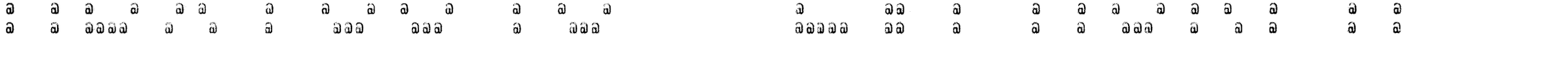
B**BB**BB**BB**BB**BB**BB**BB**BB**BB**BB**BB**BB**BB* COMPUTER SHARING SERVICES *BB**BB**BB**BB**BB**BB**BB**BB**BB**BB**BB**BB**B3**B
B**BB**BB**BB**BB**BB**BB**BB**BB**BB**BB**BB**BB**BB* COMPUTER SHARING SERVICES *BB**BB**BB**BB**BB**BB**BB**BB**BB**BB**BB**BB**B3**B



B**BB**BB**BB**BB**BB**BB**BB**BB**BB**BB**BB**BB**BB* COMPUTER SHARING SERVICES *BB**BB**BB**BB**BB**BB**BB**BB**BB**BB**BB**BB**B3**B
263 263 263 263 263 263 263 263 263 263 263 263 263 263 263 263 263 263 263 263 263 263 263 263 263 263 263 263 263 263 263 263 263

02/04/82 15:49:05 PRINTOUT #263

DENNIS HAUGH

```
% title 'Disk Pack Formatter';
% index;
/*

 $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
 $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
 $$                                                            $$
 $$                                                            $$
 $$           PROPRIETARY TRADE SECRET INFORMATION             $$
 $$                                                            $$
 $$   TO BE USED ONLY UNDER LICENSE FROM DTSS INCORPORATED.    $$
 $$                                                            $$
 $$                                                            $$
 $$     UNPUBLISHED COPYRIGHTED WORK OF DTSS INCORPORATED.     $$
 $$                                                            $$
 $$                                                            $$
 $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
 $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$



  THE RELEASE DATE OF THIS VERSION OF packfmt IS:

                15 July 81


  PLEASE INCLUDE THIS DATE IN ALL CORRESPONDENCE WITH
  DTSS INCORPORATED CONCERNING THIS VERSION OF packfmt

   See Document 1372 for instructions.


                          Points of interest:

   Document 1372 has instructions.

   Honeywell EPS-1 DSC181/DSC190 Controller (rev. B) 43A232230 has details.

   Look at the section describing the ATT (Alternate Track Table).  That is our
   basic data structure.

   Look at the section describing the Track Header.  That is the hardware on
   which we operate.

   Indications are that MSU501 disks are best handled by a different program.
   There is almost no resemblance between our formatting and theirs.

   If you're not sure about things, get a spare pack & play around with it.


                             Things to Do:
```

```
53              This program uses several insert files and library routines.  When an
54              official software library is set up it should be changed to reference that.
55              Look for comments with *****.
56
57              Allow PackFmt to format allocated devices (save it with more trap bits).
58              Make it ask the user whether it really means to mess with the file system.
59              Allow only non-destructive commands, ALTERNATE, and perhaps TI.  Don't allow
60              TEST, FORMAT, REFORMAT.
61
62              Think about MSU501s.
63
64
65                                          Things Not to Do:
66
67              Call me if there's any problem.
68
69
70                                          Thought:
71
72          Since Roe and Shakow (1942) have shown that a variety of mental conditions may impair the
73          repitition of digits forward, the question may be raised whether "attention" rather than
74          retention is the determining factor in this performance.  The results with auditory digit
75          span backward, on the contrary, indicate that the performance is particularly impaired
76          in general paresis, chronic alcoholism with psychosis, hebephrenic and unclassified
77          dementia praecox.  These latter findings would indicate a closer relationship between the
78          difficulties with auditory digit span backward and mental disease.
79
80          Digits forward (Correct in 1 of 2 trials)       Digits backward (Correct in 1 of 2 trials)
81          7 good average, 6 low average, 5 marginal       5 average, 4 marginal
82
83          Kuhlmann 1939; Terman and Merill 1937; Wechsler 1941; Peatman and Locke 1934
84
85          -- Wells and Ruesch. Mental Examiner's Handbook.
86             New York: The Psychological Corporation, 1972
87
88          */
```

```
      89          % page;
      90
      91
      92          % library 'noprom.b';                          /*******/
      93
      94    PackFmt:proc options (main,noPage,noClear);
      95
      96              % include 'literals';                      /*******/
       1              %Declare and          literally '&';
       2              %Declare or           literally '|';
       3              %Declare not          literally '~';
       4              %Declare false        literally '''0''b1';
       5              %Declare true         literally '(not false)';
       6              %Declare boolean      literally 'bit (1) aligned';
       7              %Declare indefinitely literally 'while (true)';
      97              %Declare elif         literally 'else if';
      98              %Declare result       literally 'parameter';
      99
     100              dcl PerCatFrn fixed,                       /* FRN for PERCAT (opened in Init) */
     101                  PerCat    int file constant;           /* PERCAT, opened */
     102
     103              dcl SysIn    ext file constant,            /* default input source */
     104                  SysPrint ext file constant;            /* default output destination */
```

Exec Calls

```
105             % subtitle 'Exec Calls';
106
107             /* Issues a MME and checks that the status is zero */
108
109 MME0:       proc (M,R);
110                 dcl M fixed parameter,                  /* MME number */
111                     R (12) result parameter;            /* registers */
112
113                 call MME(M,R(*));                       /* issue */
114                 if (ShR(R(11),18) & 777b3) ~= 000b3     /* check exec status */
115                 then call ErrExit(SubStr(Octal(M),7,6)||' status '||Octal(R(11)));
116
117             end MME0;
```

PIO -- Definitions

```
118             % subtitle 'PIO -- Definitions';
119
120             /*   This is the format of a DCW for drives with DCWs. The MBZ fields are filled
121                in by the Exec's PIO routines.
122                     The DCWs described here are only those used from slave mode for the drive.
123                They are described in TM059. A description of hardware DCWs is in TM066. */
124
125             dcl 1 DCW       based,
126                   2 Offset  unsigned (18),              /* slave memory address */
127                   2 MBZ     bit (4),
128                   2 Action  bit (2),                    /* action (see below) */
129                   2 Count   unsigned (12);              /* record count */
130
131             /* definitions of action codes */
132             % dcl i$iotd  lit ' ''00''b ',              /* I/O transfer & disconnect */
133                   i$iotp  lit ' ''01''b ',              /* I/O transfer & proceed */
134                   i$tdcw  lit ' ''10''b ',              /* transfer to DCW (not legal from slave mode) */
135                   i$iontp lit ' ''11''b ';              /* I/O nontransfer & proceed */
136
137             /* This is the structure of the status returned by the EXEC on a drive with DCWs.
138                It can be found in Pub.1059 */
139
140             dcl 1 PIOStatus  unaligned based,
141                   2 Acc      bit (9),
142                   2 Exec     bit (9),                   /* EXEC status return */
143                   2 CSt1,                               /* channel status word 1 */
144                     3 Sync   bit,                       /* indicates status came from IOM */
145                     3 Off    bit,                       /* device was offline */
146                     3 Major  unsigned (4),              /* device major status */
147                     3 Minor  bit (6),                   /* device minor status */
148                     3 QAdd   unsigned (6),              /* queue address (EXEC simulated) */
149                   2 DCWNo    bit (36);                  /* number of bad DCW on failure */
150
```

PIO -- Track Header

```
151             % subtitle 'PIO -- Track Header';
152
153             /* This is the structure of a track header, as used by the READ TRACK HEADER
154                and FORMAT TRACK commands. More details are in EPS-1 "DSS191 and DSS190
155                Removable Media Disk Storage Subsystems" */
156
157             dcl 1 TH          unaligned,
158                   2 HA        unaligned,              /* home address of this sector */
159                     3 Cyl     unsigned (16),          /* cylinder number */
160                     3 Hd      unsigned (16),          /* head     number */
161                     3 H       bit,                    /* rewrite HA only on DSS190 (format only) */
162                     3 Z       bit,                    /* header bypass: ignore current HA (format only) */
163                     3 TI      bit (2),                /* Track Indicator (status of track) */
164                   2 ROCount   unaligned,              /* record zero count field */
165                     3 MBZ0    bit (4),
166                     3 Flag    unaligned,              /* flag field */
167                       4 A     bit,                    /* alternate format (should be zero) */
168                       4 MBZ   bit (5),
169                       4 TI    bit (2),                /* record zero TI bits */
170                     3 Cyl     unsigned (16),          /* cylinder number */
171                     3 Hd      unsigned (16),          /* head     number */
172                     3 Rec     unsigned (8),           /* record   number */
173                     3 MBZ1    bit (2),
174                     3 ChkChr  bit (6),                /* check character (format only) */
175                     3 MBZ2    bit (12),
176                   2 RODataa   unaligned,              /* record zero data field */
177                     3 MBZ     bit (4),
178                     3 Data    bit (64),               /* data */
179                     3 ISZ     bit (4);                /* is zero (fill) */
180
181             /* definitions of TI settings */
182
183             % dcl TI$GP lit ' ''00''b ',              /* primary   track, good */
184                   TI$GA lit ' ''01''b ',              /* alternate track, good */
185                   TI$BA lit ' ''10''b ',              /* defective track,    alternate assigned */
186                   TI$BN lit ' ''11''b ';              /* defective track, no alternate assigned */
187             /* an I/O descriptor for the above */
188
189             dcl 1 THDCW like DCW unaligned; THDCW = IOTD(WAddr(TH),WLen(TH));/* set up DCW */
190
191             /* convert the HA (header address) to a standard form (CCC/HH) */
192
193  HAName:    proc returns (char(6));
194                 return (TrackName(Track((TH.HA.Cyl),(TH.HA.Hd))));/* just fix it up for someone else */
195             end HAName;
```

PIO -- Drive Modes

```
196             % subtitle 'PIO -- Drive Modes';
197
198             /* these are for data transfer operations, called DriveWithDCWs */
199
200             % dcl d$read  lit '400000b3',                  /* read */
201                   d$write lit '600000b3',                  /* write */
202                   d$rhead lit '420000b3',                  /* read track header */
203
204             /* the format track drive has the form 43X000 where X000 = TI*1000 */
205
206                   d$form   lit '430000b3',                 /* the basic format command */
207                   d$formGP lit '430000b3',                 /* good primary */
208                   d$formGA lit '431000b3',                 /*      alternate */
209                   d$formBA lit '432000b3',                 /* bad  alternated */
210                   d$formBN lit '433000b3',                 /*      alternateless */
211
212             /* these are single action drives (no DCWs) */
213
214                   d$recov lit '140000b3',                  /* enable EXEC error recovery */
215                   d$norcv lit '150000b3';                  /* disable EXEc error recovery */
216
217             /* makes a DCW. Call this with the word address of the buffer and the word length,
218                both in fixed form. */
219
220  IOTD:      proc (A,L) returns (1 like DCW unaligned);
221                 dcl A fixed parameter,                     /* address of buffer */
222                     L fixed parameter;                     /* length of buffer */
223
224                 dcl 1 D like DCW unaligned;                /* make a DCW here */
225
226                 D.Offset = A;                              /* address */
227                 D.MBZ = ''b;                               /* make zero */
228                 D.Action = i$iotd;                         /* I/O transfer & disconnect */
229                 D.Count = L;                               /* length */
230
231                 return (D);
232
233             end IOTD;
```

PIO -- Drive with DCWs

```
234             % subtitle 'PIO -- Drive with DCWs';
235
236             /* Do a drive with DCWs. This is the slave interface into the EXEC's PIO
237                (Physical I/O) routines.  This drive must be done on a device file. The action
238                is specified by a code known as I$MODE, after a symbol in the EXEC. The I$MODEs
239                are mapped into device commands by the EXEC.
240                   This procedure takes a device file, opened as a PL1 file, a mode code, and
241                a single DCW. It returns a PioStatus. */
242
243 PIO:        proc(File,Mode,D) returns (1 like PioStatus unaligned);
244                 dcl File file  parameter,                  /* file to do it to */
245                     Mode fixed parameter,                  /* I$MODE */
246                   1 D like DCW unaligned;                  /* the DCW */
247
248                 % dcl mme$Drive lit '500232b3';            /* for funny IO */
249
250                 % list off; % include 'regs'; % list on;   /*******/
251
252                 dcl 1 S like PioStatus unaligned;          /* status return */
253
254                 X0 = Frn(File);                            /* FRN of device file */
255                 X1 = WAddr(D);                             /* DCW */
256                 X4 = 0;                                    /* no flags */
257                 X6 = 0;                                    /* use BI trap */
258                 X7 = 1;                                    /* number of dcws */
259                 RA = 24000000b3 + Mode + 1;                /* mode and record count */
260
261                 call MME(mme$drive,Regs);                  /* Issue DWDCW */
262
263                 String(S) = UnSpec(SWs);                   /* return status */
264
265                 return (S);
266
267             end PIO;
```

PIO -- Drive with DCWs

```
268             % page;
269
270             /* This calls PIO and checks the status. If it looks bad it is printed in octal
271                and ErrExit is taken. */
272
273 PIOD:       proc (Mode,D);
274                 dcl Mode fixed parameter,                   /* the type of drive */
275                   1 D like DCW parameter unaligned;         /* the (single) DCW */
276
277                 dcl 1 S like PioStatus unaligned;           /* the status */
278
279                 S = PIO(Pack.File,Mode,D);
280
281                 if S.Exec = 000b3 or S.Exec = 001b3 or S.Exec = 400b3 or S.Exec = 420b3
282                 then do;                                    /* acceptable EXEC status */
283                     if not S.Off and S.Major = 0
284                     then return;                            /* ok PIO status */
285                 end;
286                 call ErrExit('PIO status '||Octal(FixedBin(SubStr(String(S),1,36))));
287
288             end PIOD;
```

Pack -- Attributes

```
289             % subtitle 'Pack -- Attributes';
290
291             /* This structure is similar to a FCB. It contains what we know about our pack.
292                The label is not part of this structure, but is available as Track0.
293                If we have no pack then Pack.File=NullF().
294
295                Since we are interested in only one pack, the members of this structure are
296                usually referred to without full qualification. To make things simpler,
297                certain invariants are literal constants. */
298
299             dcl 1 Pack,                                  /* data associated with the particular pack */
300                    2 File     file variable,             /* the PL1 file */
301                    2 Loc      fixed,                     /* current *logical* address */
302                    2 HasATT   boolean,                   /* true iff ATT is good */
303                    2 Name,                               /* pack name */
304                     3 Prefix  char (6),                  /* PERCAT prefix */
305                     3 Suffix  char (2),                  /* PERCAT suffix */
306                    2 Shape,                              /* various dimensions */
307                     3 LogRecs fixed,                     /* number of physical records to a logical */
308                     3 UseCyls fixed,                     /* number of user cylinders */
309                     3 Cyls    fixed,                     /* sum, total number of cylinders */
310                     3 Range   fixed;                     /* number of logical records */
311
312             /* the following dimensions are constant */
313             % dcl RecSize lit '64',                      /* words/record */
314                   Sectors lit '40',                      /* sectors/track */
315                   Heads   lit '19',                      /* heads/cylinder */
316                   AltCyls lit '3',                       /* alternate cylinders */
317                   TDCyls  lit '1';                       /* T & D cylinder */
318
319             /* make sure that there is a current pack */
320
321  PackCheck: proc; if Pack.File = NullF() then call ErrExit('no pack.'); end PackCheck;
```

Pack -- Open

```
322             % subtitle 'Pack -- Open';
323
324             /* This procedure opens a device file in :PERCAT as the current drive. The
325                argument should be a string with the pack name. This name should look like
326                a default pack name created during reconfiguration. (See SM326, Environment
327                Deck.) The physical type name (prefix) must be one of those known to this
328                program.
329
330                Any previously opened pack is closed. The named pack is opened. If it cannot
331                be opened with RWA ErrExit(stw1) is taken. Otherwise the Pack data is set up,
332                based on the physical type name. Executive error recovery is disabled and
333                the file pointer is set to 0. */
334
335 GetPack:    proc(NameStr);
336                 dcl NameStr char var parameter;          /* the name given */
337
338                 dcl FCB     int file constant,           /* a place to open files */
339                     Name    char (8) var,                /* pack name */
340                     I       fixed;                       /* index into the types table */
341
342                 /* close any pack which was open */
343
344                 call DropPack;                           /* thus */
345
346                 /* check the name for length & make it handy */
347
348                 if Length(NameStr) < 2 or Length(NameStr) > 8
349                 then call ErrExit('name should be 2-8 characters');
350                 else Name = NameStr;                     /* get into a handier form */
351
352                 /* Open the pack in :PERCAT (which is already open) with RWA. We print our
353                    own message in case of failure. */
354
355                 begin;                                   /* error recovery block */
356                     on undefinedfile(FCB) go to Fail;    /* we decide what's failure */
357                     open file (FCB)                      /* open this FCB */
358                          title (Name)                    /* with the supplied name */
359                          env (catfrn (PerCatFrn)         /* in :PERCAT */
360                               unformatted                /* not your average file */
361                               access (007000b3));        /* (RWA in the vulgar) */
362 Fail:           end;                                     /* leave quietly on failure */
363
364                 if (StW1(FCB)&000777000000b3) ^= ''b
365                 then call ErrExit('can''t open '||Name||'; status '||Octal(StW1(FCB)));
```

Pack ** Open

```
366                 % page;
367
368                 /* Analyze the name and determine the physical device type */
369
370                 Pack.Suffix = SubStr(Name,Length(Name)-1,2);/* suffix is last 2 letters */
371                 Pack.Prefix = SubStr(Name,1,Length(Name)-2);/* prefix before */
372
373                 do I = 1 to HBound(Type,1) while (Type(I).Name~=Pack.Prefix); end;
374
375                 /* If the pack was of a known type then set its characteristics. If not, then give it
376                    up. By this devious means we hope to enforce standardization of names. Also
377                    set other attributes of the pack */
378
379                 if I > HBound(Type,1)                       /* not found? */
380                 then call ErrExit(Pack.Prefix||' is not a valid device type.');
381
382                 Pack.File = FCB;                            /* accept this pack */
383                 Pack.LogRecs = LogRecs(I);                  /* records/logical records */
384                 Pack.UseCyls = UseCyls(I);                  /* user cylinders */
385                 Pack.Cyls    = Pack.UseCyls + AltCyls + TDCyls;/* total cylinders */
386                 Pack.Range   = Pack.Cyls * Heads * Sectors / Pack.LogRecs;/* number of logical records */
387
388                 /* Also determine the code to be used in the ATT (see below) to mark a free
389                    alternate track. My guess is that when 451s were introduced and negative
390                    cylinder numbers (>=512) became legal, it was too late to fix the old ones. */
391
392               String(FreeTrk) = FreeCode(I);                /* set up table marker */
393
394               /* Disable EXEC error recovery on the pack & set the file pointer to 0. */
395
396               call Drive(Pack.File,d$norcv);
397               call FileP(0);                                /* get into a known state */
398
399               Pack.HasATT = false;                          /* assume nothing */
400               if ReadTrack0() then if GoodLabel() then if GoodATT() then Pack.HasATT = true;
401               if not Pack.HasATT then put line ('unlabelled pack');/* keep op informed */
```

Pack -- Types

```
402                 % subtitle 'Pack -- Types';
403
404                 /* This is a table of the physical characteristics of all physical pack types
405                    known to this program. */
406
407                 dcl 1 Type(2)     static structure,       /* known pack types */
408                       2 Name char (6) init (              /* physical type name */
409                           'D191',                         /* DSS 190B, also called DSS 191 */
410                           'M451'),                        /* MSU 451 */
411                       2 LogRecs fixed init (              /* physical records per logical record */
412                            2,                             /* 2 physical record per DSS191 record */
413                            4),                            /* 4    "       "     "  MSU451   "    */
414                       2 UseCyls fixed init (              /* cylinders per pack */
415                            407,                           /* 407 cylinders per DSS191 pack */
416                            811),                          /* 811     "       "  MSU451  "   */
417                       2 FreeCode bit (18) init (          /* code to mark free ATT entry */
418                            '400000'b3,                    /* for DSS191 pack */
419                            '777777'b3);                   /* for MSU451 (cyl. 512 is legal) */
420
421               end GetPack;
```

Pack -- Close

```
422             % subtitle 'Pack -- Close';
423
424             /* This procedure closes the pack if there is one. Before doing so, Executive
425                error recovery is re-enabled. */
426
427  DropPack:  proc;
428
429                 if Pack.File ~= NullF()
430                 then do;                                /* only if there's something */
431                     call Drive(Pack.File,d$recov);      /* enable error recovery */
432                     close file (Pack.File);             /* detach */
433                     Pack.File = NullF();                /* mark as closed */
434                 end;
435
436             end DropPack;
```

I/O -- Drive

```
437             % subtitle 'I/O -- Drive';
438
439
440             % dcl mme$drive lit '500132b3';              /* MME DRIVE */
441
442             /* This procedure issues a single-action drive on a device file.
443                The status is checked. Call with the drive code (see above) */
444
445 Drive:      proc (File,Code);
446                 dcl Code fixed parameter,               /* action code */
447                     File file  parameter;               /* file to drive it on */
448
449                 % list off; % include 'regs'; % list on;  /*******/
450
451                 X0 = Frn(File);                         /* FRN to drive */
452                 X4 = 0;                                 /* no flags */
453                 X6 = 0;
454                 RA = 12000000b3 + Code;                 /* action code */
455                 call MME0(mme$drive,Regs);              /* do it right */
456
457             end Drive;
```

I/O -- Set Pointer

```
458             % subtitle 'I/O -- Set Pointer';
459
460             /* This procedure is used because I'm not sure what the PL1 RESET statement does.
461             This issues a SET POINTER on the file, which works for drives as well as COPY type
462             MMEs. */
463
464             % dcl mme$setp lit '500213b3';                  /* set file pointer */
465
466 FileP:      proc (Loc);
467                 dcl Loc  fixed parameter;                  /* loc to set to */
468
469                 % list off; % include 'regs'; % list on;   /*******/
470
471                 X0 = Frn(Pack.File);                       /* on the file */
472                 X6 = 0;                                    /* BI trap */
473                 RA  = Loc;                                 /* pointer setting */
474
475                 call MME0(mme$setp,Regs);                  /* SET PTR */
476
477                 Pack.Loc = Loc;                            /* success, remember that */
478
479             end FileP;
480
481             /* converts <cylinder,head,sector> to seek address for the current pack */
482
483 SeekAd:     proc (C,H,S) returns (fixed);
484                 dcl(C,                                     /* cylinder */
485                     H,                                     /* head */
486                     S)fixed parameter;                     /* sector */
487
488                 if C ~< Cyls or H ~< Heads or S ~< Sectors then call ErrExit('invalid address');
489
490                 return ((((C * Heads) + H)                 /* convert to heads */
491                                       * Sectors) + S);/* then sectors (much easier in APL) */
492             end SeekAd;
493
494             /* converts <Cylinder,Head,Sector> to DTSS record number */
495
496 FileAd:     proc (C,H,S) returns (fixed);
497                 dcl (C,H,S) fixed;                         /* as above */
498
499                 dcl L fixed;                               /* record # */
500
501                 L = SeekAd(C,H,S);                         /* first convert to sector count */
502                 if Mod(L,LogRecs) ~= 0
503                 then call ErrExit('not aligned on a logical record');
504                 else return (L/LogRecs);                   /* convert to logical record number */
505
506             end FileAd;
```

String Functions

```
507             % subtitle 'String Functions';
508
509             /* Drop: drops an initial segment off a string. The first argument is the string
510                to shorten. The second is the index of the first character to leave. If this
511                is zero then nothing is left.  This is most useful in combination with one
512                of the builtin functions, such as:
513
514                     call Drop(S,Verify(S,' '));
515
516             This removes any leading spaces from the string S. */
517
518 Drop:       proc (S,I);
519                 dcl S char var parameter,                  /* the string to decapitate */
520                     I fixed parameter;                     /* first index to leave */
521                 if I = 0 then S = ''; else S = SubStr(S,I);/* thus */
522             end Drop;
523
524             /* Break: breaks off an initial segment of a string. The first argument is the
525                string to break. The second argument is the index of the first character to
526                leave. If this is zero then nothing is left. The substring broken off is
527                returned as the result. This is useful in combination with builtin string
528                functions. e.g.
529
530                     A = Break(B,Verify(B,'0123456789'));
531
532                This removes the leading digits from string B and assigns them to A.
533
534                     A = Break(B,Index(B,';'));
535
536                This assigns A the substring of B before the first semicolon in B.
537                This substring is removed from B, which now begins with the semicolon.
538                If there was no semicolon then A is assigned all of B and B is assigned the
539                empty string. */
540
541 Break:      proc (S,I) returns (char var);
542                 dcl S char var parameter,                  /* the original */
543                     I fixed    parameter,                  /* where to break */
544                     R char var;                            /* the result */
545                 if I = 0                                   /* if there's nothing to leave */
546                 then do; R = S; S = ''; end;               /* then take all */
547                 else do; R = SubStr(S,1,I-1); S = SubStr(S,I); end;/* otherwise take before I */
548                 return (R);                                /* deliver */
549             end Break;
```

Conversions

```
550             % subtitle 'Conversions';
551
552             /* Upper: returns its argument with all lowercase letters converted to
553                uppercase. */
554
555 Upper:      proc (L) returns (char var);
556                 dcl L char var parameter;                   /* the mixed case string */
557
558                 return (Translate(L,'ABCDEFGHIJKLMNOPQRSTUVWXYZ','abcdefghijklmnopqrstuvwxyz'));
559             end Upper;
560
561             /* converts 36 bits of BCD to six characters of ASCII. */
562
563 ACI:        proc(B) returns (char(6));
564                 dcl B bit(36) parameter;                    /* six BCD characters */
565
566                 dcl ASCII(0:77b3) char static init          /* BCD-ASCII map */
567                         ('0','1','2','3','4','5','6','7','8','9',
568                          '[','#','@',':','>','?',' ','A','B','C',
569                          'D','E','F','G','H','I','&','.',']','(',
570                          '<','\','^','J','K','L','M','N','O','P',
571                          'Q','R','-','$','*',')',';','''','+','/',
572                          'S','T','U','V','W','X','Y','Z','_',',',
573                          '%','=','"','!');
574
575                 dcl C(0:5) unsigned (6),                    /* bits as characters */
576                     A(0:5) char,                            /* characters */
577                     I      fixed;
578
579                 String(C) = B;                              /* break bits into characters */
580                 do I = 0 to 5; A(I) = ASCII(C(I)); end;     /* translate thrugh the table */
581                 return (String(A));                         /* glomm back into string */
582
583             end ACI;
584
585             /* converts a six-character string into its 36-bit BCD representation. Both
586                uppercase and lowercase are acceptable. */
587
588 BCI:        proc(S) returns (bit(36));
589                 dcl S char (6) parameter;
590
591                 dcl BCD(40b3:137b3) unsigned (6) static init/* ASCII-BCD map */
592                         (20b3,77b3,76b3,13b3,53b3,74b3,32b3,57b3,
593                          35b3,55b3,54b3,60b3,73b3,52b3,33b3,61b3,
594                          00b3,01b3,02b3,03b3,04b3,05b3,06b3,07b3,
595                          10b3,11b3,15b3,56b3,36b3,75b3,16b3,17b3,
596                          14b3,21b3,22b3,23b3,24b3,25b3,26b3,27b3,
597                          30b3,31b3,41b3,42b3,43b3,44b3,45b3,46b3,
598                          47b3,50b3,51b3,62b3,63b3,64b3,65b3,66b3,
599                          67b3,70b3,71b3,12b3,37b3,34b3,40b3,72b3);
600
601                 dcl I fixed,                                /* character index */
```

Conversions

```
602                     C fixed,                                /* character code */
603                     B(1:6) unsigned (6);                    /* accumulator for result */
604
605                 do I = 1 to 6;
606                     C = Byte(S,I); if C >= 140b3 then C = C-040b3;/* get the character code (uppercased) */
607                     B(I) = BCD(C);                          /* move into result */
608                 end;
609                 return (String(B));                         /* convert to bits */
610
611             end BCI;
```

Conversions -- Dates

```
612             % subtitle 'Conversions -- Dates';
613
614             /* packed date (in MMDDYY form) */
615
616 BDate:      proc returns (bit(36));
617                 dcl 1 D unal, 2(M char(2),_1 char, D char(2),_2 char, Y char(2));/* date structure */
618                 String(D) = Date();                         /* get it from the OS */
619                 return (BCI(D.M||D.D||D.Y));                /* return relevant portions */
620             end BDate;
621
622             /* convert BCD date to ASCII formatted date */
623
624 ADate:      proc (B) returns (char(8));
625                 dcl B bit (36) parameter;                   /* BCD date */
626                 dcl A char (6);                             /* a place to stuff ASCII */
627
628                 if B = (36)'1'b                             /* is this special */
629                 then return ('(never)');                    /* so it is */
630                 else do;                                    /* ordinary BCD date */
631                     A = ACI(B);                             /* convert first */
632                     return (SubStr(A,1,2)||'/'||SubStr(A,3,2)||'/'||SubStr(A,5,2));/* slash up */
633                 end;
634             end ADate;
```

Conversions -- Addresses

```
635             % subtitle 'Conversions -- Addresses';
636
637             /* MakeAddress: converts its argument into a disk address with
638                <Cylinder,Head,Sector>. This can be given in three forms:
639                     CCC/HH/SS     where CCC is the cylinder number, HH the head (surface)
640                                   and SS the sector.  The numbers are in decimal.
641                     CCC/HH        same as above, with a sector of 0 assumed. This is a
642                                   track address.
643                     000000000000 up to twelve octal digits, a seek address.
644
645                The cylinder, head, and sector numbers are the three result parameters.
646
647                This procedure returns true if successful. No error messages are printed. */
648
649  (conversion,stringSize):
650  MakeAddress:proc (Str,Cyl,Hd,Sect) returns (boolean);
651                   dcl Str  char var parameter,          /* string to convert */
652                       Cyl  fixed result parameter,      /* cylinder number result */
653                       Hd   fixed result parameter,      /* head number     result */
654                       Sect fixed result parameter,      /* sector number   result */
655                       Seek fixed,                       /* seek address, if given */
656                       S1   fixed,                       /* index of first slash */
657                       S2   fixed;                       /* index of second slash */
658
659                on conversion go to Fail;                /* be careful */
660
661                S1 = Index(Str,'/');                     /* look for a slash */
662                if S1 = 0
663                then do;                                 /* no slash, is seek address */
664                    get edit (Seek) (b3(Length(Str))) string (Str);/* read as octal */
665                    Cyl  = Seek / (Heads * Sectors);      /* convert to cylinder */
666                    Hd   = Mod(Seek,Heads * Sectors) / Sectors;/*      &     head */
667                    Sect = Mod(Seek,Sectors);             /*      &     sector */
668                end;
669                else do;                                 /* must be given as CCC/HH[/SS] */
670                    Cyl = FixedBin(SubStr(Str,1,S1-1));  /* evaluate CCC */
671                    S2 = Index(Str,'/',S1+1);            /* look for second slash */
672                    if S2 = 0                            /* another / */
673                    then do;                             /* no; must be CCC/HH */
674                        Hd   = FixedBin(SubStr(Str,S1+1));/* head */
675                        Sect = 0;                        /* assume sector 0 */
676                    end;
677                    else do;                             /* CCC/HH/SS */
678                        Hd   = FixedBin(SubStr(Str,S1+1,S2-S1-1));/* head */
679                        Sect = FixedBin(SubStr(Str,S2+1));/* sector */
680                end; end;
681
682                if 0<=Cyl&Cyl<Cyls and 0<=Hd&Hd<Heads and 0<=Sect&Sect<Sectors/* check ranges */
683                then return (true);                      /* successful */
684  Fail:         return (false);                          /* failed */
685             end MakeAddress;
```

Format Pack -- Track 0

```
686             % subtitle 'Format Pack -- Track 0';
687
688             /* Track 0 on a pack is reserved for system information. Record 0 has a pack
689                label, record 3 has the AlternateTrackTable. Records 1 & 2 are used by GCOS
690                for something. Don't ask */
691
692             dcl 1 Track0(0:3) unal,                          /* we'll point into this structure */
693                   2 Rec(0:RecSize-1) char (4);
694
695             dcl 1 Track0DCW like DCW unaligned; Track0DCW = IOTD(WAddr(Track0),WLen(Track0));/* a DCW for this */
696
697             /* initialize all such data to zero. */
698
699  ClearTrack0:proc; UnSpec(Track0) = ''b; end ClearTrack0;
700
701             /* read the track from off the pack without changing the current location.
702                returns true iff it read. */
703
704  ReadTrack0: proc returns (boolean);
705                   dcl PLoc fixed,                            /* saved previous locus */
706                     1 S    like PIOStatus unaligned;         /* device status */
707
708                   PLoc = Pack.Loc;                           /* save pointer */
709                   call FileP(0);                             /* set up for seek */
710                   S = PIO(Pack.File,d$read,Track0DCW);       /* & read it */
711                   call FileP(Ploc);                          /* reset pointer */
712                   if S.Exec = 000b3 or S.Exec = 001b3 or S.Exec = 400b3 or S.Exec = 420b3
713                   then return ((not S.Off) and (S.Major = 0));/* check PIO status */
714                   else signal error;                         /* bad exec status */
715               end ReadTrack0;
716
717             /* Write (updated) track 0 back onto the pack without altering current pointer. */
718
719  WriteTrack0:proc;
720                   dcl PLoc fixed;                            /* saved pointer */
721
722                   PLoc = Pack.Loc;                           /* save state */
723                   call FileP(0);                             /* aim it at record 0.. */
724                   call PIO0(d$write,Track0DCW);              /* and do the 0 */
725                   call FileP(PLoc);                          /* reset */
726
727               end WriteTrack0;
```

Format Pack -- Label

```
728             % subtitle 'Format Pack -- Label';
729
730             /* This is the format of a GCOS pack label, as described in "System Startup and
731                Operation". All fields are in BCD. */
732
733             dcl 1 Label    based (Addr(Track0(0))) unal,  /* label begins on record 0 */
734                   2 Begin  bit (36),                      /* "H-6000" */
735                   2 Site   bit (36),                      /* site name */
736                   2 Numb   bit (36),                      /* pack number */
737                   2 Name   bit (72),                      /* " DISC*PACK  " */
738                   2 Type   bit (36),                      /* "STRUCT" or "NSTRUC" */
739                   2 Misc   fixed,                         /* miscellaneous bits */
740                   2 IDate  bit (36),                      /* initialization date */
741                   2 RDate  bit (36),                      /* retention date */
742                   2 SeqNo  bit (36);                      /* sequence number */
743
744             /* see if a label is acceptable. */
745
746  GoodLabel: proc returns (boolean);
747                 return (ACI(Label.Begin) = 'H-6000'       /* must start thus */
748                     and ACI(Label.Type) = 'STRUCT'        /* we want type STRUCT */
749                     and Label.Misc = 204000000000b3);     /* with STRUCT and ATT attributes */
750             end GoodLabel;
751
752             /* initialize a label for the current pack. */
753
754  NewLabel:  proc;
755
756                 Label.Begin = BCI('H-6000');              /* start of label */
757                 Label.Site  = BCI('DTSS  ');              /* site name */
758                 Label.Name  = BCI(' DISC*')||BCI('PACK  ');/* pack name */
759                 Label.Type  = BCI('STRUCT');              /* pack type */
760                 Label.Misc  = 204000000000b3 ;            /* structured with ATT */
761             end NewLabel;
762
763             /* Update the label & put back track 0 */
764
765  ReLabel:   proc;
766
767                 Label.Numb  = BCI('0000'||Pack.Suffix);   /* record where initialized */
768                 Label.IDate = BDate();                    /* initialized today */
769                 Label.RDate = BCI('!!!!!!!');             /* never destroy */
770                 Label.SeqNo = BCI('!!!!!!!');             /* no sequence numbers */
771                 call WriteTrack0;                         /* put it back on the pack */
772                 Pack.HasATT = true;                       /* it has an ATT now */
773             end Relabel;
```

Format Pack -- Alternate Track Table

```
774             % subtitle 'Format Pack -- Alternate Track Table';
775
776             /* Track codes are 18 bits, 10 bits for the cylinder number and 8 for the head
777                (surface) number. FreeTrk is not the code of any actual track. The value
778                used depends on the type of pack. */
779
780             dcl 1 TrackCode unaligned based structure,    /* prototype */
781                   2 Cyl unsigned (10),                    /* cylinder number */
782                   2 Hd  unsigned (8);                     /* head (surface) number */
783
784
785             /* Track: constructs a track code from the cylinder & head numbers */
786
787 Track:      proc(Cyl,Hd) returns (1 like TrackCode unaligned);
788                 dcl Cyl fixed parameter,                  /* cylinder number */
789                     Hd  fixed parameter,                  /* head (surface) number */
790                     1 T like TrackCode unaligned;         /* result */
791                 T.Cyl = Cyl; T.Hd = Hd;                   /* fill it in */
792                 return (T);                               /* deliver */
793             end Track;
794
795             /* convert a track code to text for output */
796
797 TrackName:  proc (T) returns (char (6));
798                 dcl 1 T like TrackCode unaligned parameter;/* track code (see above) */
799                 dcl 1 N static unal, 2(CCC pic '999', S char init ('/'), HH pic '99');/* picture */
800                 N.CCC = T.Cyl; N.HH = T.Hd;               /* make a picture */
801                 return (String(N));                       /* string it */
802             end TrackName;
803
804             /* Convert a logical record number (e.g. Pack.Loc) into a track code */
805
806 TrackAd:    proc(Loc) returns (1 like TrackCode unaligned);
807                 dcl Loc fixed parameter,                  /* the logical address */
808                     T   fixed;                            /* the absolute track number */
809                 T = (Loc*LogRecs)/Sectors;                /* get the track number */
810                 return (Track(T/Heads,Mod(T,Heads)));     /* deliver a code */
811             end TrackAd;
```

Format Pack -- Alternate Track Table

```
812             % page;
813
814             /* A software record of bad tracks and their alternates is kept on record 3.
815                This has one entry for each track in the alternate track area, 3x19=57
816                tracks. Each entry is one word. The lower half has the code for the alternate
817                track. The upper half has the code for the bad track to which this is
818                alternate, or FreeTrk if this alternate is unused. Bad tracks in the
819                alternate area are recorded as being self-alternate.
820                This format was not my idea. It is used by RSIP. */
821
822             dcl 1 ATT(0:AltCyls*Heads-1) based (Addr(Track0(3))),/* the ATT */
823                   2 Trk like TrackCode unaligned,          /* bad track */
824                   2 Alt like TrackCode unaligned;          /* its alternate */
825
826             /* This is a special code for an unused table entry (ATT(A).Trk). The value used
827                depends on the pack type. See the Pack procedure for details. */
828
829             dcl 1 FreeTrk like TrackCode unaligned;        /* kept here */
830
831             /* ClearATT: sets up an empty alternate track table. */
832
833 ClearATT:   proc;
834                 dcl Cyl fixed,                             /* alternate cylinder number */
835                     Hd  fixed,                             /* alternate head number (some of us have several) */
836                     A   fixed;                             /* index into ATT */
837                 A = LBound(ATT,1);                         /* start of table */
838                 do Cyl = UseCyls to UseCyls+AltCyls-1;     /* run through the alternate cylinders */
839                     do Hd = 0 to Heads-1;                  /* run through the heads */
840                         ATT(A).Trk = FreeTrk; ATT(A).Alt = Track(Cyl,Hd);/* make an empty */
841                         A = A+1;                           /* progress */
842                 end; end;
843             end ClearATT;                                  /* cleared */
```

Format Pack -- Alternate Track Table

```
844             % page;
845
846             /* DumpATT: lists the alternate track table, printing tracks in the standard
847                format. */
848
849 DumpATT:    proc;
850                 dcl A fixed,                                /* index into ATT */
851                     E boolean;                              /* true if there are no bad tracks */
852                 put skip                                    /* title */
853                     line ('bad tracks (alternates)');
854                 E = true;                                   /* nothing seen yet */
855                 do A = LBound(ATT,1) to HBound(ATT,1);      /* run through the array */
856                     if String(ATT(A).Trk) ~= String(FreeTrk)/* is this space occupied ? */
857                     then do;                                /* found a bad track */
858                         E = false;                          /* table is not empty */
859                         if String(ATT(A).Trk) ~= String(ATT(A).ALt)/* is this self-alt? */
860                         then put list (TrackName(ATT(A).Trk) || ' (' || TrackName(ATT(A).Alt) || ')');
861                         else put list (TrackName(ATT(A).Trk));
862                     end;
863                 end;
864                 if E then put list ('(none)');              /* don't puzzle the op */
865                 put skip;
866             end DumpATT;
867
868             /* GoodATT: checks the ATT's format. This check should catch most garbage
869                ATTs. */
870
871 GoodATT:    proc returns (boolean);
872                 dcl Cyl fixed,                              /* current cylinder number */
873                     Hd  fixed,                              /* current head number */
874                     A   fixed;                              /* index into ATT */
875                 A = LBound(ATT,1);                          /* start table check */
876                 do Cyl = UseCyls to UseCyls+AltCyls-1;      /* check cylinders */
877                     do Hd = 0 to Heads-1;                   /* check heads */
878                         if ATT(A).Alt.Cyl ~= Cyl or ATT(A).Alt.Hd ~= Hd/* is it OK? */
879                         then return (false);                /* break out with bad news */
880                         A = A+1;                            /* next entry */
881                 end; end;
882                 return (true);                              /* made it through the check */
883             end GoodATT;
884
885             /* call this iff about to use the ATT */
886
887 ATTCheck:   proc;
888                 if not Pack.HasATT then call ErrExit('there is no Alternate Track Table');
889             end ATTCheck;
```

Format Pack -- Alternate Track Table

```
890             % page;
891
892             /* AddATT: adds a track (specified by a track code) to the list of bad tracks in
893                the ATT. A message is printed if the entry is a duplicate. ErrExit is called
894                if the table overflows. */
895
896 AddATT:     proc(T);
897                 dcl 1 T like TrackCode unaligned parameter,/* track code */
898                     A fixed;                               /* index into ATT */
899                 if T.Cyl < UseCyls                         /* is this a data track */
900                 then do;                                   /* yes */
901                     if SearchATT(T,A)                      /* if the track is in the ATT */
902                     then do; call Dupl(T); return; end;    /* then don't add it again */
903                     call AddUse(T);                        /* not there, add it */
904                 end;
905                 else if T.Cyl < UseCyls+AltCyls then call AddAlt(T);/*  remove from alternates */
906                 else signal error;                         /* T&D or other */
907
908                 /* AddUse: adds another data track to the table. We find a free alternate
909                    (the entry is <0) and give it the track. */
910
911 AddUse:             proc(T);
912                     dcl 1 T like TrackCode unaligned parameter,/* bad track */
913                         A fixed;                           /* index into the track table */
914                     if SearchATT(FreeTrk,A)                /* look for a free track */
915                     then ATT(A).Trk = T;                   /* and fill it in */
916                     else call ErrExit('too many bad tracks');/* if there were no free slots */
917                 end AddUse;
918
919                 /* AddAlt: add a track in the alternate track area to the table. This involves
920                    finding another alternate for the track to which this is alternate. */
921
922 AddAlt:             proc(T);
923                     dcl 1 T like TrackCode unaligned parameter,/* bad track */
924                         A fixed;                           /* corresponding index of ATT */
925                     A = (T.Cyl-UseCyls)*Heads+T.Hd;        /* figure offset in table */
926                     if String(ATT(A).Trk) ^= String(FreeTrk)/* is this place used? */
927                     then do;                               /* yes */
928                         if String(ATT(A).Trk) = String(T) /* is that us? */
929                         then do; call Dupl(T); end;        /* that's a duplicate */
930                         else call AddUse(ATT(A).Trk);      /* otherwise find the occupant a new home */
931                     end;
932                     ATT(A).Trk = ATT(A).Alt;               /* make it self-alternate */
933                 end AddAlt;
934
935                 /* Dupl: prints a message about duplicate entries. No unusual exit is taken. */
936
937 Dupl:               proc(T);
938                     dcl 1 T like TrackCode unaligned parameter;/* bad track */
939                     call Error(TrackName(T)||' is already listed');/* that's all */
940                 end Dupl;
941             end AddATT;
```

Format Pack -- Alternate Track Table

```
942
943             /* This searches the table for a track. Iff found it returns TRUE and leaves the
944                index in the second argument. It can e used to search for free alternates. */
945
946 SearchATT:  proc(Trk,Find) returns (boolean);
947                 dcl 1 Trk like TrackCode unaligned parameter,/* to search for */
948                     Find  fixed result parameter,          /* index where found */
949
950                     1 T   like TrackCode unal,             /* local copy of track */
951                     A     fixed;
952
953                 T = Trk;                                   /* get a handier copy of the track code */
954                 do A = LBound(ATT,1) to HBound(ATT,1);     /* scan */
955                     if String(ATT(A).Trk) = String(T)      /* find it? */
956                     then do;                               /* found it ? */
957                         Find = A;                          /* note where */
958                         return (true);                     /* note that */
959                     end;
960                 end;
961                 return (false);                            /* not found */
962
963             end SearchATT;
```

Format Pack -- Ask User

```
964             % subtitle 'Format Pack -- Ask User';
965
966             /* Ask the operator to name any additional bad tracks. Tracks are named in the
967                the usual format. The list of tracks is ended by an empty line. */
968
969  AskOp:     proc;
970
971                 dcl Line     char var,                  /* line from the operator */
972                     Cyl      fixed,                     /* cylinder number from the operator */
973                     Hd       fixed,                     /* head                              */
974                     Sect     fixed;                     /* sector                            */
975
976                 /* Ask the operator (the user) for the names of additional bad tracks. Continue
977                    to do so until (s)he enters an empty line */
978
979                 Line = Prompt('track?  ');              /* get a line */
980                 do while (Length(Line)>0);              /* until empty line */
981                     if MakeAddress(Line,Cyl,Hd,Sect)    /* attempt conversion */
982                     then do;                            /* if it succeeds... */
983                         if 0 > Cyl or Cyl ~< UseCyls+AltCyls or 0 > Hd or Hd ~< Heads
984                         then call Error('track out of range');
985                         else call AddATT(Track(Cyl,Hd)); /* add to list */
986                     end;
987                     else call Error('incorrect format for track. Try CCC/HH');
988
989                     Line = Prompt('track?  ');          /* get the next line */
990                 end;
991
992             end AskOp;
```

Format Pack -- Write Track Headers

```
993             % subtitle 'Format Pack -- Write Track Headers';
994
995             /* This procedure formats all user tracks as primary track, good. All alternate
996                tracks are formatted as alternate track, good. T&D cylinders are formatted
997                as good.  This is the only place we touch them.
998                 BEWARE: this *destroys* all information on the pack. */
999
1000 Format:     proc;
1001
1002                 dcl(C,                                  /* cylinder number */
1003                     H)fixed,                            /* head number */
1004                     S fixed,                            /* current file pointer */
1005                     D fixed;                            /* file pointer increment */
1006
1007                 String(TH) = ''b;                       /* zero out the trackheader */
1008                 D = Sectors/LogRecs;                    /* file pointer increment for next track */
1009                 S = 0;                                  /* start with 0 */
1010
1011                 TH.HA.TI, TH.ROCount.TI = TI$GP;        /* format as good primary */
1012                 do C = 0 to UseCyls-1;                  /* format user cylinders */
1013                     do H = 0 to Heads-1;
1014                         TH.HA.Cyl,TH.ROCount.Cyl = C;   /* fill in track header for this track */
1015                         TH.HA.Hd, TH.ROCount.Hd = H;
1016                         call FileP(S);                  /* set up seek */
1017                         call PIOO(d$formGP,THDCW);      /* format */
1018                         S = S+D;                        /* next file pointer */
1019                 end; end;
1020
1021                 TH.HA.TI,TH.ROCount.TI = TI$GA;         /* format as good alternate */
1022                 do C = UseCyls to UseCyls+AltCyls-1;    /* format alternate tracks */
1023                     do H = 0 to Heads-1;
1024                         TH.HA.Cyl,TH.ROCount.Cyl = C;   /* fill in header */
1025                         TH.HA.Hd, TH.ROCount.Hd  = H;
1026                         call FileP(S);                  /* set up seek */
1027                         call PIOO(d$formGA,THDCW);      /* format */
1028                         S = S+D;                        /* next */
1029                 end; end;
1030
1031                 TH.HA.TI,TH.ROCount.TI = TI$GP;         /* format T&D as good */
1032                 do C = UseCyls+AltCyls to Cyls-1;       /* rest is T&D */
1033                     do H = 0 to Heads-1;
1034                         TH.HA.Cyl,TH.ROCount.Cyl = C;   /* fill in header */
1035                         TH.HA.Hd, TH.ROCount.Hd  = H;
1036                         call FileP(S);                  /* set up seek */
1037                         call PIOO(d$formGP,THDCW);      /* format */
1038                         S = S+D;                        /* next */
1039                 end; end;
1040             end Format;
```

Format Pack -- Write Track Headers

```
1041                % page;
1042
1043                /* This procedure reformats all the bad tracks listed in the ATT. Bad data
1044                   tracks are formatted as defective, with alternate. Bad alternate tracks are
1045                   formatted as defective without alternate. */
1046
1047  ReFormat:     proc;
1048                    dcl A   fixed,                          /* index in ATT */
1049                        Cyl fixed,                          /* bad track's cylinder */
1050                        Hd  fixed,                          /* bad track's head */
1051                        1 Trk like TrackCode unal,          /* bad track */
1052                        1 Alt like TrackCode unal;          /* alternate */
1053
1054                    do A = LBound(ATT,1) to HBound(ATT,1);  /* run through the ATT */
1055                        Trk = ATT(A).Trk;                   /* bad track address */
1056                        if String(Trk) ~= String(FreeTrk)   /* if this isn't an empty entry */
1057                        then do;                            /* reformat the track */
1058                            Cyl = Trk.Cyl; Hd = Trk.Hd;     /* get the track cylinder & head */
1059                            TH.HA.Cyl = Cyl; TH.HA.Hd = Hd; /* fill in header address */
1060                            Alt = ATT(A).Alt;               /* get the alternate */
1061                            TH.ROCount.Cyl = Alt.Cyl; TH.ROCount.Hd = Alt.Hd;/* fill in R0 with address */
1062                            call FileP(FileAd(Cyl,Hd,0));   /* seek */
1063                            if String(Trk) = String(Alt)    /* is it self-alternate? */
1064                            then do;                        /* then it's a bad alternate */
1065                                TH.HA.TI, TH.ROCount.TI = TI$BN;/* bad, alternateless */
1066                                call PIO0(d$formBN,THDCW);  /* format */
1067                            end;
1068                            else do;                        /* is bad data track */
1069                                TH.HA.TI, TH.ROCount.TI = TI$BA;/* bad, alternated */
1070                                call PIO0(d$formBA,THDCW);  /* format */
1071                            end;
1072                        end;
1073                end;
1074
1075            end ReFormat;
```

Test

```
1076                % subtitle 'Test';
1077
1078    Test:       proc returns (boolean);
1079
1080                    % dcl Trials lit '4';                   /* try up to 4 times */
1081
1082                    dcl Buffer  (0:Sectors-1) char (4*RecSize),/* a track of sectors */
1083                        1 BDCW  like DCW unaligned,         /* a DCW for same */
1084                        Fails   fixed,                      /* number of bad tracks seen */
1085
1086                        Cyl fixed,                          /* cylinder number */
1087                        Hd  fixed,                          /* surface number */
1088
1089                        Try  fixed,                         /* test counter */
1090                        Errs fixed,                         /* operation error counter */
1091
1092                        St (0:Trials-1) bit (36) unaligned, /* saved bad status returns */
1093                        I  fixed;
1094
1095                    Fails = 0;                              /* no evil seen yet */
1096                    BDCW = IOTD(WAddr(Buffer),WLen(Buffer)); /* set up the DCW */
1097                    call MakePattern;                       /* set up the initial pattern */
1098
1099                do Cyl = 0 to UseCyls-1;                    /* for each cylinder */
1100                    do Hd  = 0 to Heads-1;                  /* for each surface */
1101
1102                        call FileP(FileAd(Cyl,Hd,0));       /* set the pointer to this track */
1103                        Errs = 0;                           /* nothing wrong yet */
1104
1105                        do Try = 0 to Trials-1;             /* do everything several times */
1106                            if not TestOp(d$write)          /* try to write it */
1107                            then Errs = Errs+1;
1108                            else do;                        /* if it wrote */
1109                                if not TestOp(d$read)       /* get it back */
1110                                then do;
1111                                    call MakePattern;       /* fix it up if it was messed up */
1112                                    Errs = Errs+1;
1113                        end; end; end;
1114
1115                        if Errs>0
1116                        then do;                            /* something to report */
1117                            if Fails = 0 then put skip list ('track','status');/* head */
1118                            Fails = Fails+1;
1119                            put skip list (TrackName(Track(Cyl,Hd)));
1120                            do I = 0 to Errs-1;             /* list the statii */
1121                                put list (SubStr(Octal(St(I)),7,6));
1122                        end; end;
1123
1124                end; end;
1125
1126                put skip;                                   /* finish any output */
1127                return (Fails=0);                           /* tell about failures */
```

Test

```
1128
1129
1130  TestOp:          proc (Op) returns (boolean);
1131
1132                       dcl Op     fixed parameter,          /* operation code */
1133                           1 S    like PIOStatus unaligned, /* PIO status */
1134                           Retry fixed;                     /* retry counter */
1135
1136                       do Retry = 0 to 4-1;                 /* try several times */
1137
1138                           S = PIO(Pack.File,Op,BDCW);      /* issue operation */
1139                           St(Errs) = SubStr(String(S),1,36);/* save status in case */
1140
1141                           if S.Exec ^= 000b3 and           /* OK */
1142                              S.Exec ^= 001b3 and           /* SFE */
1143                              S.Exec ^= 400b3 and           /* RERR */
1144                              S.Exec ^= 420b3               /* UERR */
1145                           then go to Fail;                 /* should be one of those status */
1146                           else do;                         /* it's a PIO status */
1147                                if S.Off                    /* powered off? */
1148                                then go to Fail;
1149                                else do;                    /* IO error */
1150
1151                                     if   S.Major = 00b3
1152                                     then do;               /* channel ready */
1153                                          if (S.Minor&'010011'b)^=''b/* auto retry or EDAC? */
1154                                          then return (false);  /* no good */
1155                                          else return (true);   /* no substatus is OK */
1156                                     end;
1157                                     elif S.Major = 03b3
1158                                     then do;               /* data alert: */
1159                                          if   (S.Minor&'000011'b)^=''b
1160                                          then;             /* transmission parity or transfer timing; retry */
1161                                          elif (S.Minor&'000100'b)^=''b/* invalid seek? */
1162                                          then go to Fail;  /* bad nuze */
1163                                          else return (false);  /* header verif., check char., or comp. fail */
1164                                     end;
1165                                     elif S.Major = 07b3
1166                                     then;                  /* software timeout; retry */
1167                                     elif S.Major = 13b3
1168                                     then do;               /* MPC data alert: */
1169                                          if S.Minor = '000100'b/* Byte locked out */
1170                                          then go to Fail;  /* why? */
1171                                          else return (false);  /* was bad */
1172                                     end;
1173                                     else go to Fail;       /* unreasonable status */
1174
1175                           end; end;
1176
1177                       end;                                 /* end retry loop */
1178
1179                       return (false);                      /* too many retries */
```

Test

```
1180  Fail:               ;                                   /* come here on unexpected status */
1181                      call ErrExit('status '||Octal(SubStr(String(S),1,36)));/* bad nuze */
1182
1183                  end TestOp;
1184
1185                  /* Set up the test pattern. We just write all records with a pattern of 6's */
1186
1187  MakePattern:    proc;
1188                      dcl Pattern(0:4*RecSize-1) char static init ((4*RecSize)Chr(666b3));/* beastlie test case */
1189                      dcl R fixed;                        /* record counter */
1190
1191                      do R = 0 to Sectors-1;              /* fill in all sectors */
1192                          Buffer(R) = String(Pattern);    /* fill in the pattern */
1193                      end;
1194
1195                  end MakePattern;
1196
1197              end Test;
```

Command Table

```
1198              % subtitle 'Command Table';
1199
1200              dcl CommandTable(14) char (12) static init (  /* what's defined */
1201                     'HELP',                                /* obvious */
1202                     'REFORMAT',                            /* reformat a formatted pack */
1203                     'FORMAT',                              /* format a fresh pack */
1204                     'ALTERNATE',                           /* give a track an alternate */
1205                     'TEST',                                /* test a pack */
1206                     'EXIT',                                /* _ */
1207                     'PACK',                                /* attach a pack */
1208                     'MAP',                                 /* dump the TI bits */
1209                     'LABEL',                               /* list the label */
1210                     'SET',                                 /* set the file pointer */
1211                     'TRACKHEADER',                         /* read the track header */
1212                     'TI',                                  /* rewrite the header */
1213                     'DISABLE',                             /* disable EXEC error recovery (initial state, */
1214                     'ENABLE');                             /*  enable EXEC error recovery */
1215
1216              /* This procedure is called by the command scanner when it has parsed a command.
1217                 It is called with the index of the command in the command table and is
1218                 responsible for passing the buck. */
1219
1220  Dispatch:   proc (CN);
1221                 dcl CN fixed parameter;                    /* the command to execute */
1222
1223                 do case (CN);                              /* branch */
1224                      call _Help;                           /* help the user */
1225                      call _ReFormat;                       /* reformat a pack */
1226                      call _Format;                         /* format a pack */
1227                      call _Alternate;                      /* give a track an alternate */
1228                      call _Test;                           /* check it out */
1229                      call _Exit;                           /* leave! */
1230                      call _Pack;                           /* attach a pack */
1231                      call _Map;                            /* decode the TI bits on the whole pack */
1232                      call _Label;                          /* dump the pack label */
1233                      call _Set;                            /* set the seek address */
1234                      call _TrackHeader;                    /* read the header */
1235                      call _TI;                             /* rewrite the TI bits */
1236                      call _Disable;                        /* disable EXEC error recovery */
1237                      call _Enable;                         /*  enable EXEC error recovery */
1238                 end;
1239
1240           end Dispatch;
```

Command Scan

```
1241            % subtitle 'Command Scan';
1242
1243            /* CommandLoop: this and associated procedures provide a command scanner.
1244               Command lines are prompted with an octathorpe (#). Commands may be
1245               abbreviated to any degree as long as they are unambiguous. See the Lookup
1246               procedure for details on abbreviations.
1247
1248               Several commands can be given on a line if they are seperated by octathorpes
1249               (#). The latest command can be repeated with an ampersand (&).
1250
1251               Arguments to a command follow the command, and are seperated from each other
1252               by spaces.
1253
1254               If an error occurs processing the command, call the procedure ErrExit with an
1255               error message. It will print the message and abort the current command line.
1256               It will not return.
1257
1258               CommandTable must be an array of command names. The LBound should be 1.
1259               Names should be char (n) nonvarying, and should be in uppercase.
1260
1261               Dispatch(n) is called with the subscript in CommandTable of the command to
1262               execute. */
1263
1264            dcl CmdLine  char var,                  /* the rest of the current command line */
1265                CmdError condition,                 /* signal to abandon command */
1266                CmdNum   fixed;                     /* the current command number (#n) */
1267
1268  CommandLoop:proc;
1269
1270                dcl
1271
1272                    Code    fixed,                  /* the code for the current command */
1273                    Letters char (53) static init   /* the legal command characters */
1274                            ('ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz_');
```

Command Scan

```
1275                    % page;
1276
1277                    on condition (CmdError) go to ReStart;    /* trap errors here */
1278
1279    ReStart:        Code = 0;                                 /* remember nothing */
1280
1281                    do indefinitely;                          /* command loop */
1282
1283                        CmdLine = Prompt('#'); CmdNum = 0;    /* start a new line */
1284
1285                        do while (Length(CmdLine) > 0);       /* process the line */
1286
1287                            if SubStr(CmdLine,1,1) = '#'
1288                            then CmdLine = SubStr(CmdLine,2); /* drop command delimiter */
1289                            else do;
1290                                CmdNum = CmdNum+1;            /* next number (for error messages) */
1291                                if SubStr(CmdLine,1,1) = '&'  /* "&" convention? */
1292                                then do;                      /* means repeat previous command */
1293                                    if Code = 0
1294                                    then call ErrExit('undefined "&"');/* no previous command */
1295                                    else CmdLine = SubStr(CmdLine,2);/* previous command, drop "&" */
1296                                end;
1297                                else Code = Lookup(Upper(Break(CmdLine,/* pop off the command, */
1298                                                               Verify(CmdLine,Letters))),/* standardize, */
1299                                                   CommandTable(*));/* & look it up */
1300                                if Code = 0 then signal condition (CmdError);/* couldn't look it up */
1301
1302
1303                                call Drop(CmdLine,Verify(CmdLine,' '));/* drop spaces after command */
1304
1305                                call Dispatch(Code);          /* do the thing */
1306
1307                                call Drop(CmdLine,Search(CmdLine,'#&'));/* get to next command */
1308                            end;
1309                        end;
1310                    end;
1311
1312            end CommandLoop;
```

Command Scan -- Errors

```
1313              % subtitle 'Command Scan -- Errors';
1314
1315              /* Error: prints its argument as an error message & returns */
1316
1317  Error:      proc (Mess);
1318                  dcl Mess     char var parameter;          /* the error message */
1319                  put skip                                  /* print it */
1320                      line ('!!! #',CmdNum,') ERROR: ',Mess);
1321              end Error;
1322
1323              /* ErrExit: prints its argument as an error message & escapes up
1324                 to the command loop, aborting the current command line. */
1325
1326  ErrExit:    proc (Mess);
1327                  dcl Mess     char var parameter;          /* parting words */
1328                  call Error(Mess);                         /* issue as standard error message */
1329                  signal condition (CmdError);              /* beam us up, Scotty */
1330              end ErrExit;
```

Command Scan -- Input

```
1331              % subtitle 'Command Scan -- Input';
1332
1333              /* Prompt: prints its argument as a prompt, then returns the next line
1334                 from the terminal (SysIn) as a string.
1335
1336                 The prompt is printed after a newline, but is not followed by one. */
1337
1338  Prompt:     proc (Prompt) returns (char var);
1339                  dcl Prompt char (*) parameter,          /* prompt: this is usually a const. */
1340
1341                      Line   char var;                    /* get the line into here */
1342
1343                  put skip                                /* prompt the user */
1344                      edit (Prompt) (a);
1345                  get line (Line);                        /* get the response */
1346                  return (Line);                          /* deliver */
1347
1348              end Prompt;
```

Command Scan -- Lookup

```
1349              % subtitle 'Command Scan -- Lookup';
1350
1351              /* Lookup:looks up a string in a table. This is intended as a command
1352                 parser. The table is searched for a match and the subscript of the match
1353                 is returned.
1354
1355                 The match is chosen according to the following rules:
1356                      1) A table entry identical to the string is an exact
1357                         match, and has priority. Failing that,
1358                      2) if the string is an unambiguous abbreviation of
1359                         a table entry then that is a match. An abbreviation
1360                         is an initial segment of an entry. An abbreviation
1361                         is unambiguous if it abbreviates only one entry.
1362                      3) If both 1) and 2) fail then an error message is printed
1363                         (via the Error() procedure) and the result is 0.
1364
1365                 Thus, commands may be abbreviated as long as they are unambiguous.
1366                 If a command is ambiguous the error message lists the possibilities.
1367                 In particular, unless it is defined, the empty string is completely
1368                 ambiguous. */
1369
1370  (noSubscriptRange,noStringRange):
1371  Lookup:     proc (Given,Table) returns (fixed);
1372
1373              /* parameters */
1374
1375              dcl Given    char var parameter,          /* the string to look up */
1376                  Table(*) char (*) parameter unaligned,/* the table to look in */
1377
1378              /* local */
1379
1380                  I        fixed,                      /* subscript into Table */
1381                  Match    fixed,                      /* subscript of last match, or 0 */
1382                  Unique   boolean,                    /* only one match found */
1383                  GLen     fixed,                      /* length of search string */
1384                  TLen     fixed;                      /* width of each entry in Table(*) */
```

Command Scan -- Lookup

```
1385                % page;
1386
1387                Unique = true;                          /* assume initial uniqueness */
1388                TLen = Length(Table(LBound(Table,1)));  /* width of arbitrary table entry */
1389                GLen = Length(Given);                   /* length of given string */
1390                Match = 0;                              /* no match yet */
1391
1392                if GLen ~> TLen                         /* if there's hope of a match */
1393                then do I = LBound(Table,1) to HBound(Table,1);/* search table */
1394                    if Given = SubStr(Table(I),1,GLen)  /* partial match? */
1395                    then do;                            /* yes, this is a possibility */
1396                        if Match ~= 0 then Unique = false;/* remember whether this was unique */
1397                        Match = I;                      /* remember this match */
1398                        if Table(Match) = Given         /* does it match exactly? */
1399                        then return (Match);            /* then that's our man! */
1400                    end;
1401                end;
1402
1403                if Match = 0                            /* anything seen? */
1404                then call Error(Given||' is not defined!');
1405                elif Unique                             /* is this ambiguous? */
1406                then return (Match);                    /* if not then we're done */
1407                else do;                                /* tutor the user a little */
1408                    call Error(Given||' is ambiguous. It could be any of:');/* warn what's coming */
1409                    do I = LBound(Table,1) to HBound(Table,1);/* list all matches */
1410                        if SubStr(Table(I),1,GLen) = Given/* if it was possible */
1411                        then put line (Table(I));       /* then list it */
1412                    end;
1413                end;
1414                return (0);                             /* in case we fall through, return 0 */
1415
1416            end Lookup;
```

Command Scan -- Arguments

```
1417              % subtitle 'Command Scan -- Arguments';
1418
1419              /* ArgLeft: tells whether there are any further arguments to this command */
1420
1421  ArgLeft:    proc returns (boolean);
1422                  if Length(CmdLine) = 0
1423                  then return (false);                          /* nothing left at all */
1424                  else return (Index('#&',SubStr(CmdLine,1,1)) = 0);/* or starts new command */
1425              end ArgLeft;
1426
1427              /* Arg: pops the next argument off the current command line. The argument
1428                 is delimited by spaces. It is not uppercased. If no more arguments
1429                 were supplied then the empty string is returned. */
1430
1431  Arg:        proc returns (char var);
1432                  dcl R        char var;                        /* the result to be */
1433
1434                  R = Break(CmdLine,Search(CmdLine,'#& '));  /* break at the next space or delim. */
1435                  call Drop(CmdLine,Verify(CmdLine,' '));    /* then throw away trailing spaces */
1436                  return (R);                                   /* deliver */
1437              end Arg;
1438
1439              /* ArgCheck: makes sure that there aren't too many arguments to a command. If
1440                 there are then ErrExit is called. */
1441
1442  ArgCheck:   proc;
1443                  if ArgLeft() then call ErrExit('too many arguments!');
1444              end ArgCheck;
```

Argument Scan

```
1445              % subtitle 'Argument Scan';
1446
1447              /* ArgAddress: converts the next argument to an address, setting the result
1448                 parameters to the cylinder, head, and sector.
1449
1450                 If the argument isn't in a legitimate format then ErrExit() is called. */
1451
1452  ArgAddress: proc (Cyl,Hd,Sect);
1453                  dcl (Cyl,Hd,Sect) fixed parameter;        /* the results */
1454                  if not MakeAddress(Arg(),Cyl,Hd,Sect)     /* try the conversion */
1455                  then call ErrExit('bad format for address.');
1456              end ArgAddress;
1457
1458              /* ArgTI: converts the next argument to a TI value. If it doesn't conform then
1459                 ErrExit() is called.
1460
1461                 TI bits are specified as two binary digits. */
1462
1463  ArgTI:      proc returns (bit(2) unaligned);
1464                  dcl A char var,                           /* the argument string in ASCII */
1465                      B bit (2);                            /* the bits */
1466                  A = Arg();                                /* get an argument */
1467                  if Length(A) ^= 2 or Verify(A,'01') ^= 0  /* check format */
1468                  then call ErrExit('incorrect format for TI bits.');
1469                  else get edit (B) (b(2)) string (A);      /* if OK then convert */
1470                  return (B);                               /* deliver */
1471              end ArgTI;
1472
1473              /* ArgDev: gets a device name as an argument. This is uppercased. */
1474
1475  ArgDev:     proc returns (char var); return (Upper(Arg())); end ArgDev;
```

Argument Scan

```
1476                % page;
1477
1478                /* OptAddress: if an argument was supplied then it is converted to an address
1479                   and the current pointer on the pack is set there. Otherwise there should be
1480                   no arguments */
1481
1482  OptAddress: proc;
1483                    dcl (Cyl,Hd,Sect) fixed;                    /* as specified */
1484                    if ArgLeft()                                /* if anything was given */
1485                    then do;                                    /* then use it */
1486                        call ArgAddress(Cyl,Hd,Sect);           /* as an address */
1487                        call FileP(FileAd(Cyl,Hd,Sect));        /* for the pack */
1488                        call ArgCheck;                          /* nothing else */
1489                    end;
1490                end OptAddress;
1491
1492                /* OptDev: if an argument was supplied it is opened for the current pack.
1493                   There should be no other arguments */
1494
1495  OptDev:       proc;
1496                    if ArgLeft()                                /* if there was anything */
1497                    then do;
1498                        call GetPack(ArgDev());                 /* use it */
1499                        call ArgCheck;
1500                    end;                                        /* accept no more */
1501                end OptDev;
```

Commands -- Help

```
1502            % subtitle 'Commands -- Help';
1503
1504            /*        HELP
1505
1506               The user needs help. Tell him what we know by listing a file on the TTY.
1507               Gives up if the file is not found. No big deal. */
1508
1509  _Help:    proc;
1510                dcl HFile int file constant,                /* the help file */
1511                    Line  char var,                         /* a line therefrom */
1512                    HFileName char (25) static init (':smsyscat:simcatl:packfmt');/* help? file */
1513
1514                on UndefinedFile(HFile) call ErrExit('can''t open '||HFileName);/* give up easy */
1515
1516                open file (HFile) title (HFileName) stream input;/* get the file */
1517
1518                do while (More(HFile));                     /* copy it out */
1519                    get line (Line) file (HFile); put line (Line);
1520                end;
1521
1522                close file (HFile);                         /* clean up */
1523
1524            end _Help;
```

Commands -- Exit, Enable, Disable

```
1525              % subtitle 'Commands -- Exit, Enable, Disable';
1526
1527              /*        EXIT
1528
1529                 cleans up loose ends and terminates. */
1530
1531  _Exit:      proc;
1532
1533                  call ArgCheck;                          /* accept no arguments */
1534                  call DropPack;                          /* clean up */
1535                  close file (PerCat);
1536                  stop;
1537
1538              end _Exit;
1539
1540              /*        ENABLE
1541
1542                 enables Executive error recovery on the current drive. This is done anyway
1543                 when a drive is closed. */
1544
1545  _Enable:    proc;
1546
1547                  call ArgCheck;                          /* accept no arguments */
1548                  call PackCheck;                         /* make sure we've something */
1549                  call Drive(Pack.File,d$recov);          /* drive */
1550
1551              end _Enable;
1552
1553              /*        DISABLE
1554
1555                 disables Executive error recovery on the current device. This is the
1556                 initial state. */
1557
1558  _Disable:   proc;
1559
1560                  call ArgCheck;                          /* accept no arguments */
1561                  call PackCheck;                         /* make sure we've something */
1562                  call Drive(Pack.File,d$norcv);
1563
1564              end _Disable;
```

Commands -- Set, Pack

```
1565                % subtitle 'Commands -- Set, Pack';
1566
1567                /*          SET <address>
1568
1569                   Sets the pointer on the current pack to the <address>. The current logical
1570                   record is the one beginning at that address. The current track is the one
1571                   containing the address.
1572
1573                   Note that this address *must* be aligned on a logical record boundary.
1574
1575                   The argument is not optional. */
1576
1577  _Set:         proc;
1578
1579                    call PackCheck;                          /* make sure we've something */
1580                    if ArgLeft()                             /* if argument was supplied */
1581                    then call OptAddress;                    /* then treat like optional address */
1582                    else call ErrExit('must supply <address>');/* just not optional */
1583
1584                end _Set;
1585
1586                /*          PACK <device name>
1587
1588                   Open a device as the current pack, but do nothing with it. This is useful
1589                   when you intend to do something besides (re)format it. */
1590
1591  _Pack:        proc;
1592
1593                    if ArgLeft()                             /* if argument was supplied */
1594                    then call OptDev;                        /* treat like optional pack name */
1595                    else call ErrExit('must supply <device name>');/* just not optional */
1596
1597                end _Pack;
```

Commands -- Test

```
1598            % subtitle 'Commands -- Test';
1599
1600            /*        TEST [<device name>]
1601
1602               tests the pack. Each track is written with a pattern, then reread. Bad
1603               status returns are printed, but nothing is done about them.
1604               All data except the label area are destroyed.
1605               If a device name is given that becomes the current pack. */
1606
1607  _Test:    proc;
1608
1609                call OptDev;                          /* get the pack */
1610                call PackCheck;                       /* got it? */
1611
1612                if Test()                             /* check it out */
1613                then put skip
1614                         line ('no errors');
1615
1616                call WriteTrack0;                     /* put back the label we just trashed */
1617
1618            end _Test;
```

Commands -- TrackHeader

```
1619              % subtitle 'Commands -- TrackHeader';
1620
1621              /*        TRACKHEADER [<address>]
1622
1623                 dump the current trackheader. This is in a bizarre form. The header address
1624                 and ROCount address are dumped as track addresses. The TI bits are dumped
1625                 in binary. The entire track header is then dumped in hex as a series of
1626                 eight-bit bytes.
1627
1628                 If an address is supplied that becomes the current address */
1629
1630  _TrackHeader:proc;
1631
1632                  call PackCheck;                          /* make sure we've something */
1633                  call OptAddress;                         /* optional address */
1634
1635                  call PIOO(d$rhead,THDCW);                /* issue the IO call */
1636
1637                  put skip
1638                      edit ('HA     ',' address: ',HAName()) (a)
1639                           (' TI: ',TH.HA.TI) (a,b1(2));
1640                  put skip
1641                      edit ('ROCount',' address: ',TrackName(Track((TH.ROCount.Cyl),
1642                                                              (TH.ROCount.Hd)))) (a)
1643                           (' TI: ',TH.ROCount.Flag.TI) (a,b1(2));
1644
1645              dcl Alias(0:21) bit (8) unaligned based (Addr(TH));/* a byte alias */
1646
1647              put skip
1648                  edit ('Track Header') (a)
1649                       (Alias(*)) (skip,11(x(1),b4(2)));
1650
1651          end _TrackHeader;
```

Commands -- TI

```
1652            % subtitle 'Commands -- TI';
1653
1654            /*        TI <ti bits> [<address>]
1655
1656               This command patches the track header of the current track. The track
1657               indicator is set to <ti bits>, which must be a two-bit binary number.
1658               If an <address> is supplied it is used as the record 0 count field
1659               address.
1660
1661               This command is intended for debugging use. It is possible to seriously mess
1662               things up by incorrect use of this command. */
1663
1664  _TI:      proc;
1665
1666                call PackCheck;                          /* make sure we've got something */
1667
1668                call PIOO(d$rhead,THDCW);                /* read the header */
1669
1670                TH.HA.TI,TH.ROCount.TI = ArgTI();        /* get the TI bits */
1671
1672                if ArgLeft()                             /* if the address was supplied */
1673                then begin;                              /* use the address in the record 0 count */
1674                    dcl (Cyl,Hd,Sect) fixed;             /* get some temps */
1675                    call ArgAddress(Cyl,Hd,Sect);        /* get the address */
1676                    TH.ROCount.Cyl = Cyl; TH.ROCount.Hd = Hd;/* put it where it goes */
1677                end;
1678
1679                call ArgCheck;                           /* should be no more arguments */
1680
1681                call PIOO(d$form+ShL(FixedBin(TH.HA.TI),9),/* get appropriate format drive */
1682                          THDCW);
1683
1684           end _TI;
```

Commands -- Map

```
1685                % subtitle 'Commands -- Map';
1686
1687                /*          MAP [<device name>]
1688
1689                   Print a map of the TI bits on the pack, showing which tracks are good, which
1690                   defective, which primary, and which alternate. For defective tracks with
1691                   alternates the address of the alternate is also given.
1692
1693                   The state of each track is determined by reading the track's header. This
1694                   reveals the actual state of the track, and is more reliable than the table on
1695                   track 0. However, it takes a long time (as long as FORMAT). This may be
1696                   worthwhile if a pack has been formatted before, but the tables have been
1697                   destroyed (as by DEVTEST). */
1698
1699  _Map:         proc;
1700
1701                    dcl Loc  fixed,                     /* current logical record number */
1702                        Step fixed,                     /* logical records on a track */
1703                        PLoc fixed;                     /* logical address before we began */
1704
1705                    call OptDev;                        /* optional device name */
1706                    call PackCheck;                     /* make sure this is valid */
1707
1708                PLoc = Pack.Loc;                        /* remember where we were */
1709                Step = Sectors/LogRecs;                 /* track size, in logical records */
1710                Loc = 0;                                /* start at the outside */
1711
1712
1713                do while (Loc < Range);                 /* run to the end */
1714                    call ReadTH;                        /* get the current header handy */
1715
1716                    if TH.HA.TI = TI$GP
1717                    then call Area(TI$GP,'good primary'); /* group good tracks */
1718                    elif TH.HA.TI = TI$GA
1719                    then call Area(TI$GA,'good alternate');
1720                    else do;                            /* these will be isolated instances */
1721                        if TH.HA.TI = TI$BA
1722                        then put skip                   /* bad, with alternate */
1723                                edit (HAName(),' defective track, alternate is ') (a)
1724                                    (TrackName(Track((TH.ROCount.Cyl),(TH.ROCount.Hd)))) (a);
1725                        else put skip                   /* bad, alternateless */
1726                                edit (HAName()) (a)
1727                                    ('defective track without alternate') (col(20),a);
1728                        Loc = Loc+Step;                 /* next */
1729                    end;
1730                end;
1731
1732                call FileP(PLoc);                       /* get back into known state */
```

Commands -- Map

```
1733                    % page;
1734
1735                    /* we expect that most of the tracks will be (primary,good) or (alternate,good).
1736                       This procedure dumps a series of such tracks as a range (first ... last).
1737                       Call with the TI bits to look for and the name of the kind of track. */
1738
1739    Area:           proc (TI,Name);
1740                        dcl TI   bit (2) parameter,          /* the kind of TI bits we're looking for */
1741                            Name char (*)parameter;          /* the name of this kind */
1742
1743                        put skip                             /* print starting (possibly only) in range */
1744                            edit (HAName()) (a);
1745
1746                        /* If there was an isolated track of this type then just print it. If, however,
1747                           there was more than one, print it as a range. Stop when the pack runs out
1748                           or some other TI turns up */
1749
1750                        Loc = Loc+Step;                      /* look at the next */
1751                        if Same()                            /* more of same? */
1752                        then do;                             /* there are several such here */
1753                            put edit (' ... ') (a);          /* show that a range is coming */
1754                            Loc = Loc+Step;                  /* move ahead */
1755                            do while (Same()); Loc = Loc+Step; end;/* find end of range */
1756                            put edit (TrackName(TrackAd(Loc-Step))) (a);/* print the last address before */
1757                        end;
1758                        put edit (Name) (col(20),a);         /* follow with name of area */
1759
1760                        /* this procedure determines whether the current track (indicated by Loc) exists
1761                           and if so, whether it has the same TI bits. */
1762
1763    Same:               proc returns (boolean);
1764                            if Loc = Range                   /* does it exist? */
1765                            then return (false);             /* no, difficult ontological question */
1766                            else do;                         /* yes, see if it is the same */
1767                                call ReadTH;                 /* look at the header */
1768                                return (TH.HA.TI = TI);      /* well? */
1769                            end;
1770                        end Same;
1771                    end Area;
1772
1773                    /* procedure to read the track header of the track indicated by Loc */
1774
1775    ReadTH:         proc;
1776                        call FileP(Loc); call PIOO(d$rhead,THDCW);/* read the track header */
1777                    end ReadTH;
1778
1779                end _Map;
```

Commands -- Label

```
1780              % subtitle 'Commands -- Label';
1781
1782              /*        LABEL [<device name>]
1783
1784                 reads the label and AlternateTrackTable off the pack. The label is dumped as
1785                 is. If it looked reasonable, the ATT is also dumped. */
1786
1787  _Label:     proc;
1788
1789                  call OptDev;                              /* get a optional pack name */
1790                  call PackCheck;                           /* make sure we've something to do */
1791
1792                  /* assume that there is something of a label, & dump it. */
1793
1794                  put skip                                  /* dump the label */
1795                      edit ('label type',  ACI(Label.Begin),
1796                            'site',        ACI(Label.Site),
1797                            'number',      ACI(Label.Numb),
1798                            'pack name',   ACI(SubStr(Label.Name,1,36))||ACI(SubStr(Label.Name,37,36)),
1799                            'pack type',   ACI(Label.Type),
1800                            '',            Octal(Label.Misc),
1801                            'initialized', ADate(Label.IDate),
1802                            'retain until',ADate(Label.RDate),
1803                            'sequence #',  ACI(Label.SeqNo))
1804                                                        (skip,a(25),a);
1805
1806                      call ATTCheck;                        /* don't continue without ATT */
1807                      call DumpATT;                         /* dump one */
1808
1809              end _Label;
```

Commands -- Format

```
1810            % subtitle 'Commands -- Format';
1811
1812            /*        FORMAT [<device name>]
1813
1814               Format a pack, ignoring any information that might be there. This is necessary
1815               if a pack has not been formatted before, or if the label area (Track 0) is
1816               unreadable. The user must list all bad tracks. The pack is formatted using
1817               this data and relabelled.
1818
1819               If no pack name is given then the current pack is used. */
1820
1821 _Format:   proc;
1822
1823                call OptDev;                          /* get a optional pack name */
1824
1825                call PackCheck;                       /* make sure there's something to do */
1826
1827                call ClearTrack0;                     /* clear the info */
1828                call NewLabel;                        /* get a Label */
1829                call ClearATT;                        /* no bad tracks yet */
1830                put line('what are the bad tracks?'); call AskOp;/* see if anything to add */
1831                call Format;                          /* quick T&D format */
1832                call ReFormat;                        /* take care of alternate tracks */
1833                call DumpATT;                         /* give final version */
1834
1835                call ReLabel;                         /* rewrite pack information in label */
1836
1837            end _Format;
```

Commands -- ReFormat

```
1838            % subtitle 'Commands -- ReFormat';
1839
1840            /*        REFORMAT [<device name>]
1841
1842               The reformat command reformats the current pack. This pack must be one that
1843               has already been formatted and labelled. For the user's edification
1844               the interesting data in the label is decoded and printed. The list of
1845               tracks in the ATT is assumed valid and used to construct the new ATT.
1846               The user is given an opportunity to list further bad tracks. Then the pack
1847               is reformatted and relabelled.
1848
1849               If no pack name is given then the current pack is used. */
1850
1851  _ReFormat:  proc;
1852                   dcl Line char var;                    /* place to get answer to questions */
1853
1854                   call OptDev;                          /* get a optional pack name */
1855
1856                   call PackCheck;                       /* make sure we've something to do */
1857
1858                   call ATTCheck;                        /* make sure it's got a label */
1859
1860                   put skip                              /* tell what we know */
1861                       edit ('pack ',ACI(Label.Numb),
1862                             ' last formatted on ',ADate(Label.IDate),
1863                             ' for ',ACI(Label.Site)) (a);
1864
1865              call DumpATT;                              /* tell all we know */
1866              put line('any other bad tracks?'); call AskOp;/* see what the user has to add */
1867              call Format;                               /* first a simple formatting */
1868              call ReFormat;                             /* then patch up the bad spots */
1869              call DumpATT;                              /* give final version */
1870
1871              call ReLabel;                              /* we've reformatted */
1872
1873         end _ReFormat;
```

Commands -- Alternate

```
1874              % subtitle 'Commands -- Alternate';
1875
1876              /*        ALTERNATE [<address>]
1877
1878                 Give the indicated track an alternate and update the ATT. This requires an
1879                 ATT. If the track already has an alternate, then we mark that bad and find
1880                 another.
1881
1882                 If an address is supplied it becomes the current address. */
1883
1884  _Alternate: proc;
1885
1886                  dcl 1 Trk like TrackCode unal,          /* code for the bad track */
1887                      A     fixed;                        /* ATT index of */
1888
1889                  call PackCheck;                         /* must have a pack */
1890                  call OptAddress;                        /* get the address */
1891
1892                  call ATTCheck;                          /* got to have an ATT */
1893
1894                  Trk = TrackAd(Pack.Loc);
1895                  if Trk.Cyl ~< UseCyls then call ErrExit('That''s not a data track');
1896
1897                  if SearchATT(Trk,A)                     /* already bad? */
1898                  then do;                                /* then we must re-link */
1899                      dcl PLoc fixed;                     /* loc of bad track */
1900
1901                      PLoc = Pack.Loc;                    /* save previous address */
1902                      call FileP(FileAd((ATT(A).Alt.Cyl),  /* go to bad Alt */
1903                                        (ATT(A).Alt.Hd),
1904                                        0));
1905                      call PIOO(d$rhead,THDCW);           /* read the header */
1906                      TH.HA.TI, TH.ROCount.TI = TI$BN;    /* bad, no alternate */
1907                      call PIOO(d$formBN,THDCW);          /* rewrite header */
1908                      String(ATT(A).Trk) = String(ATT(A).Alt);/* make self-alternate */
1909                      call FileP(Ploc);                   /* go back to bad track */
1910                  end;
1911
1912                  if not SearchATT(FreeTrk,A) then call ErrExit('too many bad tracks');
1913                  ATT(A).Trk = Trk;                       /* make this alternated */
1914                  call PIOO(d$rhead,THDCW);               /* get the header */
1915                  TH.HA.TI, TH.ROCount.TI = TI$BA;        /* defective, with alternate */
1916                  TH.ROCount.Cyl = ATT(A).Alt.Cyl; TH.ROCount.Hd = ATT(A).Alt.Hd;/* link to Alt */
1917                  call PIOO(d$formBA,THDCW);              /* reformat */
1918
1919                  call ReLabel;                           /* we've sort of reformatted */
1920
1921              end _Alternate;
```

Startup

```
1922            % subtitle 'Startup';
1923
1924            /* set the initial pack to an undefined file, and get a FRN for PERCAT, to
1925               make opening devices easier. */
1926
1927  Init:     proc;
1928                dcl NoPrompt ext entry(file);           /* suppress builtin prompt */
1929
1930                Pack.File = NullF();                    /* mark this as undefined. */
1931
1932                open file (PerCat)
1933                     title ('PERCAT')
1934                     env (unformatted                   /* no I/O, just the FRN */
1935                          catfrn (0)
1936                          access (411000b3));           /* open with CSR */
1937                PerCatFRN = FRN(PerCat);                /* get what we wanted */
1938
1939                call NoPrompt(SysIn);                   /* we do our own prompting */
1940
1941            end Init;
```

Driver

```
1942            % subtitle 'Driver';
1943
1944            call Init;                          /* set up */
1945
1946            call CommandLoop;                   /* start doing things */
1947
1948        end PackFmt;
```

line	name	type	attributes	references by line
1048	A	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	1054 1055 1060
951	A	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	954 955 957
924	A	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	925 926 928 930 932 932
913	A	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	914 915
898	A	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	901
874	A	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	875 878 878 880 880
850	A	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	855 856 859 859 860 860 861
836	A	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	837 840 840 841 841
626	A	CHAR(6)	AUTO INT NONVARYING UNAL VARIABLE	631 632 632 632
576	A	CHAR(1)	AUTO DIM(0:5) INT NONVARYING UNAL VARIABLE	580 581
1887	A	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	1897 1902 1903 1908 1908 1912 1913 1915 1916
1464	A	CHAR	ALIGNED AUTO INT VARIABLE VAR	1466 1467 1467 1469
221	A	FIXED BIN	ALIGNED INT PARM PREC(35) REAL VARIABLE	226 220
167	A	BIT(1)	INT MEMBER NONVARYING UNAL VARIABLE	
4	ACC	BIT(9)	INT MEMBER NONVARYING UNAL VARIABLE	
252	ACC	BIT(9)	INT MEMBER NONVARYING UNAL VARIABLE	
277	ACC	BIT(9)	INT MEMBER NONVARYING UNAL VARIABLE	
4	ACC	BIT(9)	INT MEMBER NONVARYING UNAL VARIABLE	
4	ACC	BIT(9)	INT MEMBER NONVARYING UNAL VARIABLE	
141	ACC	BIT(9)	INT MEMBER NONVARYING UNAL VARIABLE	
706	ACC	BIT(9)	INT MEMBER NONVARYING UNAL VARIABLE	
1133	ACC	BIT(9)	INT MEMBER NONVARYING UNAL VARIABLE	
563	ACI	ENTRY	CONSTANT INT RETURNS	631 747 748 1795 1796 1797 1798 1793 1799 1803 1861 1863
1083	ACTION	BIT(2)	INT MEMBER NONVARYING UNAL VARIABLE	
128	ACTION	BIT(2)	INT MEMBER NONVARYING UNAL VARIABLE	
224	ACTION	BIT(2)	INT MEMBER NONVARYING UNAL VARIABLE	228
695	ACTION	BIT(2)	INT MEMBER NONVARYING UNAL VARIABLE	
189	ACTION	BIT(2)	INT MEMBER NONVARYING UNAL VARIABLE	
275	ACTION	BIT(2)	INT MEMBER NONVARYING UNAL VARIABLE	
246	ACTION	BIT(2)	INT MEMBER NONVARYING UNAL VARIABLE	
624	ADATE	ENTRY	CONSTANT INT RETURNS	1801 1802 1862
922	ADDALT	ENTRY	CONSTANT INT	905
896	ADDATT	ENTRY	CONSTANT INT	985
	ADDR	BUILTIN	INT	8 8 8 733 822 1645
911	ADDUSE	ENTRY	CONSTANT INT	903 930
1645	ALIAS	BIT(8)	BASED DIM(0:21) INT NONVARYING UNAL VARIABLE	1649
1052	ALT	STRUCTURE	AUTO INT LIKE UNAL VARIABLE	1060 1063
824	ALT	STRUCTURE	INT LIKE MEMBER UNAL VARIABLE	840 859 860 932 1060 1908
316	ALTCYLS	LITERALLY		385 822 838 876 905 983 1022 1032
1	AND	LITERALLY		283 682 682 713 748 749 1141 1142 1143
1739	AREA	ENTRY	CONSTANT INT	1717 1719
1431	ARG	ENTRY	CONSTANT INT RETURNS	1454 1466 1475
1452	ARGADDRESS	ENTRY	CONSTANT INT	1486 1675
1442	ARGCHECK	ENTRY	CONSTANT INT	1488 1499 1533 1547 1560 1679
1475	ARGDEV	ENTRY	CONSTANT INT RETURNS	1498
1421	ARGLEFT	ENTRY	CONSTANT INT RETURNS	1443 1484 1496 1580 1593 1672
1463	ARGTI	ENTRY	CONSTANT INT RETURNS	1670

line	name	type	attributes	references by line
566	ASCII	CHAR(1)	DIM(0:63) INIT INT NONVARYING STATIC UNAL VARIABLE	580
969	ASKOP	ENTRY	CONSTANT INT	1830 1866
822	ATT	STRUCTURE	ALIGNED BASED DIM(0:56) INT VARIABLE	837 855 855 875 954 954 1054 1054
887	ATTCHECK	ENTRY	CONSTANT INT	1806 1858 1892
564	B	BIT(36)	INT NONVARYING PARM UNAL VARIABLE	579 563
1465	B	BIT(2)	AUTO INT NONVARYING UNAL VARIABLE	1469 1470
603	B	UNSIGNED	AUTO BIN DIM(1:6) INT PREC(6) REAL UNAL VARIABLE	607 609
625	B	BIT(36)	INT NONVARYING PARM UNAL VARIABLE	628 631 624
591	BCD	UNSIGNED	BIN DIM(32:95) INIT INT PREC(6) REAL STATIC UNAL VARIABLE	607
588	BCI	ENTRY	CONSTANT INT RETURNS	619 756 757 758 758 759 767 769 770
616	BDATE	ENTRY	CONSTANT INT RETURNS	768
1083	BDCW	STRUCTURE	AUTO INT LIKE UNAL VARIABLE	1096 1138
734	BEGIN	BIT(36)	INT MEMBER NONVARYING UNAL VARIABLE	747 756 1795
6	BOOLEAN	LITERALLY		302 650 704 746 851 871 946 1073 1130 1382 1421 1763
541	BREAK	ENTRY	CONSTANT INT RETURNS	1297 1434
1082	BUFFER	CHAR(256)	AUTO DIM(0:39) INT NONVARYING UNAL VARIABLE	1096 1096 1192
	BYTE	BUILTIN	INT	606
1002	C	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	1012 1014 1022 1024 1032 1034
602	C	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	606 606 606 606 607
575	C	UNSIGNED	AUTO BIN DIM(0:5) INT PREC(6) REAL UNAL VARIABLE	579 580
497	C	FIXED BIN	ALIGNED INT PARM PREC(35) REAL VARIABLE	501 496
484	C	FIXED BIN	ALIGNED INT PARM PREC(35) REAL VARIABLE	488 490 483
799	CCC	PICTURE	INT MEMBER UNAL VARIABLE	800
174	CHKCHR	BIT(6)	INT MEMBER NONVARYING UNAL VARIABLE	
	CHR	BUILTIN	INT	1188
833	CLEARATT	ENTRY	CONSTANT INT	1829
699	CLEARTRACKO	ENTRY	CONSTANT INT	1827
1265	CMDERROR	CONDITION	EXT	1277 1300 1329
1264	CMDLINE	CHAR	ALIGNED AUTO INT VARIABLE VAR	1283 1285 1287 1288 1288 1291 1295 1295 1297 1298 1303 1303 1307 1307 1422 1424 1434 1434 1435 1435
1266	CMDNUM	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	1283 1290 1290 1320
1221	CN	FIXED BIN	ALIGNED INT PARM PREC(35) REAL VARIABLE	1223 1220
1272	CODE	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	1279 1293 1297 1300 1305
446	CODE	FIXED BIN	ALIGNED INT PARM PREC(35) REAL VARIABLE	454 445
1268	COMMANDLOOP	ENTRY	CONSTANT INT	1946
1200	COMMANDTABLE	CHAR(12)	DIM(1:14) INIT INT NONVARYING STATIC UNAL VARIABLE	1299
1083	COUNT	UNSIGNED	BIN INT MEMBER PREC(12) REAL UNAL VARIABLE	
695	COUNT	UNSIGNED	BIN INT MEMBER PREC(12) REAL UNAL VARIABLE	
246	COUNT	UNSIGNED	BIN INT MEMBER PREC(12) REAL UNAL VARIABLE	
275	COUNT	UNSIGNED	BIN INT MEMBER PREC(12) REAL UNAL	

line	name	type	attributes	references by line
			VARIABLE	
189	COUNT	UNSIGNED	BIN INT MEMBER PREC(12) REAL UNAL VARIABLE	
224	COUNT	UNSIGNED	BIN INT MEMBER PREC(12) REAL UNAL VARIABLE	229
129	COUNT	UNSIGNED	BIN INT MEMBER PREC(12) REAL UNAL VARIABLE	
706	CST1	STRUCTURE	INT MEMBER UNAL VARIABLE	
143	CST1	STRUCTURE	INT MEMBER UNAL VARIABLE	
252	CST1	STRUCTURE	INT MEMBER UNAL VARIABLE	
1133	CST1	STRUCTURE	INT MEMBER UNAL VARIABLE	
277	CST1	STRUCTURE	INT MEMBER UNAL VARIABLE	
781	CYL	UNSIGNED	BIN INT MEMBER PREC(10) REAL UNAL VARIABLE	
1049	CYL	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	1058 1059 1062
790	CYL	UNSIGNED	BIN INT MEMBER PREC(10) REAL UNAL VARIABLE	791
788	CYL	FIXED BIN	ALIGNED INT PARM PREC(35) REAL VARIABLE	791 787
1051	CYL	UNSIGNED	BIN INT MEMBER PREC(10) REAL UNAL VARIABLE	1058
798	CYL	UNSIGNED	BIN INT MEMBER PREC(10) REAL UNAL VARIABLE	800
1886	CYL	UNSIGNED	BIN INT MEMBER PREC(10) REAL UNAL VARIABLE	1895
938	CYL	UNSIGNED	BIN INT MEMBER PREC(10) REAL UNAL VARIABLE	
1052	CYL	UNSIGNED	BIN INT MEMBER PREC(10) REAL UNAL VARIABLE	1061
824	CYL	UNSIGNED	BIN INT MEMBER PREC(10) REAL UNAL VARIABLE	878 1902 1916
823	CYL	UNSIGNED	BIN INT MEMBER PREC(10) REAL UNAL VARIABLE	
972	CYL	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	981 983 983 985
829	CYL	UNSIGNED	BIN INT MEMBER PREC(10) REAL UNAL VARIABLE	
834	CYL	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	838 840
947	CYL	UNSIGNED	BIN INT MEMBER PREC(10) REAL UNAL VARIABLE	
872	CYL	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	876 878
950	CYL	UNSIGNED	BIN INT MEMBER PREC(10) REAL UNAL VARIABLE	
897	CYL	UNSIGNED	BIN INT MEMBER PREC(10) REAL UNAL VARIABLE	899 905
912	CYL	UNSIGNED	BIN INT MEMBER PREC(10) REAL UNAL VARIABLE	
923	CYL	UNSIGNED	BIN INT MEMBER PREC(10) REAL UNAL VARIABLE	925
159	CYL	UNSIGNED	BIN INT MEMBER PREC(16) REAL UNAL VARIABLE	194 1014 1024 1034 1059
1086	CYL	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	1099 1102 1119
1453	CYL	FIXED BIN	ALIGNED INT PARM PREC(35) REAL VARIABLE	1454 1452
1674	CYL	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	1675 1676

line	name	type	attributes	references by line
652	CYL	FIXED BIN	ALIGNED INT PARM PREC(35) REAL VARIABLE	665 670 682 682 650
170	CYL	UNSIGNED	BIN INT MEMBER PREC(16) REAL UNAL VARIABLE	1014 1024 1034 1061 1641 1676 1724 1915
1483	CYL	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	1486 1487
309	CYLS	FIXED BIN	ALIGNED INT MEMBER PREC(35) REAL VARIABLE	385 386 488 682 1032
275	D	STRUCTURE	INT LIKE PARM UNAL VARIABLE	279 273
246	D	STRUCTURE	INT LIKE PARM UNAL VARIABLE	255 243
224	D	STRUCTURE	AUTO INT LIKE UNAL VARIABLE	231
617	D	CHAR(2)	INT MEMBER NONVARYING UNAL VARIABLE	619
617	D	STRUCTURE	AUTO INT UNAL VARIABLE	618
1005	D	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	1008 1018 1028 1038
206	D$FORM	LITERALLY		1681
209	D$FORMBA	LITERALLY		1070 1917
210	D$FORMBN	LITERALLY		1066 1907
208	D$FORMGA	LITERALLY		1027
207	D$FORMGP	LITERALLY		1017 1037
215	D$NORCV	LITERALLY		396 1562
200	D$READ	LITERALLY		710 1109
214	D$RECOV	LITERALLY		431 1549
202	D$RHEAD	LITERALLY		1635 1668 1776 1905 1914
201	D$WRITE	LITERALLY		724 1106
7	DATA	FIXED BIN	ALIGNED INT MEMBER PREC(35) REAL VARIABLE	
7	DATA	FIXED BIN	ALIGNED INT MEMBER PREC(35) REAL VARIABLE	
178	DATA	BIT(64)	INT MEMBER NONVARYING UNAL VARIABLE	
	DATE	BUILTIN	INT	618
125	DCW	STRUCTURE	ALIGNED BASED INT VARIABLE	189 220 224 246 275 695 1083
277	DCWNO	BIT(36)	INT MEMBER NONVARYING UNAL VARIABLE	
1133	DCWNO	BIT(36)	INT MEMBER NONVARYING UNAL VARIABLE	
252	DCWNO	BIT(36)	INT MEMBER NONVARYING UNAL VARIABLE	
149	DCWNO	BIT(36)	INT MEMBER NONVARYING UNAL VARIABLE	
706	DCWNO	BIT(36)	INT MEMBER NONVARYING UNAL VARIABLE	
1220	DISPATCH	ENTRY	CONSTANT INT	1305
445	DRIVE	ENTRY	CONSTANT INT	396 431 1549 1562
518	DROP	ENTRY	CONSTANT INT	1303 1307 1435
427	DROPPACK	ENTRY	CONSTANT INT	344 1534
849	DUMPATT	ENTRY	CONSTANT INT	1807 1833 1865 1869
937	DUPL	ENTRY	CONSTANT INT	902 929
851	E	BIT(1)	ALIGNED AUTO INT NONVARYING VARIABLE	854 858 864
97	ELIF	LITERALLY		1157 1161 1165 1167 1405 1718
1326	ERREXIT	ENTRY	CONSTANT INT	115 286 321 349 365 380 488 503 888 916 1181 1294 1443 1455 1468 1514 1582 1595 1895 1912
1317	ERROR	ENTRY	CONSTANT INT	939 984 987 1328 1404 1408
1090	ERRS	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	1103 1107 1107 1112 1112 1115 1120 1139
142	EXEC	BIT(9)	INT MEMBER NONVARYING UNAL VARIABLE	
706	EXEC	BIT(9)	INT MEMBER NONVARYING UNAL VARIABLE	712 712 712 712
277	EXEC	BIT(9)	INT MEMBER NONVARYING UNAL VARIABLE	281 281 281 281

line	name	type	attributes	references by line
1133	EXEC	BIT(9)	INT MEMBER NONVARYING UNAL VARIABLE	1141 1142 1143 1144
252	EXEC	BIT(9)	INT MEMBER NONVARYING UNAL VARIABLE	
1180	FAIL	LABEL	CONSTANT INT	1145 1148 1162 1170 1173
684	FAIL	LABEL	CONSTANT INT	659
362	FAIL	LABEL	CONSTANT INT	356
1084	FAILS	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	1095 1117 1118 1118 1127
4	FALSE	LITERALLY		399 400 683 684 772 854 858 879
				882 958 961 1154 1155 1163 1171 1179
				1281 1387 1396 1423 1765
338	FCB	FILE	CONSTANT INT	356 357 364 365 382
6	FFRN	UNSIGNED	BIN INT MEMBER PREC(18) REAL UNAL VARIABLE	
6	FFRN	UNSIGNED	BIN INT MEMBER PREC(18) REAL UNAL VARIABLE	
6	FFRN	UNSIGNED	BIN INT MEMBER PREC(18) REAL UNAL VARIABLE	
244	FILE	FILE	ALIGNED INT PARM VARIABLE	254 243
300	FILE	FILE	ALIGNED INT MEMBER VARIABLE	279 321 382 396 429 431 432 433
				471 710 1138 1549 1562 1930
447	FILE	FILE	ALIGNED INT PARM VARIABLE	451 445
496	FILEAD	ENTRY	CONSTANT INT RETURNS	1062 1102 1487 1902
466	FILEP	ENTRY	CONSTANT INT	397 709 711 723 725 1016 1026 1036
				1062 1102 1487 1732 1776 1902 1909
948	FIND	FIXED BIN	ALIGNED INT PARM PREC(35) REAL VARIABLE	957 946
	FIXEDBIN	BUILTIN	INT	286 670 674 678 679 1681
166	FLAG	STRUCTURE	INT MEMBER UNAL VARIABLE	
1000	FORMAT	ENTRY	CONSTANT INT	1831 1867
417	FREECODE	BIT(18)	INIT INT MEMBER NONVARYING UNAL VARIABLE	392
829	FREETRK	STRUCTURE	AUTO INT LIKE UNAL VARIABLE	392 840 856 914 926 1056 1912
	FRN	BUILTIN	INT	254 451 471 1937
335	GETPACK	ENTRY	CONSTANT INT	1498
1375	GIVEN	CHAR	ALIGNED INT PARM VARIABLE VAR	1389 1394 1398 1404 1408 1410 1371
1383	GLEN	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	1389 1392 1394 1410
871	GOODATT	ENTRY	CONSTANT INT RETURNS	400
746	GOODLABEL	ENTRY	CONSTANT INT RETURNS	400
1003	H	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	1013 1015 1023 1025 1033 1035
161	H	BIT(1)	INT MEMBER NONVARYING UNAL VARIABLE	
485	H	FIXED BIN	ALIGNED INT PARM PREC(35) REAL VARIABLE	488 490 483
497	H	FIXED BIN	ALIGNED INT PARM PREC(35) REAL VARIABLE	501 496
158	HA	STRUCTURE	INT MEMBER UNAL VARIABLE	
193	HANAME	ENTRY	CONSTANT INT RETURNS	1638 1723 1726 1744
302	HASATT	BIT(1)	ALIGNED INT MEMBER NONVARYING VARIABLE	399 400 401 772 888
	HBOUND	BUILTIN	INT	373 379 855 954 1054 1393 1409
1886	HD	UNSIGNED	BIN INT MEMBER PREC(8) REAL UNAL VARIABLE	
1087	HD	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	1100 1102 1119
1674	HD	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	1675 1676
1453	HD	FIXED BIN	ALIGNED INT PARM PREC(35) REAL VARIABLE	1454 1452
171	HD	UNSIGNED	BIN INT MEMBER PREC(16) REAL UNAL VARIABLE	1015 1025 1035 1061 1642 1676 1724 1915
653	HD	FIXED BIN	ALIGNED INT PARM PREC(35) REAL VARIABLE	666 674 678 682 682 650
160	HD	UNSIGNED	BIN INT MEMBER PREC(16) REAL UNAL	194 1015 1025 1035 1059

line	name	type	attributes	references by line
			VARIABLE	
1483	HD	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	1486 1487
938	HD	UNSIGNED	BIN INT MEMBER PREC(8) REAL UNAL	
			VARIABLE	
789	HD	FIXED BIN	ALIGNED INT PARM PREC(35) REAL VARIABLE	791 787
1051	HD	UNSIGNED	BIN INT MEMBER PREC(8) REAL UNAL	1058
			VARIABLE	
790	HD	UNSIGNED	BIN INT MEMBER PREC(8) REAL UNAL	791
			VARIABLE	
1050	HD	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	1058 1059 1062
782	HD	UNSIGNED	BIN INT MEMBER PREC(8) REAL UNAL	
			VARIABLE	
798	HD	UNSIGNED	BIN INT MEMBER PREC(8) REAL UNAL	800
			VARIABLE	
1052	HD	UNSIGNED	BIN INT MEMBER PREC(8) REAL UNAL	1061
			VARIABLE	
824	HD	UNSIGNED	BIN INT MEMBER PREC(8) REAL UNAL	878 1903 1916
			VARIABLE	
823	HD	UNSIGNED	BIN INT MEMBER PREC(8) REAL UNAL	
			VARIABLE	
973	HD	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	981 983 983 985
829	HD	UNSIGNED	BIN INT MEMBER PREC(8) REAL UNAL	
			VARIABLE	
835	HD	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	839 840
947	HD	UNSIGNED	BIN INT MEMBER PREC(8) REAL UNAL	
			VARIABLE	
923	HD	UNSIGNED	BIN INT MEMBER PREC(8) REAL UNAL	925
			VARIABLE	
950	HD	UNSIGNED	BIN INT MEMBER PREC(8) REAL UNAL	
			VARIABLE	
873	HD	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	877 878
912	HD	UNSIGNED	BIN INT MEMBER PREC(8) REAL UNAL	
			VARIABLE	
897	HD	UNSIGNED	BIN INT MEMBER PREC(8) REAL UNAL	
			VARIABLE	
315	HEADS	LITERALLY		386 488 490 665 666 682 810 813
				822 839 877 925 983 1013 1023 1033
				1100
1510	HFILE	FILE	CONSTANT INT	1514 1516 1518 1519 1522
1512	HFILENAME	CHAR(25)	INIT INT NONVARYING STATIC UNAL VARIABLE	1514 1516
799	HH	PICTURE	INT MEMBER UNAL VARIABLE	800
1093	I	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	1120 1121
1380	I	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	1393 1394 1397 1409 1410 1411
601	I	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	605 606 607
520	I	FIXED BIN	ALIGNED INT PARM PREC(35) REAL VARIABLE	521 521 518
543	I	FIXED BIN	ALIGNED INT PARM PREC(35) REAL VARIABLE	545 547 547 541
340	I	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	373 373 379 383 384 392
577	I	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	580 580 580
135	I$IONTP	LITERALLY		
132	I$IOTD	LITERALLY		228
133	I$IOTP	LITERALLY		
134	I$TDCW	LITERALLY		

line	name	type	attributes	references by line
740	IDATE	BIT(36)	INT MEMBER NONVARYING UNAL VARIABLE	768 1801 1862
7	INDEFINITELY	LITERALLY		1281
	INDEX	BUILTIN	INT	661 671 1424
1927	INIT	ENTRY	CONSTANT INT	1944
220	IOTD	ENTRY	CONSTANT INT RETURNS	189 695 1096
179	ISZ	BIT(4)	INT MEMBER NONVARYING UNAL VARIABLE	
556	L	CHAR	ALIGNED INT PARM VARIABLE VAR	558 555
222	L	FIXED BIN	ALIGNED INT PARM PREC(35) REAL VARIABLE	229 220
499	L	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	501 502 504
733	LABEL	STRUCTURE	BASED INT UNAL VARIABLE	
	LBOUND	BUILTIN	INT	837 855 875 954 1054 1388 1393 1409
	LENGTH	BUILTIN	INT	348 348 370 371 664 980 1285 1383 1389 1422 1467
1273	LETTERS	CHAR(53)	INIT INT NONVARYING STATIC UNAL VARIABLE	1298
1852	LINE	CHAR	ALIGNED AUTO INT VARIABLE VAR	
1511	LINE	CHAR	ALIGNED AUTO INT VARIABLE VAR	1519 1519
1341	LINE	CHAR	ALIGNED AUTO INT VARIABLE VAR	1345 1346
971	LINE	CHAR	ALIGNED AUTO INT VARIABLE VAR	979 980 981 989
467	LOC	FIXED BIN	ALIGNED INT PARM PREC(35) REAL VARIABLE	473 477 466
301	LOC	FIXED BIN	ALIGNED INT MEMBER PREC(35) REAL VARIABLE	477 708 722 1708 1894 1901
1701	LOC	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	1710 1713 1728 1728 1750 1750 1754 1754 1755 1755 1756 1764 1776
807	LOC	FIXED BIN	ALIGNED INT PARM PREC(35) REAL VARIABLE	809 806
307	LOGRECS	FIXED BIN	ALIGNED INT MEMBER PREC(35) REAL VARIABLE	383 386 502 504 809 1008 1709
411	LOGRECS	FIXED BIN	ALIGNED INIT INT MEMBER PREC(35) REAL VARIABLE	383
1371	LOOKUP	ENTRY	CONSTANT INT RETURNS	1297
617	M	CHAR(2)	INT MEMBER NONVARYING UNAL VARIABLE	619
110	M	FIXED BIN	ALIGNED INT PARM PREC(35) REAL VARIABLE	113 115 109
146	MAJOR	UNSIGNED	BIN INT MEMBER PREC(4) REAL UNAL VARIABLE	
252	MAJOR	UNSIGNED	BIN INT MEMBER PREC(4) REAL UNAL VARIABLE	
277	MAJOR	UNSIGNED	BIN INT MEMBER PREC(4) REAL UNAL VARIABLE	283
706	MAJOR	UNSIGNED	BIN INT MEMBER PREC(4) REAL UNAL VARIABLE	713
1133	MAJOR	UNSIGNED	BIN INT MEMBER PREC(4) REAL UNAL VARIABLE	1151 1157 1165 1167
650	MAKEADDRESS	ENTRY	CONSTANT INT RETURNS	981 1454
1187	MAKEPATTERN	ENTRY	CONSTANT INT	1097 1111
1381	MATCH	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	1390 1396 1397 1398 1399 1403 1406
177	MBZ	BIT(4)	INT MEMBER NONVARYING UNAL VARIABLE	
127	MBZ	BIT(4)	INT MEMBER NONVARYING UNAL VARIABLE	
189	MBZ	BIT(4)	INT MEMBER NONVARYING UNAL VARIABLE	
168	MBZ	BIT(5)	INT MEMBER NONVARYING UNAL VARIABLE	
224	MBZ	BIT(4)	INT MEMBER NONVARYING UNAL VARIABLE	227
275	MBZ	BIT(4)	INT MEMBER NONVARYING UNAL VARIABLE	
1083	MBZ	BIT(4)	INT MEMBER NONVARYING UNAL VARIABLE	
246	MBZ	BIT(4)	INT MEMBER NONVARYING UNAL VARIABLE	

line	name	type	attributes	references by line
695	MBZ	BIT(4)	INT MEMBER NONVARYING UNAL VARIABLE	
165	MBZ0	BIT(4)	INT MEMBER NONVARYING UNAL VARIABLE	
173	MBZ1	BIT(2)	INT MEMBER NONVARYING UNAL VARIABLE	
175	MBZ2	BIT(12)	INT MEMBER NONVARYING UNAL VARIABLE	
1327	MESS	CHAR	ALIGNED INT PARM VARIABLE VAR	1328 1326
1318	MESS	CHAR	ALIGNED INT PARM VARIABLE VAR	1320 1317
277	MINOR	BIT(6)	INT MEMBER NONVARYING UNAL VARIABLE	
1133	MINOR	BIT(6)	INT MEMBER NONVARYING UNAL VARIABLE	1153 1159 1161 1169
252	MINOR	BIT(6)	INT MEMBER NONVARYING UNAL VARIABLE	
706	MINOR	BIT(6)	INT MEMBER NONVARYING UNAL VARIABLE	
147	MINOR	BIT(6)	INT MEMBER NONVARYING UNAL VARIABLE	
739	MISC	FIXED BIN	INT MEMBER PREC(35) REAL UNAL VARIABLE	749 760 1800
	MME	BUILTIN	INT	113 261
440	MME$DRIVE	LITERALLY		455
248	MME$DRIVE	LITERALLY		261
464	MME$SETP	LITERALLY		475
109	MMEO	ENTRY	CONSTANT INT	455 475
	MOD	BUILTIN	INT	502 666 667 810
274	MODE	FIXED BIN	ALIGNED INT PARM PREC(35) REAL VARIABLE	279 273
245	MODE	FIXED BIN	ALIGNED INT PARM PREC(35) REAL VARIABLE	259 243
	MORE	BUILTIN	INT	1518
799	N	STRUCTURE	INT STATIC UNAL VARIABLE	801
1741	NAME	CHAR(*)	INT NONVARYING PARM UNAL VARIABLE	1758 1739
737	NAME	BIT(72)	INT MEMBER NONVARYING UNAL VARIABLE	758 1798 1798
408	NAME	CHAR(6)	INIT INT MEMBER NONVARYING UNAL VARIABLE	373
303	NAME	STRUCTURE	ALIGNED INT MEMBER VARIABLE	
339	NAME	CHAR(8)	ALIGNED AUTO INT VARIABLE VAR	350 358 365 370 370 371 371
336	NAMESTR	CHAR	ALIGNED INT PARM VARIABLE VAR	348 348 350 335
754	NEWLABEL	ENTRY	CONSTANT INT	1828
1928	NOPROMPT	ENTRY	CONSTANT EXT	1939
3	NOT	LITERALLY		283 400 401 683 713 772 854 882 888 958 1106 1109 1155 1281 1387 1454 1912
	NULLF	BUILTIN	INT	321 429 433 1930
736	NUMB	BIT(36)	INT MEMBER NONVARYING UNAL VARIABLE	767 1797 1861
	OCTAL	BUILTIN	INT	115 115 286 365 1121 1181 1800
706	OFF	BIT(1)	INT MEMBER NONVARYING UNAL VARIABLE	713
252	OFF	BIT(1)	INT MEMBER NONVARYING UNAL VARIABLE	
1133	OFF	BIT(1)	INT MEMBER NONVARYING UNAL VARIABLE	1147
145	OFF	BIT(1)	INT MEMBER NONVARYING UNAL VARIABLE	
277	OFF	BIT(1)	INT MEMBER NONVARYING UNAL VARIABLE	283
275	OFFSET	UNSIGNED	BIN INT MEMBER PREC(18) REAL UNAL VARIABLE	
695	OFFSET	UNSIGNED	BIN INT MEMBER PREC(18) REAL UNAL VARIABLE	
246	OFFSET	UNSIGNED	BIN INT MEMBER PREC(18) REAL UNAL VARIABLE	
224	OFFSET	UNSIGNED	BIN INT MEMBER PREC(18) REAL UNAL VARIABLE	226
1083	OFFSET	UNSIGNED	BIN INT MEMBER PREC(18) REAL UNAL VARIABLE	
189	OFFSET	UNSIGNED	BIN INT MEMBER PREC(18) REAL UNAL	

line	name	type	attributes	references by line
			VARIABLE	
126	OFFSET	UNSIGNED	BIN INT MEMBER PREC(18) REAL UNAL VARIABLE	
1132	OP	FIXED BIN	ALIGNED INT PARM PREC(35) REAL VARIABLE	1138 1130
1482	OPTADDRESS	ENTRY	CONSTANT INT	1581 1633 1890
1495	OPTDEV	ENTRY	CONSTANT INT	1594 1609 1705 1789 1823 1854
2	OR	LITERALLY		281 281 281 348 488 488 712 712 712 878 983 983 983 1467
299	PACK	STRUCTURE	ALIGNED AUTO INT VARIABLE	
321	PACKCHECK	ENTRY	CONSTANT INT	1548 1561 1579 1610 1632 1666 1706 1793 1825 1856 1889
94	PACKFMT	ENTRY	CONSTANT EXT OPTIONS	
1188	PATTERN	CHAR(1)	DIM(0:255) INIT INT NONVARYING STATIC UNAL VARIABLE	1192
101	PERCAT	FILE	CONSTANT INT	1535 1932 1937
100	PERCATFRN	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	359 1937
243	PIO	ENTRY	CONSTANT INT RETURNS	279 710 1138
273	PIOO	ENTRY	CONSTANT INT	724 1017 1027 1037 1066 1070 1635 1663 1681 1776 1905 1907 1914 1917
140	PIOSTATUS	STRUCTURE	BASED INT UNAL VARIABLE	243 252 277 706 1133
720	PLOC	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	722 725
705	PLOC	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	708 711
1703	PLOC	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	1708 1732
1899	PLOC	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	1901 1909
304	PREFIX	CHAR(6)	INT MEMBER NONVARYING UNAL VARIABLE	371 373 380
1338	PROMPT	ENTRY	CONSTANT INT RETURNS	979 989 1283
1339	PROMPT	CHAR(*)	INT NONVARYING PARM UNAL VARIABLE	1344 1338
1133	QADD	UNSIGNED	BIN INT MEMBER PREC(6) REAL UNAL VARIABLE	
706	QADD	UNSIGNED	BIN INT MEMBER PREC(6) REAL UNAL VARIABLE	
277	QADD	UNSIGNED	BIN INT MEMBER PREC(6) REAL UNAL VARIABLE	
252	QADD	UNSIGNED	BIN INT MEMBER PREC(6) REAL UNAL VARIABLE	
148	QADD	UNSIGNED	BIN INT MEMBER PREC(6) REAL UNAL VARIABLE	
544	R	CHAR	ALIGNED AUTO INT VARIABLE VAR	546 547 548
1189	R	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	1191 1192
1432	R	CHAR	ALIGNED AUTO INT VARIABLE VAR	1434 1436
111	R	FIXED BIN	ALIGNED DIM(1:12) INT PARM PREC(35) REAL VARIABLE	113 114 115 109
164	ROCOUNT	STRUCTURE	INT MEMBER UNAL VARIABLE	
176	RODATA	STRUCTURE	INT MEMBER UNAL VARIABLE	
2	RA	FIXED BIN	ALIGNED INT MEMBER PREC(35) REAL VARIABLE	259
2	RA	FIXED BIN	ALIGNED INT MEMBER PREC(35) REAL VARIABLE	454
2	RA	FIXED BIN	ALIGNED INT MEMBER PREC(35) REAL VARIABLE	473
310	RANGE	FIXED BIN	ALIGNED INT MEMBER PREC(35) REAL VARIABLE	386 1713 1764

line	name	type	attributes	references by line
741	RDATE	BIT(36)	INT MEMBER NONVARYING UNAL VARIABLE	769 1802
1775	READTH	ENTRY	CONSTANT INT	1714 1767
704	READTRACKO	ENTRY	CONSTANT INT RETURNS	400
693	REC	CHAR(4)	DIM(0:63) INT MEMBER NONVARYING UNAL VARIABLE	
172	REC	UNSIGNED	BIN INT MEMBER PREC(8) REAL UNAL VARIABLE	
313	RECSIZE	LITERALLY		693 1082 1188 1188
1047	REFORMAT	ENTRY	CONSTANT INT	1832 1868
1	REG	STRUCTURE	ALIGNED AUTO INT VARIABLE	8
1	REG	STRUCTURE	ALIGNED AUTO INT VARIABLE	8
1	REG	STRUCTURE	ALIGNED AUTO INT VARIABLE	8
8	REGS	FIXED BIN	ALIGNED BASED DIM(1:12) INT PREC(35) REAL VARIABLE	261
8	REGS	FIXED BIN	ALIGNED BASED DIM(1:12) INT PREC(35) REAL VARIABLE	455
8	REGS	FIXED BIN	ALIGNED BASED DIM(1:12) INT PREC(35) REAL VARIABLE	475
765	RELABEL	ENTRY	CONSTANT INT	1835 1871 1919
1279	RESTART	LABEL	CONSTANT INT	1277
98	RESULT	LITERALLY		111 652 653 654 948
1134	RETRY	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	1136
2	RQ	FIXED BIN	ALIGNED INT MEMBER PREC(35) REAL VARIABLE	
2	RQ	FIXED BIN	ALIGNED INT MEMBER PREC(35) REAL VARIABLE	
1133	S	STRUCTURE	AUTO INT LIKE UNAL VARIABLE	1138 1139 1181
486	S	FIXED BIN	ALIGNED INT PARM PREC(35) REAL VARIABLE	488 491 483
497	S	FIXED BIN	ALIGNED INT PARM PREC(35) REAL VARIABLE	501 496
519	S	CHAR	ALIGNED INT PARM VARIABLE VAR	521 521 521 518
542	S	CHAR	ALIGNED INT PARM VARIABLE VAR	546 546 547 547 547 541
589	S	CHAR(6)	INT NONVARYING PARM UNAL VARIABLE	606 588
1004	S	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	1009 1016 1018 1018 1026 1028 1028 1035 1038 1038
252	S	STRUCTURE	AUTO INT LIKE UNAL VARIABLE	263 265
706	S	STRUCTURE	AUTO INT LIKE UNAL VARIABLE	710
277	S	STRUCTURE	AUTO INT LIKE UNAL VARIABLE	279 286
799	S	CHAR(1)	INIT INT MEMBER NONVARYING UNAL VARIABLE	
656	S1	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	661 662 670 671 674 678 678
657	S2	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	671 672 678 679
1763	SAME	ENTRY	CONSTANT INT RETURNS	1751 1755
	SEARCH	BUILTIN	INT	1307 1434
946	SEARCHATT	ENTRY	CONSTANT INT RETURNS	901 914 1897 1912
1483	SECT	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	1486 1487
1453	SECT	FIXED BIN	ALIGNED INT PARM PREC(35) REAL VARIABLE	1454 1452
1674	SECT	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	1675
974	SECT	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	981
654	SECT	FIXED BIN	ALIGNED INT PARM PREC(35) REAL VARIABLE	667 675 679 682 682 650
314	SECTORS	LITERALLY		386 488 491 665 666 666 667 682 809 1008 1082 1191 1709

Line	name	type	attributes	references by line
655	SEEK	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	664 665 666 667
483	SEEKAD	ENTRY	CONSTANT INT RETURNS	501
742	SEQNO	BIT(36)	INT MEMBER NONVARYING UNAL VARIABLE	770 1803
306	SHAPE	STRUCTURE	ALIGNED INT MEMBER VARIABLE	
	SHL	BUILTIN	INT	1681
	SHR	BUILTIN	INT	114
735	SITE	BIT(36)	INT MEMBER NONVARYING UNAL VARIABLE	757 1796 1863
1092	ST	BIT(36)	AUTO DIM(0:3) INT NONVARYING UNAL VARIABLE	1121 1139
5	STATUS	UNSIGNED	BIN INT MEMBER PREC(9) REAL UNAL VARIABLE	
5	STATUS	UNSIGNED	BIN INT MEMBER PREC(9) REAL UNAL VARIABLE	
5	STATUS	UNSIGNED	BIN INT MEMBER PREC(9) REAL UNAL VARIABLE	
1702	STEP	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	1709 1728 1750 1754 1755 1756
651	STR	CHAR	ALIGNED INT PARM VARIABLE VAR	661 664 664 670 671 674 678 679 650
	STRING	BUILTIN	INT	263 286 392 579 581 609 618 801 856 856 859 859 926 926 928 923 955 955 1007 1056 1056 1063 1063 1139 1181 1192 1908 1908
	STW1	BUILTIN	INT	364 365
	SUBSTR	BUILTIN	INT	115 286 370 371 521 547 547 632 632 632 670 674 678 679 1121 1139 1181 1287 1288 1291 1295 1394 1410 1424 1798 1798
305	SUFFIX	CHAR(2)	INT MEMBER NONVARYING UNAL VARIABLE	370 767
9	SW1	LITERALLY		
9	SW1	LITERALLY		
9	SW1	LITERALLY		
9	SW2	LITERALLY		
9	SW2	LITERALLY		
9	SW2	LITERALLY		
3	SWS	STRUCTURE	ALIGNED INT MEMBER VARIABLE	
3	SWS	STRUCTURE	ALIGNED INT MEMBER VARIABLE	
3	SWS	STRUCTURE	ALIGNED INT MEMBER VARIABLE	263
252	SYNC	BIT(1)	INT MEMBER NONVARYING UNAL VARIABLE	
706	SYNC	BIT(1)	INT MEMBER NONVARYING UNAL VARIABLE	
144	SYNC	BIT(1)	INT MEMBER NONVARYING UNAL VARIABLE	
277	SYNC	BIT(1)	INT MEMBER NONVARYING UNAL VARIABLE	
1133	SYNC	BIT(1)	INT MEMBER NONVARYING UNAL VARIABLE	
103	SYSIN	FILE	CONSTANT EXT	1939 1345
104	SYSPRINT	FILE	CONSTANT EXT	1866 1860 1830 1794 1758 1756 1753 1743 1725 1722 1647 1640 1637 1613 1519 1411 1343 1319 1126 1121 1119 1117 865 864 861 860 852 401
790	T	STRUCTURE	AUTO INT LIKE UNAL VARIABLE	792
798	T	STRUCTURE	INT LIKE PARM UNAL VARIABLE	797
950	T	STRUCTURE	AUTO INT LIKE UNAL VARIABLE	953 955
923	T	STRUCTURE	INT LIKE PARM UNAL VARIABLE	928 929 922
912	T	STRUCTURE	INT LIKE PARM UNAL VARIABLE	915 911

line	name	type	attributes	references by line
938	T	STRUCTURE	INT LIKE PARM UNAL VARIABLE	939 937
897	T	STRUCTURE	INT LIKE PARM UNAL VARIABLE	901 902 903 905 896
808	T	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	809 810 810
1376	TABLE	CHAR(*)	DIM(*) INT NONVARYING PARM UNAL VARIABLE	1388 1388 1393 1393 1394 1398 1409 1409 1410 1411 1371
317	TDCYLS	LITERALLY		385
1078	TEST	ENTRY	CONSTANT INT RETURNS	1612
1130	TESTOP	ENTRY	CONSTANT INT RETURNS	1106 1109
157	TH	STRUCTURE	AUTO INT UNAL VARIABLE	189 189 1007 1645
189	THDCW	STRUCTURE	AUTO INT LIKE UNAL VARIABLE	189 1017 1027 1037 1066 1070 1635 1663 1682 1776 1905 1907 1914 1917
163	TI	BIT(2)	INT MEMBER NONVARYING UNAL VARIABLE	1011 1021 1031 1065 1069 1639 1670 1681 1716 1718 1721 1768 1906 1915
1740	TI	BIT(2)	INT NONVARYING PARM UNAL VARIABLE	1768 1739
169	TI	BIT(2)	INT MEMBER NONVARYING UNAL VARIABLE	1011 1021 1031 1065 1069 1643 1670 1905 1915
185	TI$BA	LITERALLY		1069 1721 1915
186	TI$BN	LITERALLY		1065 1906
184	TI$GA	LITERALLY		1021 1718 1719
183	TI$GP	LITERALLY		1011 1031 1716 1717
1384	TLEN	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	1388 1392
787	TRACK	ENTRY	CONSTANT INT RETURNS	194 810 840 985 1119 1641 1724
692	TRACKO	STRUCTURE	AUTO DIM(0:3) INT UNAL VARIABLE	695 695 699 733 822
695	TRACKODCW	STRUCTURE	AUTO INT LIKE UNAL VARIABLE	695 710 724
806	TRACKAD	ENTRY	CONSTANT INT RETURNS	1756 1894
780	TRACKCODE	STRUCTURE	BASED INT UNAL VARIABLE	787 790 798 806 823 824 829 897 912 923 938 947 950 1051 1052 1885
797	TRACKNAME	ENTRY	CONSTANT INT RETURNS	194 860 860 861 939 1119 1641 1724 1756
	TRANSLATE	BUILTIN	INT	558
1080	TRIALS	LITERALLY		1092 1105
1886	TRK	STRUCTURE	AUTO INT LIKE UNAL VARIABLE	1894 1897 1913
1051	TRK	STRUCTURE	AUTO INT LIKE UNAL VARIABLE	1055 1056 1063
947	TRK	STRUCTURE	INT LIKE PARM UNAL VARIABLE	953 946
823	TRK	STRUCTURE	INT LIKE MEMBER UNAL VARIABLE	840 856 859 860 861 915 926 923 930 932 955 1055 1908 1913
5	TRUE	LITERALLY		400 683 772 854 882 958 1155 1281 1387
1089	TRY	FIXED BIN	ALIGNED AUTO INT PREC(35) REAL VARIABLE	1105
407	TYPE	STRUCTURE	ALIGNED DIM(1:2) INT STATIC VARIABLE	373 379
738	TYPE	BIT(36)	INT MEMBER NONVARYING UNAL VARIABLE	748 759 1799
1382	UNIQUE	BIT(1)	ALIGNED AUTO INT NONVARYING VARIABLE	1387 1396 1405
	UNSPEC	BUILTIN	INT	263 699
555	UPPER	ENTRY	CONSTANT INT RETURNS	1297 1475
308	USECYLS	FIXED BIN	ALIGNED INT MEMBER PREC(35) REAL VARIABLE	384 385 838 838 876 876 899 905 925 983 1012 1022 1022 1032 1099 1895
414	USECYLS	FIXED BIN	ALIGNED INIT INT MEMBER PREC(35) REAL VARIABLE	384
	VERIFY	BUILTIN	INT	1298 1303 1435 1467
	WADDR	BUILTIN	INT	189 255 695 1096
	WLEN	BUILTIN	INT	189 695 1096
719	WRITETRACKO	ENTRY	CONSTANT INT	771 1616

line	name	type	attributes	references by line
2	X0	FIXED BIN	ALIGNED INT MEMBER PREC(35) REAL VARIABLE	451
2	X0	FIXED BIN	ALIGNED INT MEMBER PREC(35) REAL VARIABLE	254
2	X0	FIXED BIN	ALIGNED INT MEMBER PREC(35) REAL VARIABLE	471
2	X1	FIXED BIN	ALIGNED INT MEMBER PREC(35) REAL VARIABLE	
2	X1	FIXED BIN	ALIGNED INT MEMBER PREC(35) REAL VARIABLE	
2	X1	FIXED BIN	ALIGNED INT MEMBER PREC(35) REAL VARIABLE	255
2	X2	FIXED BIN	ALIGNED INT MEMBER PREC(35) REAL VARIABLE	
2	X2	FIXED BIN	ALIGNED INT MEMBER PREC(35) REAL VARIABLE	
2	X2	FIXED BIN	ALIGNED INT MEMBER PREC(35) REAL VARIABLE	
2	X3	FIXED BIN	ALIGNED INT MEMBER PREC(35) REAL VARIABLE	
2	X3	FIXED BIN	ALIGNED INT MEMBER PREC(35) REAL VARIABLE	
2	X3	FIXED BIN	ALIGNED INT MEMBER PREC(35) REAL VARIABLE	
2	X4	FIXED BIN	ALIGNED INT MEMBER PREC(35) REAL VARIABLE	256
2	X4	FIXED BIN	ALIGNED INT MEMBER PREC(35) REAL VARIABLE	452
2	X4	FIXED BIN	ALIGNED INT MEMBER PREC(35) REAL VARIABLE	
2	X5	FIXED BIN	ALIGNED INT MEMBER PREC(35) REAL VARIABLE	
2	X5	FIXED BIN	ALIGNED INT MEMBER PREC(35) REAL VARIABLE	
2	X5	FIXED BIN	ALIGNED INT MEMBER PREC(35) REAL VARIABLE	
2	X6	FIXED BIN	ALIGNED INT MEMBER PREC(35) REAL VARIABLE	257
2	X6	FIXED BIN	ALIGNED INT MEMBER PREC(35) REAL VARIABLE	472
2	X6	FIXED BIN	ALIGNED INT MEMBER PREC(35) REAL VARIABLE	453
2	X7	FIXED BIN	ALIGNED INT MEMBER PREC(35) REAL VARIABLE	258
2	X7	FIXED BIN	ALIGNED INT MEMBER PREC(35) REAL VARIABLE	
2	X7	FIXED BIN	ALIGNED INT MEMBER PREC(35) REAL VARIABLE	
617	Y	CHAR(2)	INT MEMBER NONVARYING UNAL VARIABLE	619
162	Z	BIT(1)	INT MEMBER NONVARYING UNAL VARIABLE	
617	_1	CHAR(1)	INT MEMBER NONVARYING UNAL VARIABLE	
617	_2	CHAR(1)	INT MEMBER NONVARYING UNAL VARIABLE	

line	name	type	attributes	references by line
1884	_ALTERNATE	ENTRY	CONSTANT INT	1227
1558	_DISABLE	ENTRY	CONSTANT INT	1236
1545	_ENABLE	ENTRY	CONSTANT INT	1237
1531	_EXIT	ENTRY	CONSTANT INT	1229
1821	_FORMAT	ENTRY	CONSTANT INT	1226
1509	_HELP	ENTRY	CONSTANT INT	1224
1787	_LABEL	ENTRY	CONSTANT INT	1232
1699	_MAP	ENTRY	CONSTANT INT	1231
1591	_PACK	ENTRY	CONSTANT INT	1230
1851	_REFORMAT	ENTRY	CONSTANT INT	1225
1577	_SET	ENTRY	CONSTANT INT	1233
1607	_TEST	ENTRY	CONSTANT INT	1228
1664	_TI	ENTRY	CONSTANT INT	1235
1630	_TRACKHEADER	ENTRY	CONSTANT INT	1234

External References and Definitions

This procedure defines the following external entry points:
PACKFMT

No external data were defined.

The following external entry points were referenced:
NOPROMPT

The following external data declarations were referenced:

CMDERROR	SYSIN	SYSPRINT

The following library routines were referenced by the object code:

.TTY	.TTY	.MME	.OCTAL	.SB3C	.MVCNC
.RET	.FRN	.SASN	.LEN	..SET4	.VARYC
.ON	.OPEN	.STW1	.PXDYN	.PTSK1	.LPAO
.CLOSE	.SB2D	.SASNN	.SB3D	.TRAN3	.BYTE2
.DATE	.INDC2	.GETS	.GEFIX	.C71	.INDC3
.BCNC	.PLCHR	.PLDYN	.VERIF	.SRCH	.PXFIX
.PECHR	.GXDYN	.CCND	.PXCHR	.GEBIT	.MORE
.PEDYN	.PEBIT	.FSLSK	.FSLCL	.FSLX	

Compilation Report

There were no errors in this external procedure.
893 statements were compiled.
4885 is the length of the object code.
226 is the length of the static data.
44K core used.
DTSS PL1 version of 06/01/81.

Warning: the following names were not explicitly declared:

ADDR	BYTE	CHR	DATE	FIXEDBIN	FRN
HBOUND	INDEX	LBOUND	LENGTH	MME	MOD
MORE	NULLF	OCTAL	SEARCH	SHL	SHR
STRING	STW1	SUBSTR	TRANSLATE	UNSPEC	VERIFY
WADDR	WLEN				

Warning: the following declarations were not used:
LINE

Warning: the following dcl's were referenced but not assigned:
TYPE

The following blocks were defined:

line	type	name	stack frame size	string descriptors
94	PROC	PACKFMT	292	1
109	PROC	MMEO	18	1
193	PROC	HANAME	shares frame of block at line	1268
220	PROC	IOTD	16	0
243	PROC	PIO	30	0
273	PROC	PIOO	shares frame of block at line	1268
321	PROC	PACKCHECK	shares frame of block at line	1268
335	PROC	GETPACK	shares frame of block at line	1268
355	BEGIN		12	0
356	ON		46	0
427	PROC	DROPPACK	shares frame of block at line	1268
445	PROC	DRIVE	shares frame of block at line	1268
466	PROC	FILEP	26	0
483	PROC	SEEKAD	shares frame of block at line	496
496	PROC	FILEAD	24	2
518	PROC	DROP	shares frame of block at line	1268
541	PROC	BREAK	shares frame of block at line	1268
555	PROC	UPPER	shares frame of block at line	1268
563	PROC	ACI	18	0
588	PROC	BCI	18	0
616	PROC	BDATE	shares frame of block at line	1268
624	PROC	ADATE	shares frame of block at line	1268
650	PROC	MAKEADDRESS	24	0
659	ON		46	0
699	PROC	CLEARTRACKO	shares frame of block at line	1268
704	PROC	READTRACKO	shares frame of block at line	1268
719	PROC	WRITETRACKO	shares frame of block at line	1268
746	PROC	GOODLABEL	shares frame of block at line	1268
754	PROC	NEWLABEL	shares frame of block at line	1268

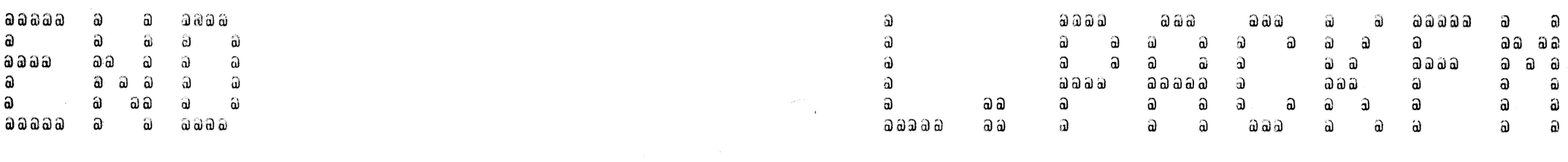
Compilation Report

```
  765  PROC   RELABEL          shares frame of block at line  1268
  787  PROC   TRACK                16                    0
  797  PROC   TRACKNAME            16                    0
  806  PROC   TRACKAD          shares frame of block at line  1268
  833  PROC   CLEARATT         shares frame of block at line  1268
  849  PROC   DUMPATT          shares frame of block at line  1268
  871  PROC   GOODATT          shares frame of block at line  1268
  887  PROC   ATTCHECK         shares frame of block at line  1268
  896  PROC   ADDATT           shares frame of block at line  1268
  911  PROC   ADDUSE           shares frame of block at line  1268
  922  PROC   ADDALT           shares frame of block at line  1268
  937  PROC   DUPL             shares frame of block at line  1268
  946  PROC   SEARCHATT         shares frame of block at line  1268
  969  PROC   ASKOP            shares frame of block at line  1268
 1000  PROC   FORMAT            shares frame of block at line  1268
 1047  PROC   REFORMAT          shares frame of block at line  1268
 1078  PROC   TEST                2610                    1
 1130  PROC   TESTOP           shares frame of block at line  1078
 1187  PROC   MAKEPATTERN      shares frame of block at line  1078
 1220  PROC   DISPATCH          shares frame of block at line  1268
 1268  PROC   COMMANDLOOP          448                   31
 1277  ON                           46                    0
 1317  PROC   ERROR                12                    0
 1326  PROC   ERREXIT              12                    0
 1338  PROC   PROMPT            shares frame of block at line  1268
 1371  PROC   LOOKUP            shares frame of block at line  1268
 1421  PROC   ARGLEFT           shares frame of block at line  1268
 1431  PROC   ARG               shares frame of block at line  1268
 1442  PROC   ARGCHECK          shares frame of block at line  1268
 1452  PROC   ARGADDRESS        shares frame of block at line  1268
 1463  PROC   ARGTI             shares frame of block at line  1268
 1475  PROC   ARGDEV            shares frame of block at line  1268
 1482  PROC   OPTADDRESS        shares frame of block at line  1268
 1495  PROC   OPTDEV            shares frame of block at line  1268
 1509  PROC   _HELP                12                    1
 1514  ON                           48                    1
 1531  PROC   _EXIT             shares frame of block at line  1268
 1545  PROC   _ENABLE           shares frame of block at line  1268
 1558  PROC   _DISABLE          shares frame of block at line  1268
 1577  PROC   _SET              shares frame of block at line  1268
 1591  PROC   _PACK             shares frame of block at line  1268
 1607  PROC   _TEST             shares frame of block at line  1268
 1630  PROC   _TRACKHEADER      shares frame of block at line  1268
 1664  PROC   _TI               shares frame of block at line  1268
 1673  BEGIN                    shares frame of block at line  1268
 1699  PROC   _MAP              shares frame of block at line  1268
 1739  PROC   AREA              shares frame of block at line  1268
 1763  PROC   SAME              shares frame of block at line  1268
 1775  PROC   READTH            shares frame of block at line  1268
 1787  PROC   _LABEL            shares frame of block at line  1268
 1821  PROC   _FORMAT           shares frame of block at line  1268
 1851  PROC   _REFORMAT         shares frame of block at line  1268
```

Compilation Report

```
1884  PROC   _ALTERNATE          shares frame of block at line  1268
1927  PROC   INIT                shares frame of block at line    94
```

B**BB**BB**BB**BB**BB**BB**BB**BB**BB**BB**BB**BB**BB* COMPUTER SHARING SERVICES *BB**BB**BB**BB**BB**BB**BB**BB**BB**BB**BB**BB**B3**B
B**BB**BB**BB**BB**BB**BB**BB**BB**BB**BB**BB**BB**BB* COMPUTER SHARING SERVICES *BB**BB**BB**BB**BB**BB**BB**BB**BB**BB**BB**BB**B3**B



B**BB**BB**BB**BB**BB**BB**BB**BB**BB**BB**BB**BB**BB* COMPUTER SHARING SERVICES *BB**BB**BB**BB**BB**BB**BB**BB**BB**BB**BB**BB**B3**B
263 263 263 263 263 263 263 263 263 263 263 263 263 263 263 263 263 263 263 263 263 263 263 263 263 263 263 263 263 263 263 263 263

02/04/82 15:53:27 PRINTOUT #263