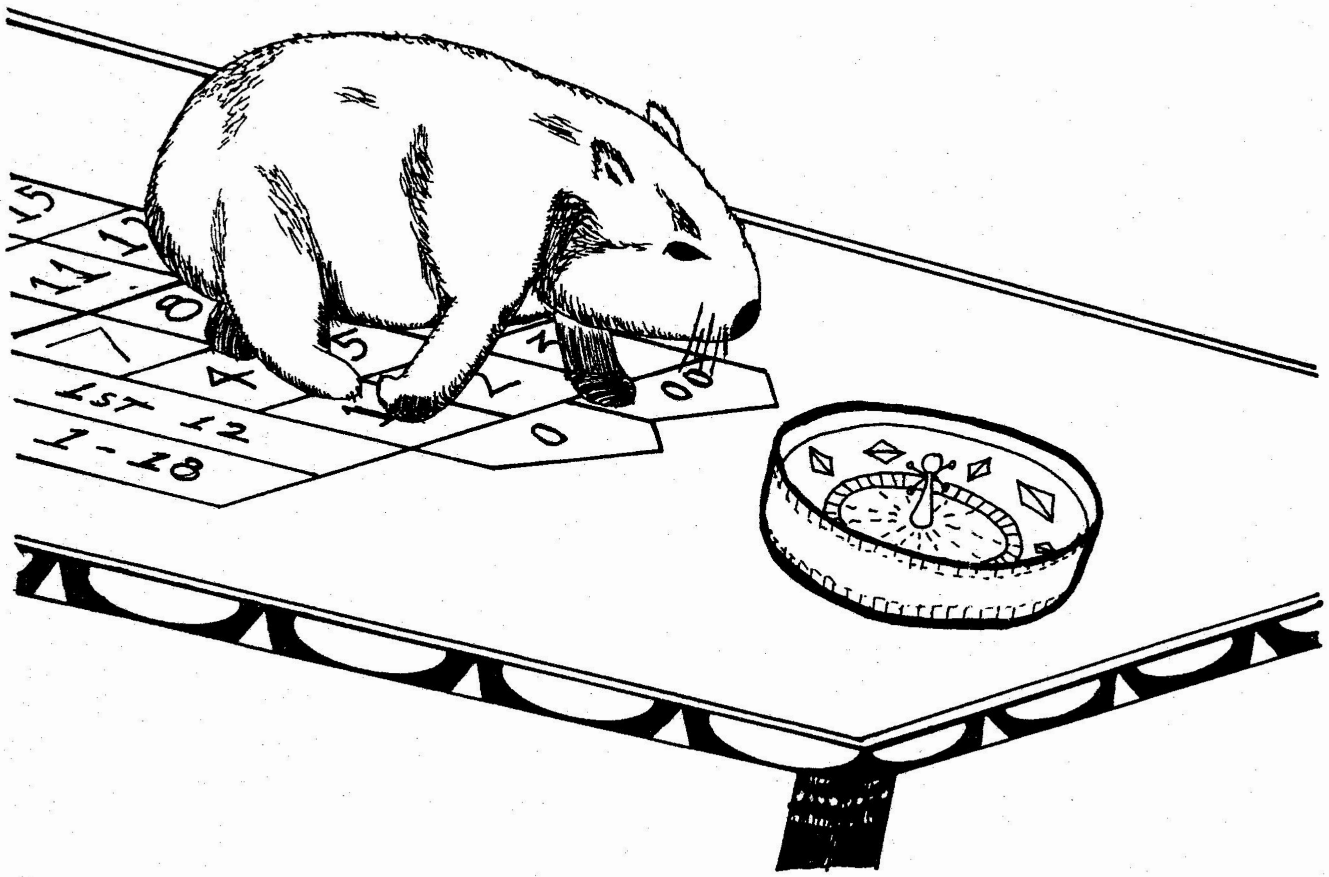
The Wombat EXAMINER

"Increases the Circulation of Anyone in America" Volume 5 Number 3

January 1984



Printed in the U.S.A.

CONTRIBUTIONS

Contributions for the newsletter can be sent to either of the following addresses:

Editor, Datatrieve Newsletter
c/o DECUS, One Iron Way
MR2-3/E55
Marlboro, MA 17562

Virginia Sventek
Datatrieve Newsletter Editor
Computer Science and Mathematics
Building 50B, Room 3238
Lawrence Berkeley Laboratory
Berkeley, California 94720

Letters and articles for publication are requested from members of the SIG. They may include helpful hints, inquiries to other users, reports on SIG business, summaries of SPRs submitted to Digital or other information for members of the Datatrieve SIG. Camera copy is appreciated, machine readable input is highly desired,
but almost anything legible will be considered.
However, this newsletter is not a forum for job and/or head hunting, nor is commercialism appropriate.

TABLE OF CONTENTS

ABOUT THE COVER

This month's cover was drawn by Bart Lederman of ITT. There we were, poor little wombats stumbling around the gaming tables.

Improving Performance of RMS ISAM Files

Harold T. Glaser
Supervising Software Engineer
James M. Montgomery, Consulting Engineers, Inc.
250 North Madison Avenue
Pasadena, California 91101

Introduction

The use of indexed files offers significant gains in overall performance over ordinary sequential files in many Datatrieve applications. Judicious selection of the file design and key structure will improve the speed and reduce the performance with tradeoffs in file size and overall database complexity. At JMM, we have been making extensive use of ISAM files for both small and large databases with impressive results. However, there are certain considerations with respect to the maintenance of these files which dramatically affect performance. We found this to be especially true for ISAM files which are subject to frequent update and modification. Two approaches for reloading of ISAM files are presented, the DTR restructure and the RMS CONVERT utility.

Getting Started with Indexed Files

The advantages of indexed files (ISAM) over ordinary sequential files have been demonstrated before. If the database designer selects a key structure which takes advantage of common field queries during record access, most applications will see a reduction in CPU and elapsed time. Also, ISAM files allow the user to ERASE records from DTR, something you can't do with sequential files. This is somewhat at the sacrifice of disk storage overhead and overall complexity of the database. The tradeoff becomes significant for larger databases (greater than 1,000 records or so, but don't pin me down to this) and in most cases, it's safe to say that indexing a file will be better than leaving it sequential. Of course, the big advantage to using Datatrieve with ISAM files is that the whole thing is transparent to the user and relatively simple to set up by the designer or applications programmer. As an example, consider the sequential domain RESULTS which will be loaded to an indexed file in the following manner:

```
Domain/Record/File Definitions:

DTR> DEFINE DOMAIN RESULTS USING
          RESULT ON RESULTS.DAT;
DTR> DEFINE RECORD RESULT USING
      01  DATA.
          03  PRIMARY_KEY
              PIC 9(4).
          03  ALTERNATE_KEY
              PIC 9(2).
          03  VALUE
              USAGE IS REAL.
      ;
DTR> DEFINE FILE FOR RESULTS          ! RESULTS is a sequential file.
```

Now setting up the new indexed file and loading taking advantage of the VAX-11 DTR alias feature for readying domains and the DTR restructure:

```
DTR> READY RESULTS AS OLD READ
DTR> DEFINE FILE FOR RESULTS,
          KEY = PRIMARY_KEY ( NO CHANGE, DUP ),
          KEY = ALTERNATE_KEY ( CHANGE, DUP )
DTR> READY RESULTS AS NEW WRITE
DTR> NEW = OLD                        ! DTR restructure
DTR> FINISH ALL                       ! Cleaning up
DTR> READY RESULTS READ               ! Now indexed version
```

At this point there is a new indexed file, RESULTS.DAT with a version number one higher than the sequential version. NOTE: This method of loading indexed files will work well if you have relatively few records. If you have more than a few hundred records, keep reading!

Using Indexed Files

As stated previously, the indexed version of the file almost always offers performance gains over the ordinary sequential file. However, we have found that indexed files subject to updating in one or more of the following ways require special attention:

(1) Storing new records

Random storage of new records may affect the primary key structure, especially in instances where it causes an RMS key bucket split. (See the RMS Tuning Guide for more information on buckets and bucket splitting.) If the key structure if affected, then additional CPU time and disk storage is required to access the records inserted in this manner as opposed to those which didn't require a bucket split. Modification of the alternate key also has a similar effect on the file.

(2) Erasing records

In general, erasing records does not release usable storage space in indexed files (although this should be qualified by saying that in special cases it is possible to reclaim the space). The reference to the record is merely deleted. After a number of updates, the file record density becomes sparse and disk space is wasted. Again, additional CPU time is required to deal with records which have been erased.

(3) Contiguity/File Extensions

These two properties are important to the efficiency of access to your disk file. Clearly, the less work required by the disk controller in finding your data, the more efficient your application. If the file resides in contiguous space, less work is required. Also, extensions of the file size with additional records should be optimized to match your record size and application.

These are just a few of the reasons why indexed files require special attention. Our experience shows that indexed files which have been modified or updated as outlined above tend to lose the inital performance gains over sequential files. This can be corrected by improving file design and maintenance.

The VAX-11 RMS Tuning Guide discusses these and other factors such as I/O units, blocks, buckets, areas, fill factors and buffers with an elegance not possible here. Because of limited space, I'll show two alternatives for maintaining indexed files we have tried at JMM. One is the obvious way, the other is the RIGHT way.

DTR Restructure of Indexed Files

In this method, Datatrieve restructure is used to load a new indexed file. Once the file is loaded, the old file is replaced. The key to this method is that all record additions are done to an empty sequential file with the same record definition as the indexed file which makes storage much faster. This file is merged with the indexed file by downloading the indexed file on top of the updated records in sequential format. The merged file is then restructured to a new indexed file.

```
Domain/Record/File Definitions:

DTR> DEFINE DOMAIN UPDATE USING
        RESULT ON UPDATE.SEQ;          ! Same record definition.
DTR> DEFINE FILE FOR UPDATE            ! Sequential
DTR> READY UPDATE WRITE                ! New records are stored in UPDATE
                                       ! instead of RESULTS
DTR> READY RESULTS SHARED READ         ! Our sequential file from before
                                       ! with assorted new records,
                                       ! erasures, alternate key modifys
DTR> UPDATE = RESULTS                  ! Dumping RESULTS down to UPDATE
DTR> DEFINE FILE FOR RESULTS,
        KEY = PRIMARY_KEY ( NO CHANGE, DUP ),
        KEY = ALTERNATE_KEY ( CHANGE, DUP )
DTR> READY RESULTS AS NEW WRITE
DTR> NEW = UPDATE                      ! DTR restructure
DTR> FINISH ALL                        ! Cleaning up
DTR> READY RESULTS READ                ! New updated indexed version
```

This new file now contains all of the records from the original indexed RESULTS file as well as all of the new records contained in UPDATE. One modification to the procedure which will slow things down but produce a better RESULTS file is to sort the merged UPDATE domain before the restructure is done. By experience, however, we've found that this whole approach is only marginally better than living with the original messy indexed file. About the only thing that is better about the new file is that it doesn't have ERASEd records in it - that's why we even bothered to add the UPDATE domain instead of just adding records to the original RESULTS domain. Depending on your application, this may or may not be worth the effort.

The restructure takes an awfully long time (if you have more than a few hundred records) and produces an indexed file which is not at all optimal. This is because Datatrieve uses RMS to build the new indexed file without telling RMS anything about how that file should be built in an optimal manner. Datatrieve, or any other language using RMS for that matter, writes each record one at a time, using RMS $PUTs. Thus, since RMS doesn't really know very much about where the index bucket boundaries go, it does some pretty simple-minded stuff and takes a long time to produce a non-optimal file. (If you non-believers think that the problem is with Datatrieve, we suggest you try writing a program in your favorite procedural language to write the file. You'll get the same crummy file, and it will take just as long to run the program as it took to run the DTR procedure. If you really want to grind a 11/780 to a halt, try loading a few thousand records into an ISAM file, one record at a time. This method of killing a VAX works best if you have lots of keys and records with lots of duplicates.) If you want an efficient file and minimal CPU time, we recommend the following approach instead.

RMS Convert of Indexed Files

In Version 3.0 or later of VMS, an RMS utility CONVERT has been provided which does just this type of thing very well. It bypasses many of the RMS standard calls which makes it efficient in execution. It also offers many options and switches which have satisfied practically every strange nuance we could think of so far. PDP-11 users are not lost. They should refer to the IFL or Indexed File Loader utility for specifc details. Although this example is built around VMS CONVERT, the same principles apply using IFL. In fact, we used IFL in versions of VMS before 3.0. (VMS users take note - CONVERT is much faster than IFL, and relatively bug-free. Don't use IFL unless you're on a PDP-11.)

To make this example a little more interesting, we'll also assume the old file has to be available for usage while the update is in progress, a fairly reasonable demand.

The first step is to merge the files together into a sequential file as we did in the previous example:

```
Domain/File Definitions:

DTR>    DEFINE DOMAIN UPDATE USING          ! All of this is the same
           RESULT ON UPDATE.SEQ;            ! as before.
DTR>    DEFINE FILE FOR UPDATE;             ! Sequential file
DTR>    READY UPDATE WRITE                  ! New records to UPDATE
DTR>    READY RESULTS SHARED READ           ! RESULTS indexed file
DTR>    UPDATE = RESULTS                    ! RESULTS records to UPDATE
DTR>    EXIT
```

At this point, we do not create a new RESULTS file, nor do we load the file using Datatrieve restructure as we did before. Instead, we first invoke the RMS ANALYZE utility to create an FDL file of RESULTS. An FDL (File Definition Language) file is an ordered sequence of file attribute keywords and associated values which lets you access RMS capabilities in a very simple way. This FDL file is then used by the CONVERT utility to create a new indexed version of UPDATE, named RESULTS.DAT. Once CONVERT is complete, the old RESULTS file is replaced with the new combined file.

```
$       ANALYZE/RMS/FDL RESULTS.DAT
$       CONVERT/STATISTICS/FDL=RESULTS UPDATE.SEQ RESULTS.DAT
```

The new indexed RESULTS file created by CONVERT should have a cleaner key structure, be free of erased records and will probably be more compact in terms of disk space. Also, applications should see an improvement in performance over the old version.

The next level of sophistication is to "tune" the indexed file. You can do this using the EDIT/FDL utility together with the RESULTS.FDL file produced by the ANALYZE/RMS/FDL step. If you use the EDIT/FDL utility, you usually would keep an FDL file that had the "tuning" information in it. Thus, you would not need to run the ANALYZE utility each time you reloaded the file - only if you made changes to the key structure or drasitically changed the nature of the data in the file or it's size. The RMS Tuning Guide has more information on tuning ISAM files.

Case Study

How does this all look when you put it together? Here is a case study of an actual DCL command procedure that uses CONVERT. Note that in this example we skip the ANALYZE step (ok, we're lazy!) and we also don't tune the file. Instead, we just take the RMS default attributes that Datatrieve uses when you invoke the DEFINE FILE command. The DTR defaults basically translate to a default bucket size of 2 and default bucket fill of 50%. By skipping the ANALYZE step, we may lose quite a bit in performance, but generally it's not a big problem. You will want to use ANALYZE and CONVERT/FDL if you have a really big file though, or if performance is really critical.

```
$! BUILD.COM - The easy way
$! This command file takes a current copy of the Public Employee File and
$! builds it into the old format for use in existing programs.  Thus, the
$! data in the old files is a strict subset of the data in the new file.
$!
$! Pam and Phil - Tue Nov 30 23:11:45 1982
$!
$ SET DEFAULT CC_11140:[PUBLIC.PERSONAL.DATA]
$ DTR
!Build the public employee file...
SET DICTIONARY CDD$TOP.CC11140.PUBLIC.PERSONAL
READY NEW_EMPLOYEES AS NEW_FORMAT SHARED
DEFINE FILE FOR EMPLOYEES, ALLOCATION = 450; !Sequential file
READY EMPLOYEES AS OLD_FORMAT WRITE
OLD_FORMAT = NEW_FORMAT
! Now, have DTR define another file.  This one will have the DTR defaults
! for bucket sizes and fill percentages, but it's very easy to define from
! inside DTR and I don't have to know anything about using ANALYZE/RMS or
! the /FDL option in CONVERT.
DEFINE FILE FOR EMPLOYEES,
        KEY = EMPNO (NO DUP),                !Can't change the primary key.
        KEY = LAST_NAME (CHANGE, DUP),       !Pick any keys you want - it
        KEY = COMP_INITS (CHANGE, DUP),      !doesn't matter since CONVERT
        KEY = SUP_NUMBER (CHANGE, DUP),      !does all the work.
        KEY = COST_CENTER (CHANGE, DUP),
        ALLOCATION = 600;                    !Note that we allow more space
                                             !for the ISAM version, because
                                             !we have bunches of keys.
```

```
!
FINISH
EXIT
$! Now convert the file to ISAM.  We can use the /NOCREATE option
$! of CONVERT because we just finished creating the file we want using
$! DTR.  Instead, we could have used CONVERT/FDL.
$ CONVERT/STATISTICS/NOCREATE EMPLOYEES.DAT;-1 EMPLOYEES.DAT;0
$ PURGE CC_11140:[PUBLIC.PERSONAL.DATA]EMPLOYEES.DAT
$ EXIT
```

Comments

We have found the CONVERT utility to compare very favorably with other techniques for loading indexed files. Typically it offers a 3 to 1 or higher advantage in CPU time over a Datatrieve restructure. The difference in elapsed time is even more impressive. It is especially fast if you are building an ISAM file that would cause many bucket splits if created using a record-oriented approach, even if that approach is implemented in a procedural language such as COBOL. All of this, of course, depends on your system hardware configuration, disk drive performance, size of database, system load, sysgen and RMS parameters and specific file attributes. We suggest you experiment to improve performance over the long haul. If updates to your database are fairly frequent or routine, the whole thing can be automated with a batch job that runs at off-hours. If you want or need to restructure your database only occasionally, the simple method shown in the example is very easy to do by hand, and doesn't require great knowledge of CONVERT or ANALYSE.

At JMM, we've done a lot of work to tune our large ISAM files and to develop fast and simple methods of building ISAM databases. In talks planned for the next DECUS, we will present some other hints on loading ISAM files, file design, file tuning, and present some benchmarks that show what improvements in load time and retrieval you can expect.

Some Examples Using a Single Hierarchical Domain

Samuel Pitluck
Lawrence Berkeley Laboratory
Berkeley California

For the past five years we have been using Datatrieve to maintain information about patients undergoing experimental radiotherapy cancer treatments. We started using Datatrieve on a PDP-11 and migrated to a VAX 11/780 in 1981. We first used the compatability mode Datatrieve on the VAX and finally changed over to the native mode VAX-11 Datatrieve.

We started using Datatrieve with just a single domain. When the variable - occurs clause became part of Datatrieve we continued to use the single domain technique. One reason for this was that at the time it seemed to be the fastest way to implement our rapidly changing record definitions. In addition, over a three year period our users had grown accustommed to using a single domain.

The examples described in this paper will be based on the following domain and record definitions listed in part below:

Domain definition:

```
                DEFINE DOMAIN PATS USING PAT_REC ON PATS.DAT
```

Record definition:

```
                DEFINE RECORD PAT_REC
                 USING
01 PATIENT.
        03 IDENT.
                06 YR PIC 99.
                06 ID PIC 999.
                06 NAME PIC X(20).
        03 DIAG.
                06 GRADE PIC X(6).
                06 STAGE PIC X(6).
                06 RX_SITE PIC X(20).
                        .
                        .
                        .
        03 DATES.
                06 RXBGN USAGE DATE EDIT_STRING IS NN/DD/YY.
                06 FOLLOW USAGE DATE EDIT_STRING IS NN/DD/YY.
                06 APPT USAGE DATE EDIT_STRING IS NN/DD/YY.
                06 DOB USAGE DATE EDIT_STRING IS NN/DD/YY.
                06 AGE COMPUTED BY FN$NINT ((FOLLOW - DOB)/365.25)
                   EDIT_STRING IS Z9.
       03 NUMBER_IONS PIC 9
          QUERY_NAME IS NIONS.
       03 IONS OCCURS 0 TO 7 TIMES DEPENDING ON NUMBER_IONS.
          05 EACH_ION.
                06 ION PIC X.
                06 DOSE PIC 9999
                         EDIT_STRING IS ZZZZ.
                06 FX    PIC 9999
                         EDIT_STRING IS ZZZZ.
                06 DAYS PIC 9999
                         EDIT_STRING IS ZZZZ.
                06 RBE  PIC 9V99
                         EDIT_STRING IS Z.ZZ.
                06 CDOSE PIC 9999
                                EDIT_STRING IS ZZZZ.
;
```

Examples:

A. Data storage:

The initial storage of data into a record is done via a procedure that promts the user for only certain fields in the record definition. The technique is taken from the example described by Joan Hilton and Jean Lemmon in the Wombat Examiner V4 #1 page 86 (Feb. 1982). For those who do not have access to older Wombat Examiners, we list our definitions below. First, we must define a domain and record definition which is a duplicate of the variable-occurs portion of the patient record definition. These definitions are defined once and placed into our dictionary. The domain and record definitions respectively are:

```
       DOMAIN IONTEMP USING IONTEMPR ON IONTEMP.DAT

                DEFINE RECORD IONTEMPR
                 USING
       01 IONS.
                06 ION PIC X.
                06 DOSE PIC 9999
                         EDIT-STRING IS ZZZZ.
                06 FX    PIC 9999
                         EDIT-STRING IS ZZZZ.
                06 DAYS PIC 9999
                EDIT-STRING IS ZZZZ.
                06 RBE  PIC 9V99
```

```
                              EDIT-STRING IS Z.ZZ.
                    Ø6 CDOSE PIC 9999
                                   EDIT-STRING IS ZZZZ.
;
```

In order to store selectively into our patient database we invoke the following procedure:

```
DEFINE PROCEDURE SPATS
! Define a new empty iontemp file
DEFINE FILE IONTEMP KEY=ION (DUP),SUPERSEDE;
READY PATS WRITE
READY IONTEMP WRITE
! The "store A in ..." allows you to reference the newly
!  entered data in A.field-name later on
        STORE A IN PATS USING
        BEGIN
                YR=*.YR
                ID=*.ID
                NAME=*.NAME
                RX-SITE=*.TREATMENT-SITE
                DOB = *."DATE OF BIRTH"
                RXBGN=*.RXBGN
                FOLLOW="       "
                APPT="       "
                        .
                        .
                        .
                NUMBER_IONS = *."NUMBER OF IONS"
! Use the number_ions just entered to control how many records are
!  stored in the iontemp file
                REPEAT A.NUMBER_IONS STORE IONTEMP USING BEGIN
                  ION=*.ION
                  DOSE=*.DOSE
                  FX=*.FRACTIONS
                  DAYS=*.DAYS
                  RBE=*.RBE
                  CDOSE=*.CDOSE
                END
! Copy the record in the iontemp file to the ions portion of
!  the main file (pats)
                ions = iontemp
! Clean out iontemp for the next pass through
                ERASE ALL OF IONTEMP

        END
        FINISH IONTEMP
END-PROCEDURE
```

B. Modifying a field calculated from other fields in the occurs clause

One of the quantities in the list is RBE. This field is calculated from other fields in the list. One can modify this field for each ion in the following way:

```
DTR> FIND PATS WITH YR=83 AND ID=100      ! find record of interest
DTR> SELECT                               ! select the record
DTR> FIND IONS                            ! make list current collection
DTR> :RBE                                 ! invoke procedure
```

where the procedure RBE is:

```
DEFINE PROCEDURE RBE
 DECLARE RB PIC 9V99.
 FOR CURRENT BEGIN
  IF DOSE GT 10 THEN
   BEGIN
     SELECT
     RB = CDOSE/DOSE
     MODIFY USING BEGIN
       RBE = RB
     END
   END
 END
END-PROCEDURE
```

Note that in the above procedure it is possible to test an individual field before carrying out the calculation. It is also possible to find the sum of a field for each entry in the list. For example, if you want to sum all the CDOSE values in the list you would proceed as follows:

```
DTR> FIND PATS WITH YR=83 AND ID=100      ! find record of interest
DTR> SELECT
DTR> :SUMCDOSE
```

where the procedure sumcdose is:

```
DEFINE PROCEDURE SUMCDOSE
  DECLARE S1 PIC 9999.
  S1 = 0
  FOR IONS BEGIN
   S1 = S1 + CDOSE
  END
  PRINT S1
END-PROCEDURE
```

C. Making collections based on fields in the occurs clause.

In our application we frequently need to make collections based on the type of ION with which we treat the patient. Below is a procedure that will find all patients treated with one particular ION.

```
DEFINE PROCEDURE FION
!
! This procedure will locate all entries with only 1 ion entry and type
!
  DECLARE ITYPE PIC X.
  ITYPE =*."ion type"
  FIND PATS WITH NIONS=1 AND ANY IONS WITH ION=ITYPE
!
END-PROCEDURE
```

Note that in order to promt for the type of ION to search for we had to define a new variable, ITYPE. The following command does not work!

```
DTR>FIND PATS WITH NIONS=1 AND ANY IONS WITH ION=*."ion type"
```

Similarly, one can make collections based on any two types of ions. Below is such a procedure:

```
DEFINE PROCEDURE F2ION
  !
  ! This procedure will locate all entries with 2 different ion entries
  !
  DECLARE ITYPE1 PIC X.
  DECLARE ITYPE2 PIC X.
  ITYPE1 =*."ion type 1"
  ITYPE2 =*."ion type 2"
  FIND PATS WITH NIONS=2 AND ANY IONS WITH ION=ITYPE1 AND ANY IONS WITH -
  ION=ITYPE2
END-PROCEDURE
```

Suppose we want to find all patients treated with either ION = H or ION = H and P. The following procedure will do this:

```
DEFINE PROCEDURE LIGHT
!
! Procedure to find patients treated with light ions
!
  FIND PATS WITH (NIONS=1 AND ANY IONS WITH ION="H") OR -
  NIONS=2 AND ANY IONS WITH ION="H" AND ANY IONS WITH ION="P"
END_PROCEDURE
```

Note that the parentheses are necessary!

Suppose we want to find all patients that were not treated with either ION = H or ION = P. The following procedure will do this:

```
DEFINE PROCEDURE HEAVY
!
! Procedure to find patients treated with heavy ions only
!
  FIND CANCS WTIH (NOT ANY IONS WITH ION="H") AND -
                  (NOT ANY IONS WITH ION="P")
END_PROCEDURE
```

```
define record THE_THREE_LITTLE_WOMBATS
   01 ooser.
      03 wombats occurs 3 times.
         05 story computed by Dan Dietterich.
;
```

Once upon a time there was small warren named Yachts. And in this warren lived three wombats named Albin, Grampian and Pearson. Now these wombats were not of the owning class, but were your basic working class sort of wombats. They were database administrators. They spoke the language Datatrieve, with dialectic variations, of course.

Now in the countryside around the warren there lived two kinds of beasties. The first kind were the Roos. A Roos was large, brown and hairy with big hind legs and a long tail and went hopping about. A Mama Roos carried her young in her pouch while hopping. A Papa Roos didn't. Since wombats also carried their young in pouches, they thought they might be descended from the Roos. And the wombats did indeed descend into their warren when the Roos came around. In any event, the Roos were mostly harmless, except for being a bit jumpy.

The second kind of beasty was the Ooser. Oosers came in a greater variety than Roos, but some were large, brown and hairy. Others were short, stout and wore bifocals. Oosers spent their time Working on Things. They liked to use compooter terminals and query databases. The concept, the central idea that all Oosers held tightly in their minds, hearts and souls was the Answer: the Answer to the Query. They were quite ferocious, especially when provoked. And nothing could provoke an Ooser like a query that was hard to answer.

Well, one night Albin, Grampian and Pearson were out catching some moon. It was a warm, lazy night - just the kind night for some grooming and simple conversation. Mostly DISPLAY statements. When a rumbling rose up and seemed to surround them. They leapt up and scattered, crying DISPLAY "Oosers! Oosers!"; but it was all just a Roos.

They re-assembled and conversed. Although they weren't in any danger from the Roos, if it had been a real Ooser, they wouldn't have been able to answer a single query. You see, they were rather young wombats and had not yet built their databases. So, they made a pact that night to build robust databases and fend off the Oosers.

Albin decided to build a database of straw. And his database looked like this:

```
DEFINE DOMAIN STRAW USING STRAW-REC ON STRAW.DAT;
DEFINE RECORD STRAW-REC USING
01 LAST_STRAW.
   03 ID LONG.
   03 DESCRIPTION PIC X(40).
   03 USE OCCURS 5 TIMES.
       06 WHERE PIC X(20).
       06 WHAT_FOR PIC X(20).
;
```

Well, Albin had constructed his database when a particularly large, brown and hairy Ooser came by. The Ooser said, "Tell me where you used the straw from the southwest field" Well, Albin was trembling with fear, but he managed to key in:

```
READY STRAW
PRINT STRAW WITH DESCRIPTION = "southwest field"
```

The answer came out:

```
                                              WHAT
ID         DESCRIPTION                WHERE   FOR

 4457   southwest field            Pearson's  floormat
                                   Grampian's drapes
                                   Albin's    database
```

He breathed a sigh of relief, but the Ooser just glared back and said, "Oooo. Here is a single piece of straw from the northeast field I've been chewing on. Can you add it to your database?" Albin was shaking almost uncontrollably, but typed:

```
    READY STRAW WRITE
    STORE STRAW
Enter ID:
```

He was just about to enter an ID when the Ooser growled said, "I don't want prompts. Just store the straw directly." Now Albin was in a cold sweat. He typed:

```
STORE STRAW USING
   BEGIN
   ID = "986"
   DESCRIPTION = "northeast field"
   WHERE = "Ooser's mouth"
   WHAT_FOR = "Chewing"
   END
```

but alas...

```
    "WHERE" is undefined or used out of context.
```

and that was the end of Albin's database. There is no way of storing into the OCCURS list directly.

Grampian decided to build her database of wood. She had heard about Albin's sad fate, so she got out her hack saw and set to work constructing a better database. Her database was:

```
DEFINE DOMAIN WOOD USING WOOD-REC ON WOOD.DAT;
DEFINE RECORD WOOD-REC USING
01 WOOD_HAVE_BEEN.
   03 ID LONG.
   03 DESCRIPTION PIC X(40).
   03 USE OCCURS 5 TIMES.
      06 WHERE PIC X(20).
      06 WHAT_FOR PIC X(20).
   03 USED REDEFINES USE.
      06 WHERE1 PIC X(20).
      06 WHAT_FOR1 PIC X(20).
      06 WHERE2 PIC X(20).
      06 WHAT_FOR2 PIC X(20).
      06 WHERE3 PIC X(20).
      06 WHAT_FOR3 PIC X(20).
      06 WHERE4 PIC X(20).
      06 WHAT_FOR4 PIC X(20).
      06 WHERE5 PIC X(20).
      06 WHAT_FOR5 PIC X(20).
;
```

She had barely gotten the database loaded when up came a lonesome Ooser looking for someone to query.

"Tell me where you used the wood from the western forests."

Grampian, smiled and typed:

```
READY WOOD
PRINT WOOD WITH DESCRIPTION = "western forests"
```

No problem. The Ooser said, "Ooo. Now here is a piece of wood from the eastern forest. I've been gnawing on it. Store it in your database. And by the way, don't use prompts." Grampian tensed a bit, but thought her database would hold up:

```
READY WOOD WRITE
STORE WOOD USING
  BEGIN
  ID = "1234"
  DESCRIPTION = "western forest"
  WHERE1 = "Ooser's mouth"
  WHAT_FOR1 = "Gnawing"
  END
```

A brief pause...

...and then...

```
DTR>
```

She relaxed. It would be alright. But then the Ooser grimaced and said, "Ooo. Tell me what you use pieces of wood for." She thought for a piece and then typed:

```
PRINT ALL ALL WHAT_FOR OF USE OF WOOD
```

And the answer came out

```
   WHAT
   FOR

house
barn
lean-to

house
privy

table

gnawing
```

She thought she was safe, but the Ooser grumbled, "You call that a query? That's a mess." She said, "Don't do anything rash. I'll do it again."

```
    PRINT WOOD CROSS USE REDUCED TO WHAT_FOR

  WHAT
  FOR

barn
gnawing
house
lean-to
privy
table
```

"Ooo, that's better.", said the Ooser, "but you waste a lot of space storing duplicate items. You have two 'house' stored. Worse yet, what if I have more than 5? Now tell me for each of those uses, the description of the wood." Grampian was shaken, but continued to think. And she thought...

...and she thought...

...and she thought...

...and finally typed:

```
FOR WOOD CROSS A IN USE REDUCED TO WHAT_FOR
  BEGIN
  PRINT WHAT_FOR
  FOR WOOD WITH ANY USE WITH WHAT_FOR = A.WHAT_FOR
      PRINT COL 30, DESCRIPTION
  END
```

"Ooo, yuck!", said the Ooser, "That's a terrible query. You need to loop through the data twice to get the answer!" Grampian stiffened and watched through tearless eyes as her database crumbled.

Now Pearson heard about poor Albin and Grampian and was determined to do better. She took a different approach and defined a database of bricks.

```
DEFINE DOMAIN BRICK USING BRICK-REC ON BRICK.DAT;
DEFINE RECORD BRICK-REC USING
01 BRICK_IT.
   03 ID LONG.
   03 DESCRIPTION PIC X(40).
;

DEFINE DOMAIN USES USING USE-REC ON USES.DAT;
DEFINE RECORD USE-REC USING
01 USE_IT.
   03 ID LONG.
   03 WHERE PIC X(20).
   03 WHAT_FOR PIC X(20).
;
```

Can you guess what is about to happen? Well sure enough, along comes one of those short, stocky Oosers with the bifocals. And the Ooser says, "Tell me what your red bricks are used for." Pearson calmly typed:

```
        PRINT WHAT_FOR OF BRICK CROSS USES OVER ID WITH DESCRIPTION = "RED"
```

"Oooo. Very nice.", said the Ooser, squinting to stare at the compooter terminal. "Now, I have a brick. I use it to smush bugs. Store it into your database. And no prompts, understand?" Pearson understood all too well, but simply typed:

```
    STORE BRICK USING
       BEGIN
       ID = 123
       DESCRIPTION = "bug"
       END

    STORE USES USING
       BEGIN
       ID = 123
       WHERE = "In Ooser's hand"
       WHAT_FOR = "Smushing"
       END
```

"Oooo", said the Ooser, "I don't like the extra STORE, but you can add as many uses as you like. What about my other brick? I use it for smushing eggplants. Store that." Pearson went ahead and typed

```
    STORE BRICK USING
       BEGIN
       ID = 123
       DESCRIPTION = "eggplant"
       END
```

"Oooo. Only one store. Now tell me what bricks are used for." Pearson smiled, her database was in normal form after all:

```
    PRINT WHAT_FOR OF USES
```

"Oooo my", murmered the Ooser. "Tell me for each use what the description of the brick is." Pearson wiped her brow and set to work.

```
    FOR USES
       BEGIN
       PRINT WHAT_FOR
       FOR BRICK WITH ID = USES.ID
           PRINT COL 30, DESCRIPTION
       END
```

And the Ooser trundled away saying, "Ooooo! Ooooo! Ooo, my." That was the End Ooser and that is the end of the story.

How to Write Plots in DTR

Don Becker
Atlas Steels
Welland, Ontario

1.0 Introduction

There is currently no documentation on how to write plots for DTR. Given a little time to become familiar with the plots supplied with DTR, every user comes up with their own pet plot they would like implemented.

In this article I hope to explain enough to get you started in producing DTR plots.

This article will assume a knowledge of DTR procedures.

At Atlas Steels we have implemented plots for 2 reasons

1. Some simple plots for use on non-graphics terminals (VT100, VT52).

2. To implement some analysis unique to Atlas Steels.

2.0 Entry Points Unlike procedures, plots have multiple entry points.

Entry points have parameters (similar to procedures in Pascal, Fortran, etc.) that allow a more flexible interface than DTR procedures. The form of an entry statement is

```
        ENTRY N [( P1 [:  TYPE ] [, Pn [:  TYPE] ])]
```

where

1. N is an integer literal

2. P1 through Pn are variable names

3. TYPE is one of the valid data types (see below)

4. STATEMENT is one of the valid statements, including the BEGIN-END compound statement

An example of an entry statement would be

```
ENTRY 1 ( X :REAL, Y :REAL, CLASS :STRING )
```

2.1 Parameters

Parameters are declared in the Entry point statement. Each parameter must have a data type (see below). A colon (":") separates the parameter from its type specifier.

2.2 Entry Point 0

Entry point 0 is used to initialize the plot. It is called with each of the parameters on the DTR command line (DTR> PLOT ...) named in a string. Thus the number of parameters (type STRING) in the ENTRY 0 statement defines how many parameters the user must specify to execute the plot.

This entry point is called only once during the execution of the plot.

2.3 Entry Point 1

Entry Point 1 is called once for each set of values that are to be entered in the plot function. The data types match the data types of those parameters NAMED in Entry point 0.

2.4 Entry Point 2

Entry Point 2 is called once during the execution of a plot. It is called without parameters. It is used to wrap up the execution of the plot before returning to DTR command level.

3.0 Declarations

Plots have a GLOBAL definition area (besides the definitions included as parameters to entry points). This area immediately follows the plot definition statement and precedes the first entry point definition (0).

3.1 Data Types

DTR supplies 3 data types, REAL, DATE, and STRING, to plot definitions.

A variable can be declared by a statement such as

```
DECLARE TYPE LABEL [, LABEL...]
```

where TYPE is REAL, DATE, or STRING. (the default type is REAL so the "REAL" is optional).

3.2 Data Structures

DTR plots allow for the use of dynamic arrays. By this I mean that there is no need to define the length of an array, each element of the array is created by assigning a value into it.

An array definition would look like

```
        DECLARE TYPE VECTOR LABEL [, LABEL...]
```

Vectors can be indexed by literal integers (notice no integer date types) or reals. The reals are rounded to the nearest integer before being used as an index. Thus for a N element vector, the valid indexes I are 0.5 <= I < N+0.5 .

These arrays are made useful by a SIZE function and a INCR I OVER V statement (see later).

This is a major improvement over DTR procedures.

4.0 Statements

Plots do not have the same set of statements as procedures, however many of them do look similar.

Plots do have

1. assignment statement
```
            X = 55.6
```

2. compound statement
```
            BEGIN
                X = 55.6
                Y = 99
            END
```

3. if-then-else statement
```
            IF X < 55 THEN
            BEGIN
                X = X + 55
                Y = X / 3
            END
            ELSE Y = X * 2
```

4. print statement

```
        PRINT 'P[250,44]T',QUOTE('Text to go on the plot')
```

5. plot statement

```
    PLOT LABEL 2 ( X_MIN, Y_MIN, X_VECTOR )
    (See below)
```

6. sort statement

```
    SORT( X_VECTOR )
```

7. incr statement

```
        INCR I FROM 1 TO 100
        or
        INCR I OVER X_VECTOR
    (I varies from 1 to SIZE( X_VECTOR ) )
```

5.0 Expressions

Plots do have REAL expressions with the normal operators (including exponentiation **) however they do not have concatenation operators for strings.

A full set of relational operators is not implemented (>, <, LT, LE, EQ, GE, GT are).

6.0 Functions

There are a number of functions that can be called. However the user defined functions, and the DTR defined functions cannot be called.

6.1 Size

Size takes an array as a parameter and returns the number of elements in the array.

6.2 SQRT

SQRT takes a REAL expression as a parameter and returns the square root of the expression.

6.3 MIN

MIN takes an array as a parameter an returns the minimum value of its elements.

6.4 MAX

MAX takes an array as a parameter and returns the maximum value of its elements.

6.5 CVT

CVT takes a REAL as a parameter and returns it as a string.

6.6 LOG

LOG takes a REAL as a parameter and returns its natural log.

6.7 INT

INT takes a REAL as a parameter and returns the closest integer.

6.8 CENTER

CENTER takes 4 values

1. A REAL value giving a X coordinate.
2. A REAL value giving a Y coordinate.
3. A STRING whose upper left corner is to be placed at the coordinate
4. The size of the characters to be displayed (in pixels).

and returns a string representing the REGIS commands to display the string.

6.9 LXY

LXY takes 2 REAL parameters, the X and Y coordinates, and returns a string representing the absolute REGIS coordinates [X, Y].

6.10 LX

LX takes 1 REAL parameter, the X coordinate, and returns a string representing the absolute REGIS coordinates [X].

6.11 LY

LY takes 1 REAL parameter, the Y coordinate, and returns a string representing the absolute REGIS coordinates [, Y].

6.12 RXY

RXY takes 2 REAL parameters, the X and Y coordinates, and returns a string representing the relative REGIS coordinates [+X, +Y].

6.13 RX

RX takes 1 REAL parameter, the X coordinate, and returns a string representing the absolute REGIS coordinates [+X].

6.14 RY

RY takes 1 REAL parameter, the Y coordinate, and returns a string representing the absolute REGIS coordinates [, +Y].

6.15 QUOTE

QUOTE takes a STRING as a parameter and returns the string surrounded by quotes (necessary to insert it in a REGIS command).

6.16 SEARCH

SEARCH takes a string, and an array of strings as parameters and returns the index of the string in the array, or 0 if it is not found in the array.

7.0 Utility Plots

Utility plots are plots with whose entry points can be called directly from the inside of another plot by using the PLOT statement.

The PLOT statement is of the form

```
          PLOT PLOT-NAME N ( PARAMETER-1 [, PARAMETER-N...])
```

where N is a literal integer representing a entry point in the named plot.

7.1 HOUSEKEEP

HOUSEKEEP is a plot for entering and exiting graphics mode on the terminal. You would normally call entry point 0 with no parameters to enter REGIS and entry point 2 to exit REGIS.

7.2 LABEL

LABEL is a plot for doing a number of things, related to the mapping of the plot to the Regis screen.

It performs the following functions:

1. Draw a boundary box on the screen, label the axis
2. Scale the X axis and array
3. Scale the Y axis
4. Scale the log of the X axis and array
5. Scale the log of the Y axis
6. Scale the X axis and array (date type)
7. Compute the linear regression
8. Scale the Y array
9. Display Y axis scale lines
10. Display X axis scale lines
11. And more...

There are more utility plots that can be used to make your plotting experience easier.

EXAMPLES

The following is an example of a VT125 plot that displays an approximate frequency distribution of a field.

```
DTR> set dictionary vt125
DTR> show frequency
PLOT FREQUENCY
DECLARE         X_AXIS, Y_AXIS, X_LENGTH, Y_LENGTH
DECLARE         X_MAX, X_MIN, Y_MIN, Y_MAX
DECLARE         X_POS, Y_POS, K, TEMP
DECLARE         I, J
DECLARE VECTOR  XS, YS, XS1, YS1
DECLARE VECTOR  V
DECLARE         SUM, SUMSQ, MEAN, STD_DEV, MIN_VAL
DECLARE         MAX_VAL, RANGE, INTERVAL
DECLARE         INTERVALS
DECLARE STRING          LABEL
DECLARE         MEAN_POINT, POINTS, STRIDE, ORIGIN, SCALE
ENTRY 0 (X_LABEL : STRING)
    BEGIN
        PLOT HOUSEKEEP 0
        X_AXIS = 100            ! locate the axis'
        Y_AXIS = 360
        X_LENGTH = 600
        Y_LENGTH = 350
        LABEL = 'Frequency'
        PLOT LABEL 0 (X_AXIS, Y_AXIS, X_LENGTH,
                                  Y_LENGTH, X_LABEL, LABEL)
        INTERVALS = 50
        SUM = 0
        SUMSQ = 0
    END
ENTRY 1 (VAL)
    BEGIN
        SUM = SUM + VAL         ! for use later in statistics
        SUMSQ = SUMSQ + VAL*VAL
        V(SIZE(V)+1) = VAL

    END
! Print scatter plot
```

```
ENTRY 2
    BEGIN
        !       compute statistics
        MEAN = SUM/SIZE(V)
        STD_DEV = SQRT((SUMSQ - SIZE(V)*MEAN*MEAN)/(SIZE(V) - 1))
        MIN_VAL = MIN(V)
        MAX_VAL = MAX(V)
        !       calculate the range and interval
        RANGE = MAX_VAL - MIN_VAL
        INTERVAL = RANGE / INTERVALS
        K = MIN_VAL - (INTERVAL/2)       ! calculate outside of loop
        INCR I FROM 1 TO INTERVALS
        BEGIN
                XS1(I) = (I*INTERVAL) + K
                YS1(I) = 0
        END
        !       calculate frequency
        INCR I FROM 1 TO SIZE(V)
        BEGIN
                J = ((V(I)-MIN_VAL)/INTERVAL) + .5 ! inside interval j
                YS1(J) = YS1(J) + 1                ! update count
        END
        ! Push boarder case of MAX_VAL back into last interval
        YS1(INTERVALS) = YS1(INTERVALS) + YS1(INTERVALS+1)
        !
        ! calculate min/max for scaling
        !
        X_MIN = MIN (XS1)
        X_MAX = MAX (XS1)
        Y_MAX = MAX (YS1)
        Y_MIN = MIN (YS1)
        !
        ! strip off 0 frequencies inside of the plot
        ! this will spruce up the look of the plot
        !
        INCR I FROM 1 TO INTERVALS
                IF YS1(I) > 0 THEN
                BEGIN
                        XS(SIZE(XS)+1) = XS1(I)
                        YS(SIZE(YS)+1) = YS1(I)
                END
        !
        IF Y_MIN > 0
            THEN Y_MIN = 0
        !
        ! scale, drawing scales on the screen
        !
        PLOT LABEL 2 (X_MIN, X_MAX, XS)
        PLOT LABEL 3 (Y_MIN, Y_MAX)
        PLOT LABEL 8 (YS)
        !
        ! plot the frequency distribution
        !

        INCR I OVER XS
```

```
                PRINT CENTER (XS(I), YS(I)-9, '+', 9)
            !
            ! plot stats along X axis
            !
            PLOT LABEL 10 (X_MIN, X_MAX, POINTS, STRIDE, ORIGIN)
            SCALE = X_LENGTH/(POINTS*STRIDE)
            MEAN_POINT =   X_AXIS + SCALE * ( MEAN - ORIGIN )
            PRINT   CENTER ( MEAN_POINT  , Y_AXIS+10, '~M', 5)
            !
            ! and give statistics values (at bottom of the screen)
            !
            PRINT 'P', LXY( 20, 393),
                    'T(s1)', QUOTE('Sample Size = '),
                            'T(s1)', QUOTE( CVT(SIZE(V)) )
            PRINT 'P', LXY( 20, 415),
                    'T(s1)', QUOTE('MAX = '),
                            'T(s1)', QUOTE( CVT(MAX_VAL) ),
                    'T(s1)', QUOTE(',  MIN = '),
                            'T(s1)', QUOTE( CVT(MIN_VAL) )
            PRINT 'P', LXY( 20, 437),
                    'T(s1)', QUOTE('Mean = '),
                            'T(s1)', QUOTE( CVT(MEAN) ),
                    'T(s1)', QUOTE(',  STD DEV = '),
                            'T(s1)', QUOTE( CVT(STD_DEV) )
            !
            ! and cleanup before leaving
            !
            PLOT HOUSEKEEP 2
        END
    END_PLOT

    DTR> set dictionary -.demo
    DTR> ready yachts
    DTR> find yachts with price not equal 0
     [50 records found]
    DTR> plot frequency price of current
    DTR> plot connect
```

The following is an example of a VT100 plot. It produces a scatter-graph of the 2 input parameters. They use 2 utility plots for VT100's, one draws an axis (AXIS) and one scales the X and Y vectors.

The TERMINAL plot is included to show how support can be added for a new terminal type, simply rewrite the 3 entry points in TERMINAL, all plots using TERMINAL would then support the new terminal.

We currently maintain 3 graphics dictionaries, VT125, VT100, and VT52. VT125 has all of the DEC supplied plots plus a few of our own. VT100 and VT52 both have identical plots, a small subset of VT125.

```
DTR> set dictionary vt100
DTR> show x_y
PLOT X_Y
!
!               x-y     --      plot each x-y point
!
                DECLARE VECTOR          X,Y
                DECLARE                 I, PX, PY, L, C
                DECLARE STRING          PLOT_CHAR, STR
!
        ENTRY 0 (X_LABEL : STRING, Y_LABEL : STRING)
        BEGIN
                PLOT TERMINAL 0                 ! clear the screen
                PLOT AXIS 0 (X_LABEL, Y_LABEL)  ! plot axis
                PLOT_CHAR = '*'
                L = 1
                C = 1
                STR = 'X_Y PLOT'
                PLOT TERMINAL 1 ( L, C, STR)
        END
        ENTRY 1 ( X_ELEMENT, Y_ELEMENT )
        BEGIN
                X(SIZE(X)+1) = X_ELEMENT        ! store elements as
                Y(SIZE(Y)+1) = Y_ELEMENT        ! they arrive
        END
        ENTRY 2
        BEGIN
                PLOT SCALE 0 (X)                ! scale x axis
                PLOT SCALE 1 (Y)                ! scale y axis
                INCR I OVER X
                BEGIN
                        PX = X(I)
                        PY = Y(I)
                                ! plot the point
                        PLOT TERMINAL 1 (PY,PX,PLOT_CHAR)
                END
                PLOT TERMINAL 2                 ! exit
        END
END_PLOT


DTR> show terminal
PLOT TERMINAL
!
!               TERMINAL --     will clear the screen
!                               and do cursor
!                               addressing for vt100
!
        ENTRY 0         ! clear screen
        BEGIN
                PRINT '<ESC>[2J'
        END
!
        ENTRY 1 (L, C, STR : STRING) ! PLACE STRING AT ADDRESS
        BEGIN
```

```
                L = INT( L )
                C = INT( C )
                PRINT '<ESC>[',CVT(L),';',CVT(C),'H',STR
        END
!
        ENTRY 2          ! exit - cursor to bottom
        BEGIN
                PRINT '<ESC>[22;1H'
        END
!
END_PLOT

DTR> set dictionary -.demo
DTR> ready yachts
DTR> find yachts with beam not equal Ø and price not equal Ø
[47 records found]
```

Time In Datatrieve

Dick Azzi
Motorola
Phoenix, Arizona

I recently came upon a situation where I had to enter the time into a date field in Datatrieve and found that there was no way to do it. I wrote a short procedure that may be of use to others that want to do the same thing.

You will have to declare 4 variables in the calling procedure and have those variables loaded with data when calling the procedure to place the time in your date field. These variables are as follows:

```
DECLARE FOODATE USAGE DATE. ! Date to be in field.
DECLARE FOOHR PIC 99.       ! Hours of time to be in field.
DECLARE FOOMIN PIC 99.      ! Minutes of time to be in field.
DECLARE FOOSEC PIC 99.      ! Seconds of time to be in field.
```

Once you load these variables call the following procedure and the time and date will be returned in the variable FOODATE.

```
DEFINE PROCEDURE FOOTIME
DECLARE SEC_FOO COMPUTED BY 10000000.
DECLARE MIN_FOO COMPUTED BY SEC_FOO * 60.
DECLARE HR_FOO COMPUTED BY SEC_FOO * 3600.
DECLARE DATE_FOO PIC 9(18)
USAGE IS COMP.
DATE_FOO=FOODATE
DATE_FOO=DATE_FOO + FOOHR*HR_FOO + FOOMIN*MIN_FOO + FOOSEC*SEC_FOO
FOODATE=DATE_FOO
END_PROCEDURE
```

Four-Column Mailing Labels in Datatrieve

Timothy T. Toyne
600 South St. - Rm 1017
New Orleans, La 70130
(504)589-2972

After reading the March 83 issue of the WOMBAT EXAMINER, (WOMBAT MAGIC - a two column list), I became interested in the possibility of printing four-column mailing labels. Because of the number of mailings that we must complete each month, and the size of the file (about 2000 records), four-column labels would be a definite time saver in our operation.

I made an effort to use the procedure for a two-column list, but I was not able to get it to work with multiple line address requirements. Besides, I wasn't too keen on doing this operation in the report writer. (Not that I have anything against the report writer.)

I tried several ideas - but none worked. So, I went back to my standard methodology when I have problems developing something in datatrieve: If I were going to do this operation in COBOL (sorry, I know you can't compare datatrieve to a high level programming language), how would I set up the storage area?

The procedure speaks for itself, once I declared variables for all fields in each address group for each column, the battle was over. By incrementing a counter and calling a print procedure after the fourth store in the inner nest, we have four column labels. Be sure that you include three dummy records at the end of the file to insure that all records print.

Example:

```
DELETE FAST_LABELS;
DEFINE PROCEDURE FAST_LABELS
!
!  THIS PROCEDURE PRODUCES FOUR COLUMN MAILING LABELS
!
!  DATE:  5 DECEMBER 1983
!
!  WRITTEN BY:  CWO T. T. TOYNE
!
!  COMMANDER EIGHTH COAST GUARD DISTRICT (b)
!
!  NEW ORLEANS, LA
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
!   THIS SECTION DECLARES NECESSARY COLUMN VARIABLES
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
!  YOU MUST BE SURE TO SET THE TERMINAL WIDTH TO 132
!  AND IN DTR -- THE COLUMNS-PAGE TO 132
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
!                        I M P O R T A N T
!
!  PUT THREE DUMMY RECORDS IN THE FILE TO BE SURE THAT
!  ALL RECORDS PRINT
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
SET ABORT
READY MBR
DECLARE TFNAME1          PIC IS X(15).
DECLARE TFNAME2          PIC IS X(15).
DECLARE TFNAME3          PIC IS X(15).
DECLARE TFNAME4          PIC IS X(15).
DECLARE TLNAME1          PIC IS X(15).
DECLARE TLNAME2          PIC IS X(15).
DECLARE TLNAME3          PIC IS X(15).
DECLARE TLNAME4          PIC IS X(15).
DECLARE TSTRT1           PIC IS X(20).
DECLARE TSTRT2           PIC IS X(20).
DECLARE TSTRT3           PIC IS X(20).
DECLARE TSTRT4           PIC IS X(20).
DECLARE TCITY1           PIC IS X(15).
DECLARE TCITY2           PIC IS X(15).
DECLARE TCITY3           PIC IS X(15).
DECLARE TCITY4           PIC IS X(15).
DECLARE TST_ZIP1         PIC IS X(8).
DECLARE TST_ZIP2         PIC IS X(8).
DECLARE TST_ZIP3         PIC IS X(8).
DECLARE TST_ZIP4         PIC IS X(8).
DECLARE COUNTER          PIC 9.
COUNTER = 0
```

```
FOR MBR
  BEGIN
    COUNTER = COUNTER + 1
    IF COUNTER = 1 THEN
      BEGIN
        TFNAME1 = FNAME
        TLNAME1 = LNAME
        TSTRT1 = STRT
        TCITY1 = CITY
        TST_ZIP1 = ST_ZIP
      END ELSE
    IF COUNTER = 2 THEN
      BEGIN
        TFNAME2 = FNAME
        TLNAME2 = LNAME
        TSTRT2 = STRT
        TCITY2 = CITY
        TST_ZIP2 = ST_ZIP
      END ELSE
    IF COUNTER = 3 THEN
      BEGIN
        TFNAME3 = FNAME
        TLNAME3 = LNAME
        TSTRT3 = STRT
        TCITY3 = CITY
        TST_ZIP3 = ST_ZIP
      END ELSE
    IF COUNTER = 4 THEN
      BEGIN
        TFNAME4 = FNAME
        TLNAME4 = LNAME
        TSTRT4 = STRT
        TCITY4 = CITY
        TST_ZIP4 = ST_ZIP
        :FAST_LABELS2
        COUNTER = 0
      END
  END
END-PROCEDURE


DELETE FAST_LABELS2;
DEFINE PROCEDURE FAST-LABELS2
!
!     THIS SECTION PRINTS THE FOUR COLUMN LABELS
!
PRINT COL 3, TMEM-NO1(-), COL 37, TMEM-NO2(-), COL 72, TMEM-NO3(-),
COL 105, TMEM-NO4(-), COL 3, TFIR-NAM1|||TSIG-NAME1(-), COL 37,
TFIR-NAM2|||TSIG-NAME2(-), COL 72, TFIR-NAM3|||TSIG-NAME3(-), COL 105,
TFIR-NAM4|||TSIG-NAME4(-), COL 3, TSTREET1(-), COL 37, TSTREET2(-),
COL 72, TSTREET3(-), COL 105, TSTREET4(-), COL 3, TCITY1|||TSTATE1|||TZIP1(-),
COL 37, TCITY2|||TSTATE2|||TZIP2(-), COL 72, TCITY3|||TSTATE3|||TZIP3(-),
COL 105, TCITY4|||TSTATE4|||TZIP4(-), SKIP 2
END-PROCEDURE
```

DISTRIBUTED DATATRIEVE ACCESS FACILITY (DDAF)

Terry Cassidy
Digital Equipment Corporation

Overview

```
MYVAX:: OURVAX:: connection
  OURVAX:: DOMAIN YACHTS USING YACHT ON YACHT.DAT
MYVAX:: READY YACHTS AT OURVAX
MYVAX:: PRINT YACHTS WITH LOA EQ 20

MYVAX:: OURVAX:: connection
with VAX-11 CDD, VAX-11 Files, request from MYVAX::
DTR32.EXE talking to DDMF.EXE
READY YACHTS AT OURVAX
PRINT YACHTS WITH LOA EQ 20

MYVAX:: OUR11:: Connection
        OUR11:: DOMAIN YACHTS USING YACHT ON YACHT.DAT
MYVAX:: READY YACHTS AT OUR11
MYVAX:: PRINT YACHTS WITH LOA EQ 20

MYVAX:: OUR11:: Connection
with DTR-11 Data Dictionary, DTR-11 Files, request from MYVAX::
DTR32.EXE talking to DDMF.TSK
READY YACHTS AT OUR11
PRINT YACHTS WITH LOA EQ 20
DISTRIBUTED ACCESS
```

Efficient access to remote data

Data lives on other VAXes
Data lives on PDP-11s
Define a domain that points to remote
READY a domain AT remote
Query decomposition
Maximizes parallel processing
Minimizes network traffic

Distributed Access Components

HOSTS

VAX
- DTR32xx.EXE, the terminal server
- Applications programs using VAX-11 DATATRIEVE Call Interface

PDP-11
- REMDTR.TSK, allows access to a single remote server
- Application programs using the PDP-11 DATATRIEVE Call Interface

SERVERS

VAX
- DDMFxx.EXE

PDP-11
- DDMF.TSK

A sample session

HOST (MYVAX)	SERVER (OURVAX)
DATATRIEVE user types command READY YACHTS AT OURVAX	
DATATRIEVE asks DECnet to establish a link to OURVAX with known object 30 (DDMF)	
	DECnet responds and starts DDMF
	DDMF sends an initialization packet
DTR looks at the initialization packet to ensure that the object on the other end is a DDMF and that it is not trying to use an obsolete protocol	

DTR sends an initialization
packet of its own

DDMF looks at the initialization
packet to ensure that the
object on the other end is a
legitimate host and that it
is not trying to use an
obsolete protocol

DTR sends a command packet like
READY YACHTS AS DTR$D0 READ;
DISPLAY "READY";

DDMF processes the READY command
DDMF processes the DISPLAY
statement and sends a display
packet

DTR receives the display packet

DTR sends packets requesting
information about the READYed
domain (fields, etc)

DDMF responds with information
packets

DTR prompts the user for more
input
DATATRIEVE user types command
PRINT YACHTS WITH LOA = 30

DTR sends this command packet

```
BEGIN
BEGIN
DECLARE PORT DTR$0 DTR$GROUP STRUCTURE.
DTR$1 DATATYPE IS TEXT SIZE IS 10 CHARACTERS.
DTR$2 DATATYPE IS TEXT SIZE IS 10 CHARACTERS.
DTR$3 DATATYPE IS TEXT SIZE IS 6 CHARACTERS.
DTR$4 DATATYPE IS TEXT SIZE IS 3 CHARACTERS.
DTR$5 DATATYPE IS UNSIGNED NUMERIC SIZE IS 5 DIGITS.
DTR$6 DATATYPE IS UNSIGNED NUMERIC SIZE IS 2 DIGITS.
DTR$7 DATATYPE IS UNSIGNED NUMERIC SIZE IS 5 DIGITS.
END DTR$GROUP STRUCTURE.
END.;
FOR DTR$D0 WITH (DTR$D0.LENGTH_OVER_ALL EQ 30)
BEGIN
STORE DTR$0 USING
BEGIN
DTR$1 = DTR$D0.MANUFACTURER;
DTR$2 = DTR$D0.MODEL;
```

```
DTR$3 = DTR$D0.RIG;
DTR$4 = DTR$D0.LENGTH_OVER_ALL;
DTR$5 = DTR$D0.DISPLACEMENT;
DTR$6 = DTR$D0.BEAM;
DTR$7 = DTR$D0.PRICE;
END;
END;
DISPLAY "DTR$0";
END;
END;
```

DTR continues processing the
PRINT statement and prints the
column headers

```
MANUFACTURER   MODEL     RIG ...
```

DDMF receives the command packet
DDMF processes the PORT declaration
DDMF processes the FOR statement
DDMF processes the STORE statement
by sending data packets from the
port as they are stored

DTR receives the data packets
and prints lines

```
ALBIN          BALLAD     SLOOP ...
CAL            3-30       SLOOP ...
FISHER         30         KETCH ...
```

DDMF processes the DISPLAY
statement and sends a display
packet

DTR receives the display packet
DTR prompts the user for more
input

DATATRIEVE user types command
FINISH YACHTS

DTR sends a command packet like

```
FINISH DTR$D0;
DISPLAY "FINISH";
```

DDMF processes the FINISH command
DDMF processes the DISPLAY
statement and sends a display
packet

DTR receives the display packet

DTR sends a shutdown packet

DDMF receives the shutdown packet
and has DECnet drop the link

DTR detects the dropped link
and considers the session
terminated
DTR prompts the user for more
input

Tracking Problems

The DDMF Server

The log file containing commands, statements, and error messages

SYS$LOGIN:NETSERVER.LOG (DECnet-VAX Phase IV)
SYS$LOGIN:DDMFxx.LOG (DECnet-VAX Phase III)
DDMF.LOG (in login directory) (PDP-11s)

Example (VAX)

```
$ SET NOVERIFY

Connect request received at 21-OCT-1983 09:43:41.31
from remote process MYVAX::"0=000D0047"
for object "SYS$SYSROOT:[SYSEXE]DDMF.COM;"

VAX-11 DATATRIEVE Remote Server
Protocol 3.1
READY YACHTS AS DTR$D0   READ; DISPLAY "READY";
Statement completed successfully.
Statement completed successfully.
 BEGIN BEGIN DECLARE PORT DTR$0 DTR$GROUP STRUCTURE.
DTR$1 DATATYPE IS TEXT SIZE IS 10 CHARACTERS .
DTR$2 DATATYPE IS TEXT SIZE IS 10 CHARACTERS .
 .
 .
 .
FINISH DTR$D0; DISPLAY "FINISH";
Statement completed successfully.
Statement completed successfully.
VAX-11 DATATRIEVE Remote Server terminating with status 00000001
  CASSIDY      job terminated at 21-OCT-1983 09:44:07.99
```

The VAX Host

```
$      RUN/DEBUG SYS$SYSTEM:DTR32
Some or all global symbols not accessible
        .
        .
        DBG> GO
        VAX-11 Datatrieve V2.0
        .
        .
        DTR> READY YACHTS AT OURVAX
        REMCMD> READY YACHTS AS DTR$D0   READ; DISPLAY "READY";
        DTR> PRINT YACHTS WITH LOA = 30
        REMCMD>  BEGIN BEGIN DECLARE PORT DTR$0 DTR$GROUP STRUCTURE.
        REMCMD> DTR$1 DATATYPE IS TEXT SIZE IS 10 CHARACTERS .
        REMCMD> DTR$2 DATATYPE IS TEXT SIZE IS 10 CHARACTERS .
        .
        .
        .

                                  LENGTH
                                   OVER
        MANUFACTURER   MODEL    RIG    ALL   WEIGHT BEAM  PRICE

         ALBIN      BALLAD     SLOOP   30     7,276  10  $27,500
         CAL       3-30      SLOOP   30    10,500  10
         FISHER     30        KETCH   30    14,500  09
         .
         .

        DTR> FINISH YACHTS
        REMCMD> FINISH DTR$D0; DISPLAY "FINISH";
```

Security

SERVER

- use dictionary protections
- use user name and password in the access string

```
DTR> READY YACHTS AT OURVAX"USER LETMEIN"
DTR> DEFINE DOMAIN YACHTS USING CDD$TOP.MYLIB.YACHTS AT
     OURVAX"USER LETMEIN";
```

HOST

- Password can be given at READY time (VAX only)

```
DTR> DEFINE DOMAIN YACHTS USING CDD$TOP.MYLIB.YACHTS AT
```

```
     OURVAX"USER *.'Password'"
DTR> READY YACHTS
Enter Password:                         (not echoed)
```

Restrictions

```
- FOR REMOTE_YACHTS WITH BUILDER CONT *.BUILDER ...
- FIND REMOTE_YACHTS
  SELECT FIRST WITH PRICE = 0
- Validation errors cause ABORTs
- MODIFY and STORE of lists
- FOR REMOTE_YACHTS WITH PRICE > 3*DISP
- FOR REMOTE_YACHTS CROSS REMOTE_FAMILIES
  - works on VAX if both domains use the same VAX server
- REDUCE (PDP-11 Server)
- FROM (PDP-11 Server)
- STARTING WITH (PDP-11 Server)
```

VAX-11 DATATRIEVE TECHNICAL TUTORIAL

Terry Cassidy
Digital Equipment Corporation

What is DATATRIEVE anyway

DEC Query and Report System
High Level Application Language
Callable Data Manager

Competing Goals

Performance
Ease of use
Flexibility
Productivity

DATATRIEVE APPLICATIONS

RMS Key Optimazation

Exact over range retrieval
Unique over duplicates
Primary over alternate
Collections are not keyed

Checking for Key Optimization

```
$      RUN/DEBUG SYS$SYSTEM:DTR32
Some or all global symbols not accessible
 .
 .
DBG> GO
VAX-11 Datatrieve V2.0
 .
 .
PRINT YACHTS WITH BUILDER = "PEARSON"
Performing EQL boolean on RMS key field YACHTS.MANUFACTURER
 .
```

DBMS Key Optimization

Equality Booleans
EQ boolean AND
May or may not be faster

Hierarchies vs Relations

If you can, use relations
Views can make it hierarchic
Easier to manage
Join using indexed fields

FIND vs FOR

FOR is faster
FOR is more flexible

Dictionary Tables

Stored in CDD
Faster than Domain Tables
Harder to update

```
DEFINE TABLE RIG_TABLE
"SLOOP" : "ONE MAST",
"KETCH" : "TWO MASTS, BIG ONE IN FRONT",
"YAWL"  : "SIMILAR TO KETCH",
"M/S"   : "SAILS AND BIG MOTOR",
ELSE    "SOMETHING ELSE"
END_TABLE
```

Domain Tables

Stored in a file
IN is ANY
VIA is FROM

```
DEFINE TABLE BUILDER_PRICE FROM YACHTS USING
BUILDER : PRICE
ELSE    0
END_TABLE
```

Equivalences

IN/ANY

IF "PEARSON" IN BUILDER_PRICE THEN ...
IF ANY YACHTS WITH BUILDER = "PEARSON" THEN ...

VIA/FROM

PRINT "PEARSON" VIA BUILDER_PRICE
PRINT PRICE FROM YACHTS WITH BUILDER = "PEARSON"

RMS Locking

READY modes control locking
SELECTED records
Records subject to change
re-READY may force re-fetch

Readers read through locks

Writers wait for locks

Re-try for 15 secs
If LOCK-WAIT is set, then tell RMS to wait

Deadlock

May happen if:
you allow shared update
you SET LOCK-WAIT
Aborts the statement
May clear the SELECTed record

DBMS Locking

Record Level Locking
True concurrent update
Read locks on collections

Join Optimization

Use Keys
Keyed Domains last
Domain with fewest records first

User Functions

If you can't do it in DTR...
Must be position independent
Use stack or VM for storage

Plot Language Restriction

Why don't we publish the graphics language?
Not the right product
Not designed for end-user

Call Interface

Anticipated Uses

Application programs (DTR as access method)
Canned packages (Mimic terminal interface)
Intelligent data manager (Generate requests on the fly)
Remote DATATRIEVE server

What is it good for?

Productivity
Data Independence
Database Independence
Efficiency
Distributed performance

Stall Points

Basic concept of the interface
DTR runs until it stalls
Stall tells you what to do
Need a command
Need a prompt
Have a print line
Have a message / condition
Have a record for the host program
Need a record from the host program
Service a user keyword

Ports

Communication mechanism
An in-memory record
DECLARE or DEFINE and READY
Act like domains in DTR language

STORE of port causes "port get"
Access to port causes "port put"

Programming Hints

Use REPEAT of FOR <port> loops
Use prompts
Synchronize with DISPLAY
Make ports specific to task

```
DECLARE PORT FOO
 01 GROUP.
    03 VALUE1 REAL.
    03 VALUE2 REAL.
;
FOR YACHTS STORE FOO USING
  BEGIN
  VALUE1 = BEAM;
  VALUE2 = LOA;
  END;
DISPLAY "FOO";
```

Very flexible interface
Application may not need it at all
Build a layer for what is needed
Use the DTR$DTR call

The DTR$DTR call handles

Selected stallpoints
Messages and conditions
DTR defined keywords
DTR conditions

Efficiency

Initialize at the start
Shutdown at the end
Never put DTR$COMMAND in a loop
The more work per DTR$COMMAND the better

What is happening

- Do not use $SET VERIFY
  - slows down the installation
  - wastes a lot of paper
- Use $@SYS$UPDATE:VMSINSTAL DTR020 MTA0: OPTIONS K,L
  - K switch shows what steps are taken
  - L switch logs updates

What is Installed

- the shareable image (SYS$SHARE:DTRSHRxx.EXE) used by the terminal server, the remote server, and application programs using the call interface
- the terminal server (SYS$SYSTEM:DTR32xx.EXE)
- the remote server (SYS$SYSTEM:DDMFxx.EXE) and its start up file (SYS$SYSTEM:DDMFxx.COM)
- the help file (SYS$HELP:DTRHELP.HLB)
- the message file (SYS$MESSAGE:DTRMSGS.EXE)
- ADT and Guide Mode Data Files (SYS$LIBRARY:DTRADT.DAT and SYS$LIBRARY:DTRGUIDE.DAT)
- the start up file (SYS$MANAGER:DTRSTUPxx.COM)
- the DATATRIEVE library (DTR$LIBRARY:) containing UETPs/Demos, object libraries, build commands, other optional material

Query and Reporting Facility on the Professional 350

Digital Equipment Corporation

What is PRO/DATATRIEVE?

- Digital's new query and reporting facility on the PROFESSIONAL 350.
- Based on DATATRIEVE-11 V3.

What can you do with PRO/DATATRIEVE?

- Describe data
- Manage data
- Query data
- Report data

Ways to describe data

- ADT (Application Design Tool)
- "DEFINE" commands
- Existing DATATRIEVE data definitions

Managing data

- Store new data
- Update existing data
- Erase existing data

Managing data made easy

- Guide Mode
- On-line help
- PRO/DATATRIEVE FOR BEGINNERS

Querying data

- Create groups of data based on
  - boolean expressions
  - subset of records
  - single file
  - several files (VIEW)
- Sort groups of data

Reporting data

- Create ad-hoc reports
- Print detail lines
- Print summary lines
- Print statistical information
- Control output format

Layered Application Facility
(LAF)

- Create PRO/DATATRIEVE applications
- Store applications in menu
- Choose applications through menus
- Allow applications to have unique data dictionary elements

Software Requirements

- P/OS V1.5 or later
- LAF requires PRO/DATATRIEVE

Hardware Requirements

- Professional 350 with hard disk

Fall DECUS 1983
Magic Session
Las Vegas, Nevada

Bert Roseberry
United States Coast Guard,
New Orleans, Louisiana.

This isn't magic in the traditional sense, but I felt the ordering numbers for version 2.0 for DATATRIEVE was pure magic!

AA-P860A-TE	Summary Description
AA-P862A-TE	VAX-11 DTR Guide to Writing Reports
AA-P863A-TE	VAX-11 DTR Guide Program Customization
AA-P864A-TE	VAX-11 DTR Pocket Guide
AA-K079B-TE	VAX-11 DTR Reference Manual
AA-K080B-TE	VAX-11 DTR User's Guide
AA-L631B-TE	VAX-11 DTR Graphics Guide

Andy Schneider - Developer of DATATRIEVE 11 - Digital

How to clear out a box of line printer paper in three commands or less:

```
READY YACHTS
REPORT YACHTS ON LP:
AT TOP OF PAGE PRINT NEW-PAGE
PRINT LOA
END-REPORT
```

I sit long hours trying new things with DATATRIEVE all the time, and I want an answer to this question on what this will do to DATATRIEVE 11. Nobody knows? (general Laughter)

This is a feature only of DATATRIEVE 11. VAX DATATRIEVE gives you an error saying you can't have new page at the top of page. We're not smart enough to say that.

Phil Necker - Montgomery Engineers

Not to belittle all these people that are trying to discredit this fine product - I have Wombat Magic on how to display a report one screen at a time. This is real magic. It actually works in DATATRIEVE 11 which surprises the heck out of me. No thats not true. It does work in DATATRIEVE 11 and it is not surprising.

I don't know how many of you have done reports where you can specify whether you want it on the screen or on the file and you've said, "Hey I've made this thing really user friendly, even my manager can use it."

The only trouble is he doesn't know where the NO SCROLL key is on the VT 100, right? So you put a little prompt there, you know: "Enter the place you want to put TT; if you want it on the terminal." He merrily puts in "TT" and the thing prints out a page which is usually the wrong number of lines because you forgot to put SET LINES = 24 or whatever. And then a form feed comes up and the thing merrily scrolls across the screen and it does that for the whole report. And he says, "That was a stupid report."

So, the fix is actually pretty cute. It turns out that DATATRIEVE is very very smart about prompting, and it'll even let you do things like this:

```
report yachts sorted by loa, builder on tt:
set report-name = "Easy to Read"/"Terminal Report"
set lines-page = 20
print builder, rig, load, price
at bottom of page print skip 1,
"  "|*."any character and <RETURN> to continue" using x
end-report
```

The "At bottom of page, PRINT SKIP 1, blanks concatenated with a Prompt, using X" will print the blank. But of course it requires the prompt every page. So very merrily DATATRIEVE goes along, prints out the whole page, skips a page, now you're at about line 24 on your terminal. It waits for you to type something - space, A, Foo, it doesn't matter what it is because no matter what it is, it's only going to print the space because its using the X. The guy hits return, and it goes up and prints up the next form. That's magic.

Pam Valentine - Montgomery Engineers

I have a little magic here I think. If you read the DATATRIEVE documentation it will tell you that you can not use signed numeric fields with FMS. And I say "yes you can."

First you make your form in FMS, and you put N's in the fields, so that restricts your input to your user, so that he is restricted to only putting a sign and numbers and one decimal. Now the magic part. For your record definition, you give it a picture of X, however many, VXX, and DATATRIEVE magically does the right thing when it passes the records back.

```
  .
  .
  .
03 Amount Pic x(7)vxx
   query-header is "amount"
   edit-string is zz,zzz.99CR
  .
  .
  .
```

Even if the user puts plusses in some of his records and not in others, when it does the arithmetic it just does the right thing. I think that's magic.

Diane Harris - Computer Services

I don't have any foils because this goes in the category of War Story, therefore, a question. The first thing is we had a problem. We wanted to be able to use EDT from a FORTRAN program, and we didn't know how to do this except through SPAWN which everybody knows is drastically slow. But we have this programmer who is very good at finding new ways of doing things and he doesn't read the manuals. And the manual for DATATRIEVE Version 1 says, "Thou shalt not try to get to EDT through the Cobol interface."

Well, luckily he doesn't read such things and he was able to call EDT through the DATATRIEVE call interface. Well, everything was well and good and it was much faster than SPAWN. However, Version 2 comes out. The manual states You can now use EDT from the callable DATATRIEVE interface. Well, this is wonderful. But of course the first time my program runs it says this is out of date so we have to compile it and we have to link it with the appropriate DTR share and library/include EDT. Doesn't work. OK, this is fine. Read further in the manual, and it says you can use the terminal server. Our problem now, is that we want to keep the user trapped in the program. We don't want the user to be able to enter any DATATRIEVE commands. We want to put him in the editor to edit text. How can we do it?

Phil Naecker - Montgomery Engineers

It's easy to do SPAWN not being slow. For example, this is the way SPAWN works if you do it right: You do a SPAWN inside DATATRIEVE.

```
DTR> spawn
(5 seconds elapsed; 2 seconds CPU)
$ directory
  .
  .
  .
$ pop
DTR> SPAWN
(0.5 seconds elapsed; 0.1 seconds CPU)
$
```

Obviously there is this new user defined function called SPAWN. We leave out the FN$ - in our case we don't like FN$SPAWN. We want people to think this is DATATRIEVE doing this, not us monkeying around with it. And uh, by the way this is DATATRIEVE 32 - sorry 11 users. It takes about 5 seconds elapsed time on our system normally, and about 2 seconds of CPU to spawn a sub process because you're doing this terribly slow process create. As Gail said it takes a long time. So you do a SPAWN and you have a directory and when you're all done with whatever you're doing here - we have a command that we use - it's POP - some of you unix users might recognize that - and then next time you do a SPAWN lo and behold, it only takes about a half second elapsed - less than a tenth of a second CPU time.

It's a fairly trivial FORTRAN program to do that. Now you can do SPAWNS like this - the trouble is of course, in between the $ signs the users responsible for his own input which is probably a restriction that you failed to mention but - ok I'm sorry - well that's easy too because it turns out that when you do a SPAWN you can also request that SPAWN execute a command.

There is an argument in the LIB$SPAWN call so you can execute a command procedure over here which he has no control over which itself does the POP to return to the SPAWNing process. The FORTRAN looks roughly like this. This is stripped down a little bit, but if you put the declarations and everything in they'll see and it doesn't take much longer than that.

```
SUBROUTINE SPAWN_DCL_PROCESS
  .
  .
  .
DECLARATIONS
  .
  .
  .
IF (SUBPROCESS_PID .EQ. 0) THEN
       STATUS = SYS$GETJPI(0,JPIREQ,PID)
       WRITE (PID_TEXT,'RB')PID
       CALL LIB$SPAWN ("  ','POP:==
              ATTACH/ID='/(PID_TEXT,SUBPROCESS_PID)
ELSE
       CALL LIB$ATTACH(SUBPROCESS_PID)
ENDIF
  .
  .
  .
ERROR DETECTION
  .
  .
  .
RETURN
END
```

Your user define function which you define as it says in the manual for defining a user define function. You check first to see if you've done this before. If you have created a sub process you'll have something in sub process PID which (by the way in FORTRAN you have to do something tricky) you have to do a save operation to keep the value of that variable from instance to instance of the invocation in FORTRAN. Basically you find out what your existing process is by requesting something from GETJPI. You convert that into text then you use that in the SPAWN command to say from now on use POP as being attached/ID=whatever. And then you retrieve you subprocess PID Process Idea subprocess. Next time you call it you're going to take this ELSE clause and you're simply going to attach it to the subprocess. Well, the point is that you can do SPAWNS efficiently. You can do a SPAWN - the first time it cost you a PROCESS CREATE which you can't get around in any way in VMS, but from then on out it works as efficiently as any context which does in VMS.

Dick Azzi - Motorola

We did the same type thing with uh, I don't know who did this, but somebody did it in BASIC.

```
BASIC PROGRAM FOR SPAWN FUNCTION

10      SUB DTR_SPAWN(INPUT_STRING$)
        EXTERNAL LONG FUNCTION LIB$SPAWAN
20      STS% = LIB$SPAWN( INPUT_STRING$,,,,,,,,,)
        PRINT 'THE STATUS WAS: ';STS% IF STS%<>1%
32767 SUBEND
```

But a little idea what it does:

```
FORMAT FOR SPAWN FUNCTION

DTR> PRINT FN$SPAWN("COMMAND") ON NL:

EXAMPLES:

PRINT FN$SPAWN ("") ON NL: - puts you in DCL; exit with logoff
PRINT FN$SPAWN ("PHONE") ON NL: - puts you in phone utility
PRINT FN$SPAWN ("MAIL" ON NL: - puts you in mail utility
PRINT FN$SPAWN ("SET TERM/MODIFIER") ON NL: - sets terminal
PRINT FN$SPAWN ("MONITOR PROCESS") ON NL: - puts you in monitor
PRINT FN$SPAWN ("SHOW MODIFIER") ON NL: - allows show commands
```

In DATATRIEVE you can do a Print FN$SPAWN and you got a whole bunch of choices - either give it the quotes inside inside the parentheses and that puts you in DCL and do whatever you want to do in DCL and exit with a logoff and you're back in DATATRIEVE - or - you can put anything you want between the quotes and it'll execute that function and if you're in the middle of DATATRIEVE and somebody is phoning you - right? You have got to get out of what you're doing, get back to the system manager, phone, and all that junk. The next command, just do a PRINT FN$SPAWN PHONE ON NL: it puts you in the phone utility - you hang up on you phone after you talk to him, and you're back in DATATRIEVE. I do that with anything, set terminals, monitor process, show stuff. What we've done is set up some of these like phone, mail and all that, as procedures - call it Procedure Phone - and then put those in CDD$TOP - create a global symbol called DTR_PHONE and then no matter what directory you're in, just do a :DTR_PHONE and it links up and you're in the phone utility and when you hang up you're back in DATATRIEVE.

Larry Jasmann - U.S. Coast Guard

This is some serious magic. If you have a file that has a non duplicating primary key which means that there is only one instance of any particular iteration of the primary key - in other words like social security numbers and we've all got a separate one - that's what I'm talking about. Here is a neat way that you can do stores, updates, and deletes all with the same procedure and one tight little procedure. You don't have to have a separate procedure to store, a separate procedure to modify, and a separate procedure to delete which was the way I did things for about 4 or 5 years until Peggy Racel came along and this is at least half her magic too.

OK, here's a little Domain/record:

```
RECORD SAMPLE-REC USING
01 SAMPLE-REC.
      03 KEYS.
         05 PRI-KEY        PIC X(5)
      03 DATA.
         05 STAMP-DATE USAGE IS DATE
         05 REMARKS PIC X(20).
;
```

the only thing I really care about this is the primary key is PIC X5 - the data in the second 03 level is just a couple of data fields - can be whatever you want to make it.

The first thing we do is declare three variables:

```
DECLARE DEL  PIC X.
DECLARE CURRENT-COUNT PIC 99.
DECLARE CUR-REC PIC X(5).
```

DEL is just PIC X, current count is PIC 99, CUR-REC has to be declared the same as the primary key so if the primary key is PIC X(5) then CUR-REC has to be PIC X(5). If it's something else, then you have to make that equal whatever that is. OK - we print a little print statement that tells the user what we're doing:

```
PRINT "This procedure is used to store, modify, and delete records from",
col 1,
"the domain SAMPLE.  It can be adapted to any domain which has a ,",
col 1,
"non-duplicating primary key."
```

and then we use a prompting statement to load the first primary key value of the first record you want to work on into the variable CUR_REC:

```
CUR-REC = *.NOT CONT "END"
```

and then we start a big while loop which says:

```
WHILE CUR-REC NOT CONT "END
   BEGIN
       DEL = *."D to delete or N to keep record."
       CURRENT-COUNT = COUNT OF SAMPLE WITH PRI-KEY = CUR-REC
       IF CURRENT-COUNT LT 1
              BEGIN
                     PRINT "I assume that you want to add a new record!"
                     STORE SAMPLE USING
                            BEGIN
                                   PRI-KEY = CUR-REC
                                   STAMP-DATE = *."Date"
                                   REMARKS = *."Remarks"
                            END
              END
       IF CURRENT-COUNT = 1 AND DEL NE "D" AND DEL NE "d"
              BEGIN
                     PRINT "I assume that you want to modify this record!"
                     PRINT ALL SAMPLE WITH PRI-KEY = CUR-REC
                     FOR SAMPLE WITH PRI-KEY = CUR-REC MODIFY USING
                            BEGIN
                                   STAMP-DATE = *."Date"
                                   REMARKS = *."Remarks"
                            END
              END
       IF CURRENT-COUNT = 1 AND (DEL = "D" OR DEL = "d")
              BEGIN
                     PRINT ALL SAMPLE WITH PRI-KEY = CUR-REC
                     IF *."Y if you wish to delete this record" CONT
                            "Y" THEN
                     FOR SAMPLE WITH PRI-KEY = CUR-REC ERASE
              END
       CUR-REC = *."Next record to update or END"
   END
```

The next thing we do is to prompt for DEL so we get a value for DEL which is used to decide whether or not what we want to do is delete the record or not.

Then you get a count of domain with the PRI-KEY = CUR-REC. Now because you have a non duplicating primary key there are only two options. You've either got 0 records in there or you've got 1 record. So CURRENT-COUNT can only either be 0 or 1. So now we take care of the possible combinations. If CURRENT-COUNT is less than one, i.e. 0 then obviously we haven't got a record in there so what we assume is that you want to add a new record. It tells you that and then it just does a simple store.

The next thing to do (if CURRENT-COUNT = 1 then we know you've got a record in there) is either to modify it or delete it. So if CURRENT-COUNT

= 1, DEL is not = to D, then we assume that you want to modify the record - you print out the record so that they can see what it is they are working on, and then you "MODIFY USING BEGIN" and prompt for all the fields in the record except for the primary key. You don't want to prompt for the primary key because if you do that you're not supposed to change it and DATATRIEVE will get all sorts of upset.

Finally all you have to do is take care of the other possibility: if CURRENT-COUNT = 1 and DEL is = to D, then you print it out, ask the user if they really want to delete it and if they says yes, then you delete it.

Finally the last thing you do is you prompt for the next primary key value and if he puts in a primary key value other than END, it will go back to the beginning, otherwise it ends. And that's all there is to it.

Steve Duff - Santa Fe Engineering

One of the problems we had - how many people use FMS interface in DATATRIEVE? Could I see some hands? Oh - how many people use TDMS? that figures.[1] One of the problems we found with the FMS interface was that when you have a request to prompt for a specific field you have a problem and the problem is that since you have one form definition in DATATRIEVE, providing no mechanism for getting at a specific field, what you have to do is play games and I realize that this probably isn't really magic since callable DATATRIEVE lets you do just about thing. It's sort of the keys to the kingdom as it were.

```
FN$GET_FIELD(RETURNS NAME)
FN$PUT-FIELD(NAME)
FN$SET-SUPER("on"/"off")
```

What we first tried was adding these two functions which you actually can do without writing any code - you just put them in the DATATRIEVE Macro definitions and once a form is active in FMS you can call the FDV$ routines that correspond to these - the GET field shouldn't say NAME, that should say RETURNS NAME - I don't have a pen here - does anybody have a pen?[2] And these will allow you once you've activated a form to actually have DATATRIEVE move down 35 fields in the form and prompt for the ones so the user doesn't have to go "tab tab tab tab tab ADC, return." And believe me, users really do like that - they're real pleased with that. However we found that does have an awful lot of limitations, because it

[1] What figures has escaped us. We don't know how many hands went up for what.

[2] Have we fixed it for you Steve?

doesn't go through the DATATRIEVE mechanism for checking valid clauses and so forth and that tends to make it a little bit useless really, is what it boils down to. So we meditated on that a little bit more, it was a good idea and we still use it for a few odds and ends here and there, but one of the things you can do in FMS which DATATRIEVE at least as far as I know, provides no access to is supervisor mode - and that was a real inspiration, at least to me. What you can do is provide a call into DATATRIEVE through the FN$ routine - it goes into the FDV the form driver routine to turn supervisor mode on and off and then define the field that you don't want people to have to tab through in supervisor mode and the form driver is smart enough to know that you shouldn't have to tab through all those fields - it will just simply skip over them when you hit the tab, and that allows you effectively to use a single form for multiple applications - one thing I might note here just to close this off is that there does seem to be a bug in V2 DATATRIEVE - it's a minor one - if you do go ahead and do this you will probably want to define the supervisor mode business as a function which returns no value since it really does nothing of any consequence to give you back - and if you do that it won't work, and you wonder why that is - and the reason it isn't working is that it seems - maybe this is known - that the compiler optimizes somehow a call to the routine which has no value and calls it before it goes ahead and executes the routine so if you change the value of the switch while you're executing and then come back a second time its just not going to bother to do anything at all and the whole thing is really mysterious - it took quite a long time to sort that out - we really thought we had a problem there, but once we decided we would just return a no value, it seems like DATATRIEVE is happy to wait until run time to do that - and that's it.

John Jones - FDA

I kind of feel alone in here. I'm wondering how many people are still under PDP 11's on DATATRIEVE? OK, because that's the situation I find myself in. It was neat coming here and hearing about words like reduce for cutting things down, but one of the problems we've had for a long time was concerning unique value counts. And we went along with our users for a long time just saying DATATRIEVE doesn't do it, and we'll give you a record count, we'll add up anything you want us to but don't ask us for how many occurrences of something that it had.

We finally had a heavy come in and said I need the stuff and so we had to develop something for it. And basically what we had was something like this:

```
INSP-REC USING
```

```
01 INSP-REC.
      05 FIRM-NAME         PIC X(7).
      05 INSP-DATE         PIC X(6).
      05 PROJ
      05 PROD
         .
         .
         .

DECLARE COMP-FLD PIC X(13).
```

- what we're interested in for instance, was a count of inspections that some people had made on various firms and we do not have unique value keys. We are also sequenced then with inspection date and we have a lot of other information in there. To count inspections, basically what we had to do then was to run through and concatenate the firm number with the inspection date and just count those occurrences.

This was fine if all you needed was sort of a grand total of inspections over the range of the file. What was needed tho, was something that gave us that count every time another field would change - in this case it was the project that we were inspecting firms under. What we did was basically to create a new file for store purposes that we call the COUNT_FILE.

```
COUNT-FIL-REC USING
01 COUNT-REC.
      05 CTL-FLD PIC X(13).
      05 UNIQUE-CTR USAGE COMP.
```

We went through the inspection file once, creating count file records which contain the control field of firm number concatenated with the inspection date and then the count that was generated every time we had that control break.

We then set up a view of the inspection file with the count file linking in to the inspection file this count file that we had just created based on the control field of that file equaling the inspection date concatenated with the firm number.

```
VIEW OF INSP-FILE, COUNT-FIL

01 INSP-ELEM OCCURS FOR INSP-FIL.
      05 FIRM-NUM FROM INSP-FIL.
         .
         .
         .
      05 UNIQUE-PART OCCURS FOR COUNT-FIL WITH
         COUNT-FIL WITH CTL-FLD EQ INSP-REC.FIRM-NUM|INSP-DATE
         07 UNIQUE-DTR.
```

The AT BOTTOM STATEMENT for our report:

```
AT BOTTOM OF PROJECT PRINT ALL UNIQUE-CTR OF UNIQUE-PART
```

Then we could then print basically the occurring groups from the view - ignoring the rest of it - That was one way of getting at that problem for us.

Peggy Racel - EG&G

I don't know how many of you were like me and found DATATRIEVE, heard about the date type and thought it was the most fantastic thing they had ever seen. You could subtract with dates, you could add with dates, you could do everything you ever wanted to do with dates without having to convert them to Julian. Well I did this and I put a lot of record descriptions, put my date type in the record description and later on I changed to write one of my routines in COBOL and suddenly I had no idea what was in my date - and I struggled for probably 2 weeks to try to write a subroutine that converted the date - and if anyone could do it in COBOL I would love to see it done, but what I finally ended up having to do - I knew somebody would - is write a subroutine in FORTRAN that converted my date and then called the subroutine from a COBOL program:

```
SUBROUTINE CHGDT(DTRDT,DISPDT)
CHARACTER*8 DTRDT
INTEGER*4 DATETMP(2)
CHARACTER*11 DISPDT
15 FORMAT(A8)
   ENCODE(16,15,DATETMP)DATEDTR
   CALL SYS$ASCTIM(DISPDT, + DATETMP,0)

   CALL SYS$BINTIM(DISPOT,DATETMP)
45 FORMAT(A8)
   DECODE(8,45,DATETMP)
+ DTRDT
```

The secret was my integer times 4 which repeated 2 times. Subtracting the dates and doing all that sort of math isn't really the problem I had - the problem was that if I wrote a program in COBOL and I wanted to display the date and it was in 8 funny looking characters - I wanted to have a routine that I could convert those funny looking characters to something that I could display.

Bert Roseberry - U.S. Coast Guard

New person - for those of you users that don't really know how to go into DATATRIEVE and do any modifications, create user defined functions, such as SPAWN, this is a really neat way to answer the phone while you're in DATATRIEVE.

```
DTR> READY YACHTS
Jasmann is phoning you ...
DTR> ^Y
$ SPAWN
Long Message 1 2, 3, and 4
$ PHONE/ANSWER
 .
 .
 .
$ EXIT (out of phone)
$ LOGOFF
Other Long Message 1, 2, 3, and 4
$ CONTINUE
DTR> (with YACHTS still readied)
```

I believe all the other examples they showed an example of using a user defined function of SPAWN what you do: you're in DATATRIEVE, you ready a domain, you get the message somebody's phoning you, you control Y out, you're at the $ prompt, and then from there you SPAWN, you get a long message, I don't remember what it was, you get about 4 or 5 of them at least, then you answer the phone, talk to the person, you exit, you log off and then you type continue, and you're right back in DATATRIEVE again with the YACHTS still ready, so that's -- honest. This is on a VAX system.

Unknown person: that was documented in a release notes for Version 3 of VMS for those you who bother to read the release notes.

Steven Ward - Lockheed Georgia

I have a problem where I have people who always need a report that takes up more than 132 columns. I do a lot of BASIC PLUS 2 programming where I play around with the horizontal print characteristics of an LA 120. I don't know how many of you others use an LA 120 but I have a way of printing up to 217 columns, a DATATRIEVE report of 217 columns and what it requires is a small procedure I call ESC so that you can pass the escape character to the LA 120 from a DATATRIEVE report writer procedure.

```
DEFINE PROCEDURE ESC
DECLARE ESC PIC X.
ESC =
END-PROCEDURE

:ESC
REPORT ON DOODAH.TMP
SET COLUMNS-PAGE = 217
AT TOP OF REPORT PRINT
ESC||"4W",NEW-PAGE
PRINT DOODAH-REC
```

And what you have to do is create a procedure called ESCAPE or I call it ESC. You have to do this from the EDT editor because DATATRIEVE doesn't recognize the escape character as far as I know. You create a procedure, define procedure escape from the EDT editor, declare a variable called ESC, give it a single alpha numeric field, and you say escape = and I can't draw this in here cause you can't write an escape character, but from EDT, if you type escape twice then it will generate the ASCII code for escape and recognize it as such. Then you end the procedure.

Go into DATATRIEVE, build that into your DATATRIEVE dictionary, then when you do a report, I always build my reports in procedures cause I always have to modify them. The very first thing you do at the top of your report is recall this procedure called ESC, then you just say report on DOODAH.TMP whatever file or output device, and then whenever you want to change the print characteristics of your LA 120, here I say set columns page 217 cause that's the maximum you can do it, 16 and one half characters per inch, before any print statement. Before you change characteristics, you say print, I generally do at the top of the report, I say at top of report print escape, double bar, and " whatever the character code is that follows the escape character to change it to the particular horizontal pitch that you want. I don't exactly remember off the top of my head what 4W is but I think that changes it to 16.5 characters per inch, which gives you 217 columns and you can print a report on 13 inch wide paper, 217 columns worth.

I have one where I mix it - I print a large headline and print the title page in 5 characters per inch and then I print the entire report at 16.5 characters per inch, and as far as spacing it you can specify the column number but I haven't figured out any algorithm that tells me exactly what column, I usually print it and it's never where I want - I get as close to the column and I print a sample and it doesn't come out where I want, I back it up or I move it ahead a few columns till I get it exactly where I want. Thank you.

Bert Roseberry - U.S. Coast Guard

I've been instructed to show people how to compress a common data dictionary. As far as I can remember this is what you need to do it - it's pretty simple - there's a very good help facility in CDDV just like any other DATATRIEVE product and it explains it pretty well:

```
RUN SYS$SYSTEM:CDDV
CDDV> COMPRESS CDD.DIC NEW.DIC
```

That's all there is.

Datatrieve Masters

The following list of names identifies people who have agreed to provide assistance to other Datatrieve Users:

Operating System	Computer	Name	Telephone
IAS	11/70	Chuck Watson Battelle Northwest Richland, Washington 99352	(509) 376-2227
RSX	11/70	Joe Kelly Waymon Gordon Worchester Road North Grafton, Massachusetts 01536	(617) 839-4441 Extension 5480
		Bart Lederman ITT World Communications 67 Broad Street New York, New York 10004	(212) 797-8080
		Gene Roloff Teketronix, Inc. Beaverton, Oregon 97077	(503) 627-1196
VMS	780	Larry Jasmann 124 Caneel Court Gretna, Louisiana 70053	(504) 865-5631 Extension 9

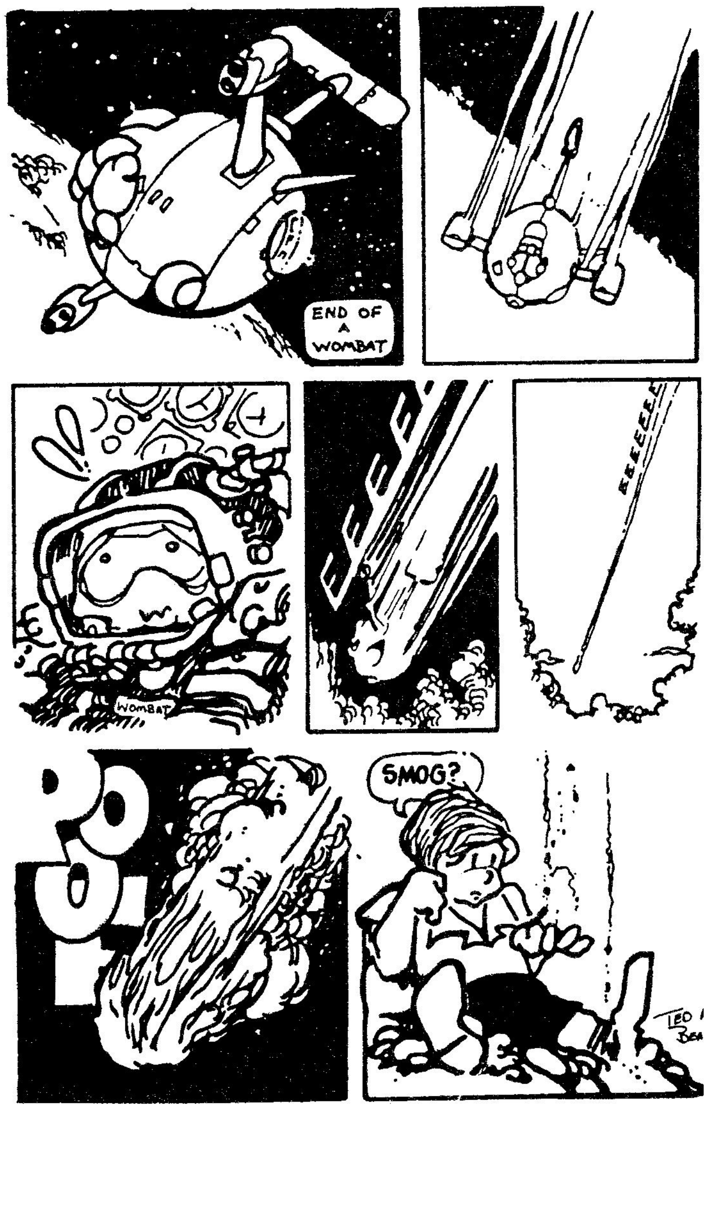
Operating System	Computer	Name	Telephone
		Joe Kelly Waymon Gordon Worchester Road North Grafton, Massachusetts 01536	(617) 839-4441 Extension 5480
		David Saad United Technologies Micro Electronics Center 1365 Garden of the Gods Road Colorado Springs, Colorado 80907	(303) 594-8098
		James Swanger G.D. Searle & Co. P.O. Box 5110 Chicago, Illinois 60680	(312) 982-7430
		Darrell Eade Naval Undersea Water E.S. Heyport, Washington 98345	(206) 396-2501
		Chris Wool E.I. DuPont Engineering Department Wilmington, Delaware 19898	(302) 366-4610
	750	Dick Azzi Motorola Corporation 5005 E. McDowell Road Phoenix, Arizona	(602) 244-4316

Miscellany

A challenge was raised in the last issue: determine what is at the end of a wombat and you'll win 35 glorious cents.

Well, Ted Bear, editor of the BASIC SIG newsletter sent in one version of the end of a wombat. I cannot believe that this truly represents "the end of a wombat," find the idea that someone would send our dear little mascot hurtling through space to an ignominious and probably painful end incredibly sad and am therefore not going to send Mr. Bear any money. So there!

I do, however, thank him for the "entry."



This form is for use by U.S. chapter members only.

DECUS

DECUS SUBSCRIPTION SERVICE ORDER FORM

RETURN TO: Subscription Service
DECUS
One Iron Way, MRO2-1/C11
Marlboro, MA 01752

- All checks payable to DECUS
- All orders MUST be paid in full
- No refunds will be made
- Prices indicated are FY'84 prices
- No purchase orders accepted

Check # ________
Bank # ________
Amount $ ________

Name ________ (First) ________ (Last) DECUS Membership No. ________

Company/Affiliation ________

Mailing Address ________ Mail Stop ________

City ________ State/Country ________ Zip Code ________ Phone ( ) ________

CODE	PUBLICATION	CODE	PUBLICATION
MSL	MUMPS/STRUCTURED LANGUAGES NEWSLETTER	RST	RSTS NEWSLETTER
LHS	LABS/HMS/SITE MGMT NEWSLETTER	LGS	LARGE SYSTEMS NEWSLETTER
OAD	OA/DIBOL/COBOL/GRAPH/PC NEWSLETTER	EDU	EDUSIG NEWSLETTER
VAX	VAX/VMS NEWSLETTER	DTR	DATATRIEVE NEWSLETTER
RSX	RSX/IAS NEWSLETTER	NTW	NETWORKS NEWSLETTER
RT	RT11 NEWSLETTER	SOS	SS&OS NEWSLETTER
SPR	Spring Proceedings	BAS	BASIC NEWSLETTER
FAL	Fall Proceedings	APL	APL NEWSLETTER
		ALL	ALL PUBLICATIONS PRODUCED

	Insert Code From Above:	Check One:		
BASIC PLAN: This plan allows you to receive one (1) selection for one year	☐	☐	Member/DIGITAL Employee	$ 12.00
		☐	Non Member	$ 24.00
STANDARD PLAN: This plan allows you to receive up to three (3) selections at one low price.	☐ ☐ ☐	☐	Member/DIGITAL Employee	$ 25.00
		☐	Non Member	$ 50.00
DELUXE PLAN: This plan allows you to receive up to six (6) selections for one year.	☐ ☐ ☐ ☐ ☐ ☐	☐	Member/DIGITAL Employee	$ 45.00
		☐	Non Member	$ 90.00
ALL: This will allow you to receive all publications listed above for one year for only one price.	ALL	☐	Member/DIGITAL Employee	$120.00
		☐	Non Member	$240.00

$20 MINIMUM REQUIRED TO CHARGE

TOTAL AMOUNT OF ORDER $________

Please charge my order to the following:

☐ Mastercard ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐ Exp. date ________

☐ Visa ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐ Exp. date ________

I understand that neither DECUS nor Digital Equipment Corporation is responsible for any publication not published by a Special Interest Group or the contents of any publication published by a Special Interest Group. I also understand that there will be no refunds even if I decide to cancel my subscription.

Signature ________ Date ________

DIGITAL Employees Only: Badge No. ________ C.C. ________

Cost Center Manager's Signature ________ C.C. ________

3/83