-----------------------------------------------------
Sat       |
8/9       |
----------+-----------------------------------------------------
Sun       |
8/10      |
----------+-----------------------------------------------------
Notes:
```

8/4 - Delta FL 111 departs Boston 9:00 and arrives New York at 9:45. Take taxi to New York office to meet with sales rep. Delta FL 222 departs New York at 4:10 and arrives Boston 4:50.

8/6 - Phone Call to John Smith concerns memo from him dated 6/20 (located in 6/20 mail log).

8/8 - Peter Franklin, wants to discuss the XYZ program.<>

8/26/80 Tue 9:24

MONTH: AUGUST

```
2nd Week  |
Aug       |08  09  10  11  12  13  14  15  16  17  18  19  20
----------+-----------------------------------------------------
Mon       |        Staff
8/11      |        10:00
----------+-----------------------------------------------------
Tues      |Boston all day
8/12      |(See Notes)
----------+-----------------------------------------------------
Wed       |                    Mike Jennings   Jack Flint
8/13      |                    12:00 - 1:00    2:30 - 3:30
----------+-----------------------------------------------------
Thur      |XYZ.....................................]   Ted Lansing
8/14      |extended to 3:30                           4:30 MK
----------+-----------------------------------------------------
Fri       |Vacation Day
8/15      |
          |
<>        |
----------+-----------------------------------------------------
Sat       |
8/16      |
----------+-----------------------------------------------------
Sun       |
8/17      |
----------+-----------------------------------------------------
Notes:
```

8/12 - Trip to Boston to consult with Bob Griffith, 606 State Street<>

GENERAL PURPOSE HINTS

The following hints have been developed over a long period of time and have been helpful to many of us. We hope that they will be of some assistance to you.

PRINTING A DOUBLE UNDERSCORE — Two Methods

Quite often there is a need to print a double underscore -- especially where columns of data are involved. Two fairly successful methods can be used.

Method One. If the double underscore is to be used with tabulated work, and the last item in the column is to be double underscored, use the select key at the beginning of the item and go to the end of the item. Enter the white "underline" key, which will cause a single underscore. (If you want the underscore to extend in front of the item, insert spaces in front of the item, and underscore those spaces.) Then, tab over to the same point on the following line and insert your underscore, using the underline key, so that it covers the same area immediately above. Using the select key at the beginning of the underscore, go to the end of the underscore and enter the GOLD SUPERSCRIPT. This will cause the second underscore to print one-half space up and will give a very presentable double double underscore. (An alternative to the superscript would be to change the left margin in the ruler to the "f" for half spacing. Just be sure to change it back again for the remaining text.)

Method Two. Several print wheels have a double underscore character as part of their character set. Currently, the following Diablo wheels have this character:

Courier Legal 10A
Pica Legal 10A
Prestige Elite Legal

1. When you get to the beginning of a line of text which is to be double underscored, put in a ruler of F for half line spacing. Tab over and type each of the numbers to be underscored. Press RETURN.

2. Tab to the first column to be double underscored. Press the vertical character key (|) [upper

case back-slash (date/time) key] once for each character to be underlined.

3. When the last long vertical character on that line has been typed, back up the line and press SELECT LINE GOLD SUPERSCRIPT. This will move your double underscore right underneath the numbers. Press RETURN twice and change your ruler back.

4. When printing the document with the Courier Legal 10A or the Pica Legal 10A wheels, you must use the replacement character (R1 ¦) from the Print Menu. This is not necessary for the Prestige Elite Legal wheel, as the "¦" character is not on that wheel but is, in fact, the double underscore.

TRIPLE SPACING BETWEEN DOUBLE SPACED PARAGRAPHS

It is a real pain to triple space between paragraphs which are created with a double spaced ruler. Here is an easy to use USER DEFINED KEY to handle the problem. Set your UDK for: GOLD PARA, GOLD RULER, L, RETURN, RETURN, GOLD RULER, D, RETURN. Upon reaching the end of your paragraph simply enter the UDK instead of your GOLD PARA for the end of the paragraph. [If you don't use GOLD PARA, then replace that instruction with a RETURN. If you use indented paragraphs, then substitute the last RETURN with a GOLD PARA.]

CHANGING THE DEFAULT RULER OR PRINT MENU SETTING

Surprisingly, many users are unaware of the fact that the default RULER and PRINT MENU settings are the same as the zero settings. If you wish to alter your default settings (i.e., the settings which are present when the document is created initially) all you have to do is change the settings for GOLD RULER 0 or PRINT MENU 0 [SS 0] to the desired defaults.

RECOVERING AN ERASED USER DEFINED KEY

Ever go into a User Defined Key to examine it and then strike a character thereby deleting the entire key? If you haven't, then you probably don't use these keys! There is a way to recover the "erased" information. But, you have to be careful (and this is not intended for 200 systems).

If you are in a document and have gone into the UDK from the Editor Menu, then you are going to have to decide whether the UDK is more important than any of the information which you may have edited. This technique is going to lose any NEW information which may have been input. (E.g., if you merely were editing an existing document, but had made no changes, then this will not affect your document. If you had made some changes, or if the document had never been filed, don't take a chance -- re-type your UDK.) If you entered the UDK from the Main Menu, then there is no risk at all.

Upon realizing that you have lost your UDK information, DO NOT PRESS THE GOLD HALT KEY. Simply turn off your computer and re-start it in the usual manner. The UDK will be saved.

STORING MORE THAN 10 UDKs ON NON-200 SYSTEMS

Generally, UDK storage is limited to 10 UDKs per SYSTEM diskette. However, it is quite simple to store as many UDKs as one wishes. Store your UDKs on different SYSTEM diskettes, as you generally would. You still are limited to 10 per SYSTEM diskette. When you want to use the UDKs stored on a different diskette, enter the F key from the Main Menu and press RETURN. Remove the SYSTEM diskette and replace it with the SYSTEM diskette which has the UDKs which you want to use. Press RETURN twice and the new UDKs now are available to you. (You even can replace the other SYSTEM diskette at this time, in case you have information on it which has to be used. Just make sure that your default libraries have not been changed.

AUTOMATIC DELETION OF SEVERAL DOCUMENTS AT ONE TIME

Often it is desireable to purge a diskette of several documents at one time and you don't necessarily want to sit there all day while you instruct the machine to do the deleting. This can be done with a simple USER DEFINED KEY and editing of the document INDEX. **ALWAYS USE CARE WHEN EDITING THE DOCUMENT INDEX. IF NECESSARY, USE A GOLD GET OF DOCUMENT #1 IN A SPARE LOCATION, SO THAT IF YOU LOSE YOUR INDEX YOU CAN RE-CREATE IT IMMEDIATELY.**

Edit document #1 (the diskette's index) and mark the documents to be deleted by inserting, immediately after the "<n>" for each such document an unique character followed by a space. For example:

<n>XX document name <#>8<>

The space between your characters and the document name is important.

After editing the index, file it. Then, using a USER DEFINED KEY (#1 here) which has been defined:

D Space XX Space Return y Return GOLD:2

and a USER DEFINED KEY (2 in this example) as:

GOLD:1 GOLD:1 GOLD:1 GOLD:1 GOLD:1 ... GOLD:1

you can proceed to delete, automatically, each of the documents which you have marked with your special characters by entering GOLD:1. [NOTE: The loop which you created is limited to 10 documents at a time.]

MOVING COLUMNS

Here's how to move columns quickly and easily. (Of course you can use the same technique for deleting columns and other uses, also.)

Assuming a columnur text of four columns, to move column 3 to column 2's location and column 2 to column 3's location, store a user defined key:

Advance Tab Advance Tab Sel Tab Cut Backup Tab Paste

Position your cursor at the beginning of the line where the columns are to be transposed. Use your UDK and the columns will be transposed. Of course you would adjust the UDK to reflect a different number of tab positions.

SWAPPING WORDS

Ever wish that you had a SWAP WORD key just as the SWAP character key? Well, why not create your own? The UDK to swap a pair of words is quite simple:

Delwrd Word G-Del

Position the cursor just in front of the offending word, enter your UDK and the words are swapped. This is a real handy tool, especially for those split infinitives!

A UDK FOR PRINTING A DOCUMENT WHILE EDITING ANOTHER

If you use SE YES (stop before printing every page) then you know that you have to enter an "R" before the printer will start printing the next page. If you are editing a document, this can become a pain, but there is a simple UDK to take care of the matter. Create your UDK:

GOLD:M R Return

Entering this UDK while you are editing a document will start the printer with a minimum of interruption to your editing. [Sorry -- this won't work for 78s as they can't use UDKs and the printer at the same time.]

SUMMARY

DEC's Word Processing Systems (and even those which utilize DEC equipment) clearly are among the most powerful available on the market today. The potential -- indeed the need -- for improvements is all too obvious, if DEC intends to remain a serious contender for the general business Word Processing Market.

In the meantime, there are numerous routines which are available in the existing system which can make it work better and faster for you, and that is what automated word processing is supposed to be all about.

The examples provided here are but a few of the many work saving features which are available. It appears that these examples never previously have been documented, at least publicly, by DEC, which really is a shame. Now, several of the hints provided here have been made available through the courtesy of DEC's Word Processing people who participated so helpfully in the presentation of the Hints and Kinks session.

We do hope that the information provided within this article will be of some assistance to the Word Processer user and that this Article may become part of your Word Processing Manuals.

Perhaps by the time you come to Los Angeles we will have a good deal more for you. In the meantime, please feel free to write to us regarding any particular matters you would like to see discussed at future Symposia.

DECUS

DIGITAL EQUIPMENT COMPUTER USERS SOCIETY
ONE IRON WAY, MR2-3/E55
MARLBORO, MASSACHUSETTS 01752

BULK RATE
U.S. POSTAGE
PAID
PERMIT NO. 129
NORTHBORO, MA
01532

INSTALLATION -

[redacted]

MOVING OR REPLACING A DELEGATE?

Please notify us immediately to guarantee continuing receipt of DECUS literature. Allow up to six weeks for change to take effect.

( ) Change of Address
( ) Delegate Replacement

DECUS Membership No.: _____

Name: _____

Company: _____

Address: _____

_____

State/Country: _____

Zip/Postal Code: _____

Mail to: DECUS - ATT: Membership
One Iron Way, MR2-3
Marlboro, Massachusetts 01752 USA

Affix mailing label here. If label is not available, print old address here. Include name of installation, company, university, etc.