G 2

```
TTTTTTTTTTTTTTT  RRRRRRRRRRRR        AAAAAAAAA        CCCCCCCCCCCC  EEEEEEEEEEEEEEE
TTTTTTTTTTTTTTT  RRRRRRRRRRRR        AAAAAAAAA        CCCCCCCCCCCC  EEEEEEEEEEEEEEE
TTTTTTTTTTTTTTT  RRRRRRRRRRRR        AAAAAAAAA        CCCCCCCCCCCC  EEEEEEEEEEEEEEE
      TTT        RRR         RRR  AAA         AAA  CCC              EEE
      TTT        RRR         RRR  AAA         AAA  CCC              EEE
      TTT        RRR         RRR  AAA         AAA  CCC              EEE
      TTT        RRR         RRR  AAA         AAA  CCC              EEE
      TTT        RRR         RRR  AAA         AAA  CCC              EEE
      TTT        RRR         RRR  AAA         AAA  CCC              EEE
      TTT        RRRRRRRRRRRR     AAA         AAA  CCC              EEEEEEEEEEEE
      TTT        RRRRRRRRRRRR     AAA         AAA  CCC              EEEEEEEEEEEE
      TTT        RRRRRRRRRRRR     AAA         AAA  CCC              EEEEEEEEEEEE
      TTT        RRR   RRR        AAAAAAAAAAAAAAA  CCC              EEE
      TTT        RRR   RRR        AAAAAAAAAAAAAAA  CCC              EEE
      TTT        RRR   RRR        AAAAAAAAAAAAAAA  CCC              EEE
      TTT        RRR      RRR     AAA         AAA  CCC              EEE
      TTT        RRR      RRR     AAA         AAA  CCC              EEE
      TTT        RRR      RRR     AAA         AAA  CCC              EEE
      TTT        RRR         RRR  AAA         AAA     CCCCCCCCCCCC  EEEEEEEEEEEEEEE
      TTT        RRR         RRR  AAA         AAA     CCCCCCCCCCCC  EEEEEEEEEEEEEEE
      TTT        RRR         RRR  AAA         AAA     CCCCCCCCCCCC  EEEEEEEEEEEEEEE
```

FILEID**TBKLIB

B 16

```
TTTTTTTTTT  BBBBBBBB    KK      KK  LL            IIIIII    BBBBBBBB
TTTTTTTTTT  BBBBBBBB    KK      KK  LL            IIIIII    BBBBBBBB
    TT      BB      BB  KK      KK  LL              II      BB      BB
    TT      BB      BB  KK      KK  LL              II      BB      BB
    TT      BB      BB  KK    KK    LL              II      BB      BB
    TT      BB      BB  KK    KK    LL              II      BB      BB
    TT      BBBBBBBB    KKKKKK      LL              II      BBBBBBBB
    TT      BBBBBBBB    KKKKKK      LL              II      BBBBBBBB
    TT      BB      BB  KK    KK    LL              II      BB      BB
    TT      BB      BB  KK    KK    LL              II      BB      BB
    TT      BB      BB  KK      KK  LL              II      BB      BB      ....
    TT      BB      BB  KK      KK  LL              II      BB      BB      ....
    TT      BBBBBBBB    KK      KK  LLLLLLLLLL    IIIIII    BBBBBBBB        ....
    TT      BBBBBBBB    KK      KK  LLLLLLLLLL    IIIIII    BBBBBBBB        ....

LL            IIIIII      SSSSSSSS
LL            IIIIII      SSSSSSSS
LL              II      SS
LL              II      SS
LL              II      SS
LL              II      SS
LL              II        SSSSSS
LL              II        SSSSSS
LL              II              SS
LL              II              SS
LL              II              SS
LL              II              SS
LLLLLLLLLL    IIIIII    SSSSSSSS
LLLLLLLLLL    IIIIII    SSSSSSSS
```

```
!--------------------------------------------------------------------
!
!        TBKLIB -- STANDARD REQUIRE FILE FOR VAX TRACE BLISS MODULES
!
!--------------------------------------------------------------------
!
! Version:      'V04-000'
!
!****************************************************************************
!*                                                                          *
!*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                 *
!*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                  *
!*  ALL RIGHTS RESERVED.                                                    *
!*                                                                          *
!*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED   *
!*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE   *
!*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER   *
!*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY   *
!*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY   *
!*  TRANSFERRED.                                                            *
!*                                                                          *
!*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE   *
!*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT   *
!*  CORPORATION.                                                            *
!*                                                                          *
!*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS   *
!*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                 *
!*                                                                          *
!*                                                                          *
!****************************************************************************
!
```

```
;   0   !++
;   0   !  TBKRST.BEG - Runtime SYmbol Table Literals and Structures
;   0   !
;   0   !  Revision History:
;   0   !
;   0   !  01   23-JUN-77       KGP    -Put together the initial version of this file.
;   0   !  02   13-JULY-77      KGP    -Changed all the data structure definitions
;   0   !                                so that now FIELD and FIELD SETs are
;   0   !                                used.
;   0   !  03   21-july-77      KGP    -Switched over to using SRM standard names
;   0   !                                for the DST record types.  (Appendix C)
;   0   !  04   28-july-77      KGP    -Started using RST_MC structure for the MC
;   0   !                                instead of BLOCK, and changed RST_MC and
;   0   !                                RST_NT structs to use an EXTERNAL LITERAL
;   0   !                                for the relocation, instead of an ordinary
;   0   !                                external, DBG$GL_RST_PTR.
;   0   !  05   02-AUG-77       KGP    -Reorganized NT and MC structures so that
;   0   !                                the shared fields were alligned so that we
;   0   !                                could look at any arbitrary record and
;   0   !                                deduce whether it was an NT or an MC record.
;   0   !  06   03-AUG-77       KGP    -Added field names to NT and MC structures so tha
;   0   !                                we can pick up the address of the symbol name.
;   0   !                                This is an incompatible change to previous
;   0   !                                versions of this file because the old field name
;   0   !                                no longer exists.
;   0   !  07   10-AUG-77       KGP    -Added the definition of GST record and
;   0   !                                types.
;   0   !  08   18-aug-77       KGP    -Added the record definition for BLISS
;   0   !                                type Zero DST records.
;   0   !  09   13-sept-77      KGP    -Added the _IS_GLOBAL flag definition to
;   0   !                                MC_RECORDs and NT_RECORDs, and stopped
;   0   !                                using the special NT_TYPE value to indicate
;   0   !                                that a symbol is global.
;   0   !                              -Also moved the flag fields in MC_RECORDs
;   0   !                                around so that the records are 1 byte shorter.
;   0   !
;   0   !  10   15-09-77        CP     Added PC correlation record type.
;   0   !
;   0   !  11   20-sept-77      KGP    -Changed DST_TYP_LOWEST and _HIGHEST
;   0   !                                as now we handle so-called SRM types for
;   0   !                                RST building.
;   0   !  12   21-sep-77       KGP    -Increased MAX_SAME_SYMBLS from 10 t0 25 to
;   0   !                                try and fix a user-reported error which is
;   0   !                                caused when >10 symbols hash to the same value.
;   0   !  13   23-sep-77       KGP    -Changed the skeleton structure of LVT and SAT,
;   0   !                                and added comments herein to document this.
;   0   !  14   27-sep-77       KGP    -Added the non-mars LABEL DTYPE DSC$K_DTYPE_SLB
;   0   !                                to the DST type collection since we now (5X07)
;   0   !                                support that type.
;   0   !  15   28-sep-77       KGP    -Reorganized the SAT and LVT structs so that they
;   0   !                                are alligned wrt _NT_PTR and _VALUE/_LB so that
;   0   !                                they can share a common sort routine.
;   0   !  16   14-OCT-77       KGP    -Added the new data descriptor types,
;   0   !                                ARRAY_BNDS_DESC and SYM_VALUE_DESC.  Also
;   0   !                                added the ACCS_ sub-types in DST records.
;   0   !  17   27-oct-77       KGP    -We now use the MC_IS_GLOBAL bit in MC
;   0   !                                records  since we now have a 'dummy' MC
;     0089  0     !                                       record to hang globals off.
;     0090  0     !                                       -Also added INIT_RST_SIZE, and changed the
;     0091  0     !                                        values for SAT_MINIMUM and LVT_MINIMUM
;     0092  0     !   18   28-OCT-77        KGP           -Added MC_LANGUAGE field in MC records.
;     0093  0     !                                        Also set up NT_not_free, NT_free and MC_free
;     0094  0     !                                        fields, so that it is now clearer
;     0095  0     !                                        just how these 'common' (NT/MC) bits
;     0096  0     !                                        interrelate.
;     0097  0     !   19   01-nov-77        KGP           -Took away the docu and definition of
;     0098  0     !                                        the now-defunct DUPLICATION_VECTORs.
;     0099  0     !   20   02-nov-77        KGP           -Took the definition of the global literal
;     0100  0     !                                        DBG$_RST_BEGIN out of this file and put it
;     0101  0     !                                        it into DBGSTO.B32 because otherwise the
;     0102  0     !                                        librarian complains about multiply defined
;     0103  0     !                                        globals since this file is REQUIREd
;     0104  0     !                                        in several files.
;     0105  0     !   21   3-NOV-77         KGP           -Carol took out all references to A_LONGWORD
;     0106  0     !                                        and changed them to %upval.
;     0107  0     !                                       -I changed the proposed VALU_DESCRIPTOR field
;     0108  0     !                                        VALU_DST_ID to VALU_NT_PTR for the benefit
;     0109  0     !                                        of DBG$SET_SCOPE.
;     0110  0     !   22   9-nov-77         KGP           -Added the MC_NT_STORAGE field to MCs, and the
;     0111  0     !                                        definition of VECT_STOR_DESCs, which we
;     0112  0     !                                        now use to manage so-called 'vector storage'.
;     0113  0     !   23   14-nov-77        KGP           -NT records are now doubly-linked into hash chains.
;     0114  0     !   24   15-nov-77        KGP           -reorganized NTs and MCs so that NT names comes at
;     0115  0     !                                        the end so that NTs can be variable-sized.
;     0116  0     !   25   16-nov-77        KGP           -Added the new storage descriptors
;     0117  0     !                                        to MCs so that we can associate LVT
;     0118  0     !                                        and SAT storage with MCs.
;     0119  0     !                                       -Threw away the old notion of SAT_COUNT being
;     0120  0     !                                        a SAT_RECORD field for future use.
;     0121  0     !   26   17-nov-77        KGP           -Added the SAT and LVT control literals to
;     0122  0     !                                        support the new GET_NEXT_SAT/LVT routines.
;     0123  0     !   27   19-nov-77        KGP           -Added the field, SL_FREE_LINK, to SAT
;     0124  0     !                                        records. (and, implicitly, to LVT records).
;     0125  0     !   28   21-nov-77        KGP           -Added SL_ACCE_MORE, to be used by add_module
;     0126  0     !   29   22-nov-77        KGP           -Another field, STOR_LONG_PTRS, of each vector
;     0127  0     !                                        storage descriptor makes MCs 3 bytes longer.
;     0128  0     !   30   28-nov-77        KGP           -Added MC_IS_DYING field to MC records.
;     0129  0     !                                        SL_ACCE_MORE changed to SL_ACCE_FREE
;     0130  0     !   31   12-dec-77        KGP           -Added literal, RST_MAX_OFFSET
;     0131  0     !   32   13-DEC-77        KGP           -Added NT_IS_BOUNDED flag bit to NTs
;     0132  0     !   33   29-12-77         CP            Add a field name to nt_record to describe
;     0133  0     !                                       the value field of a GST name table entry.
;     0134  0     !   34   13-JAN-78        DAR           Removed the literals mars-module, fortran_module,
;     0135  0     !                                       and bliss_module and put them in DBGGEN.BEG
;     0136  0     !   35   02-feb-78        KGP           -New SIZE literals for overall DST characteristics
;     0137  0     !                                        so that we can avoid overflow due to
;     0138  0     !                                        too many MCs.
;     0139  0     !   36   15-feb-78        KGP           -New sub types for DSTR_ACCESS
;     0140  0     !   37   8-mar-78         KGP           -Stole this from DEBUG to use for TRACE
;     0141  0     !                                        so that the two could remain separate.
;     0142  0     !                                       -Commented out some of the DSC definitions
;     0143  0     !  38    09-NOV-78        DAR           Added new DST record type declarations.
;     0144  0     !                                        as they now appear in SYSDEF.REQ finally.
;     0145  0     !   39   06-JAN-81        DLP           Added new DST and SRM types
;   0146  0        --
;   0147  0      !++
;   0148  0      ! Since the DEBUG free-storage manager currently
;   0149  0      ! works in 'units', we define the following macro to
;   0150  0      ! convert a byte-unit quantity into whatever units it
;   0151  0      ! requires.  We expect to change the free-storage manager
;   0152  0      ! to work in byte units, so eventually this macro should
;   0153  0      ! just reduce to its actual parameter.  For now, however,
;   0154  0      ! it 'rounds up' to the smallest number of LONGWORDS
;   0155  0      ! which are required to contain the indicated number of
;   0156  0      ! bytes.
;   0157  0      !--
;   0158  0
;   0159  0      MACRO
; M 0160  0              RST_UNITS( bytes ) =
; M 0161  0
; M 0162  0                      ( ((bytes) + %upval-1)/%upval )
;   0163  0              %;
;   0164  0      !
;   0165  0      ! MACROS:
;   0166  0      !
;   0167  0
;   0168  0      MACRO
;   0169  0              YES_NO( question )
;   0170  0
;   0171  0                      ! Ask a question and return the Y/N answer.
; M 0172  0                      =
;   0173  0                      QUERY( UPLIT( %ASCIC question )) %;
;   0174  0
```

```
!++
! RST-Pointers
!
! So-called RST-pointers are referred to throughout the RST
! code.  They are simply the means of access to RST data
! structures, and we purposely talk of them as if they were
! their own TYPE so that we can change this implementation
! detail if/when we feel it is necessary.
!
! For now, RST-pointers are 16-bit items which are manipulated by
! the special RST storage routines DBG$RST_FREEZ and DBG$RST_RELEASE.
! No code outside of the RST-DST/DEBUG interface module knows
! anything more about the implementation of RST-pointers than that.
! (Other modules declare and use RST-pointers via macros, etc.)
!
! If any change is to be made to what RST-pointers actually
! are, there are only 2 criterion that the new ones much uphold:
! 1) RST-pointers must be storable in the NT, MC, SAT and LVT fields
! which are defined for them, and 2) they must be able to provide
! access to the RST_NT and RST_MC structures defined below.
!
! The following macro is provided so that one can declare REFs
! to such pointers.  Some code also applies %SIZE to this macro
! to get the size of an RST-pointer.  Note that no code should
! declare an occurrence of an RST-pointer, since we do not define
! that you can do anything meaningful with such a thing.  This is
! because we want to enforce the usage of REFs to the structures
! we declare to access RST data structures.  (e.g. we use
! "REF MC_RECORD" to say that we are declaring a pointer to
! an MC record.  REFs to MC_RECORDS also happen to be RST-pointers,
! but we don't want to build-in this coincidental characteristic.)
!--

MACRO
        RST_POINTER = VECTOR[1,WORD] %;

```

```
!++
! Pathnames
!
! Symbols in DEBUG are actually made up of sequences of
! symbols or "elements".  The concatenation of such
! elements, along with the element separation character (\), make
! up a so-called pathname because the sequence represents the
! path which one must make thru RST data structures to get to
! the desired symbol.
!
! We represent strings internal to DEBUG by passing around
! so-called counted string pointers.  They are simply LONGWORD
! pointers to a count byte followed by that many characters.
! The CS_POINTER macro allows us to declare occurrences, REFs,
! and take the %SIZE of this type of datum.
!
! Pathnames, then, are represented with vectors of CS_POINTERS.
! Like duplication vectors, they terminate with a 0 entry
! for programming ease, but also have a maximum size so that
! we can declare them LOCALly.
!
! The following macros are used in declarations to not build-in
! the above conventions.
!--

MACRO
                                ! DEBUG tells the RST module about ASCII
                                ! strings by passing a counted string pointer.
        CS_POINTER      = REF VECTOR[1,BYTE] %;

                                ! Symbol pathnames are 0-ended vectors
                                ! of CS_POINTERs.  There is a maximum
                                ! length to pathnames so that routines can
                                ! declare LOCAL vectors of pathname pointers.
LITERAL
        MAX_PATH_SIZE   = 10;
MACRO
        PATHNAME_VECTOR = VECTOR[ MAX_PATH_SIZE +1, %SIZE(CS_POINTER) ] %;
```

```
!+
! Overall Characteristics of the RST/DST, etc.
!-

!+
! The DEBUG Runtime Symbol Table (RST) free-storage area
! begins at a fixed virtual address.  This LITERAL is used
! directly by some of the RST structures since RST-pointers
! need this information.
!-

LITERAL
        ! The RST is a fixed size - but this fact is only
        ! used to allow us to set the other _SIZE literals
        ! below in such a way that we can say that the various
        ! RST uses will be percentages of the total size.

        RST_TOTAL_SIZE          = 65000,        ! RST is 65K bytes.

        ! When we SET MODULe, we will not take absolutely
        ! all the free storage that is available.  Instead, we
        ! will keep adding modules so long as the amount of
        ! free storage left (before we add the module) is
        ! atleast RST_AVAIL_SIZE bytes.

        RST_AVAIL_SIZE          = 3000, ! Storage left over for DEBUG itself

        ! During RST init, we take space for only as many MCs
        ! as will leave RST_MODU_SIZE bytes for subsequent
        ! SET MODUles.  Currently the MC space is 50% of the RST.

        RST_MODU_SIZE           = (RST_TOTAL_SIZE-RST_AVAIL_SIZE)/2,

        ! The SAT and LVT are allocated contiguous storage
        ! on a per-module basis by tallying up the number of
        ! SAT/LVT entries needed for that module.
        ! The following two minimums are used to begin the
        ! tally so that the tables will actually be somewhat
        ! larger than what the MC data implies.  The SAT and LVT
        ! minimums must be at least 1 so that we will never ask
        ! the free storage manager for 0 bytes.

        SAT_MINIMUM     = 10,   ! Minimum number of SAT entries.
        LVT_MINIMUM     = 10,   ! Minimum number of LVT entries.

        ! The NT, however, has no such fixed size.  MC statistics
        ! gathering tallies up the number of NT entries, though;
        ! we begin such a tally at NT_MINIMUM.

        NT_MINIMUM      = 0,    ! Minimum number of NT entries.

        ! We will use byte indices to fetch RST-pointers to the NT
        ! from the NT hash vector.  This vector, then, must contain
        ! NT_HASH_SIZE entries, each of which must be large enough
        ! to store an RST-pointer.  See BUILD_RST() in DBGRST.B32
        ! Also see field NT_FORWARD of the NT record definition,
        ! and the corresponding warning in the routine UNLINK_NT_RECS.
```

```

                NT_HASH_SIZE    = %X'FF',        ! NT hash vector size.

                ! We will never print "symbol+offset" when the
                ! upper bound for "symbol" is 0 and when
                ! the offset is greater than RST_MAX_OFFSET

                RST_MAX_OFFSET          = %X'100';


        !+
        ! Since scope definitions are recursive, we must
        ! stack ROUTINE BEGINs in the routine ADD_MODULE.
        ! It is no coincidence that this stack limit is the
        ! same as the limit on the length (in elements) of
        ! symbol pathnames.
        !-

        LITERAL
                MAX_SCOPE_DEPTH = MAX_PATH_SIZE;        ! Routines can be nested to a maximum depth.
```

```
!++
!  Descriptors
!
!  Just as the SRM defines various 'system wide' descriptor
!  formats, the RST modules use a few more descriptors
!  of its own invention.  They are as follows:
!--

!++
!  Value Descriptors
!
!  Value Descriptors are used to pass around all needed
!  information about a value which has been obtained
!  from the RST data base.  For now they are simply
!  2-longword blocks:
!
!               !-----longword------!
!
!                         !----------!
!                         ! NT_PTR   !
!               !-------------------!
!               !  actual value     !
!               !-------------------.
!
!  Value Descriptors must be accessed via the following
!  field names.
!--

FIELD
        VALU_FIELD_SET =
    SET
        VALU_NT_PTR     = [  0,0,16,0 ],        ! Associated NT pointer.
        VALU_VALUE      = [  2,0,32,0 ]         ! The actual value.
    TES;

!+
! Declare an occurrence or REF to a VALUE_DESCRIPTOR
! via the following macros.
!-

LITERAL
        VALU_DESC_SIZE  = 8;            ! Each one is 2 longwords long.

MACRO
        VALU_DESCRIPTOR = BLOCK[ VALU_DESC_SIZE, BYTE ] FIELD( VALU_FIELD_SET ) %;
```

```
;     0372  0       !++
;     0373  0       !  Array Bounds Des:riptor
;     0374  0       !
;     0375  0       !  An array bounds Descriptor is used to pass around all needed
;     0376  0       !  information about an array and its associated dimensions.
;     0377  0       !  Like VALU_DESCRIPTORs, they are simply 2-longword blocks,
;     0378  0       !  but this might change.
;     0379  0       !
;     0380  0       !       !------longword-----!
;     0381  0       !
;     0382  0       !       !-------------------!
;     0383  0       !       ! address of array  !
;     0384  0       !       !-------------------!
;     0385  0       !       ! length of array   !
;     0386  0       !       !-------------------!
;     0387  0       !
;     0388  0       !  Such Descriptors must be accessed via the followi.g
;     0389  0       !  field names.
;     0390  0       !--
;     0391  0
;     0392  0       FIELD
;     0393  0               ARRAY_BNDS_SET =
;     0394  0           SET
;     0395  0               ARRAY_ADDRESS   = [  0,0,32,0 ],        ! Beginning address of array.
;     0396  0               ARRAY_LENGTH    = [  4,0,32,0 ]         ! Size, in bytes, of array.
;     0397  0           TES;
;     0398  0
;     0399  0       !+
;     0400  0       ! Declare an occurrence or REF to an array bounds
;     0401  0       ! descriptor via the following macros.
;     0402  0       !-
;     0403  0
;     0404  0       LITERAL
;     0405  0               ARRAY_BNDS_SIZE = 8;            ! Each one is 2 longwords long.
;     0406  0
;     0407  0       MACRO
;     0408  0               ARRAY_BNDS_DESC = BLOCK[ ARRAY_BNDS_SIZE, BYTE ] FIELD( ARRAY_BNDS_SET ) %;
```

```
!++
!  Vector Storage Descriptors
!
!  So-called "vector storage" is the storage which
!  we allocate in relatively large chunks for the
!  explicit purpose of subsequently re-allocating the same storage
!  to someone else in smaller, variable-sized chunks.
!
!  This facility has been implemented to interface between
!  the way that the standard DEBUG storage manager
!  works, with the way that the RST routines really
!  want to 'allocate' storage.  We satisfy the former by
!  only asking for large chunks (and paying the
!  associated overhead), and we satisfy the latter by
!  'doling' out small-sized chunks with little overhead.
!  We can do this because we never have to freeup these
!  chunks so don't have to store the would-be-needed pointers, etc.
!
!       !--%size(RST_POINTER)--!
!
!       !-----(i.e. word)------!
!
!                   !----------!
!                   ! PTR type !
!       !----------------------!
!       ! beginning of STORage !
!       !----------------------!
!       !    end of STORage    !
!       !----------------------!
!       ! nxt free rec in STOR !
!       !----------------------!
!
!  Such descriptors are accessed via the
!  following field names.
!
!  The 'begin' field is the one which various routines
!  look at to decide if the field descriptor is valid.
!--

FIELD
        STOR_DESC_SET =
    SET
        STOR_LONG_PTRS  = [  0,0, 8,0 ],        ! Pointer type. 1 => full word pointers,
                                                !   0 => RST-pointer access.
        STOR_BEGIN_RST  = [  1,0,16,0 ],        ! RST pointer to beginning of storage.
        STOR_END_RST    = [  3,0,16,0 ],        ! RST pointer to end of storage.
        STOR_MARKER     = [  5,0,16,0 ]         ! Current place in storage.
                                                ! (RST pointer to next available byte).
    TES;

!+
! Declare an occurrence or REF to a vector storage
! descriptor via the following macros.
!-

LITERAL
        STOR_DESC_SIZE = 7;             ! 3 RST pointers take 6 bytes;
```

```
:     0466  0                                         !   the pointer-type byte takes 1 more.
:     0467  0
:     0468  0      MACRO
:     0469  0              VECT_STORE_DESC = BLOCK[ STOR_DESC_SIZE, BYTE ] FIELD( STOR_DESC_SET ) %;
```

```
0     !++
0     !  The Module Chain (MC) is a chain of fixed-size
0     !  records each of which has an RST_MC structure:
0     !
0     !        !<byte><byte>!<byte><byte>!
0     !
0     !        !------------!------------!
0     !        !x!flags!type! Next MC    !
0     !        !------------!------------!
0     !        !       DST Pointer       !
0     !        !------------!------------!
0     !        !   number of NT entries  !
0     !        !------------!------------!
0     !        ! first name bytes ! count!
0     !        !------------!------------!
0     !        !     more  name  bytes   !
0     !        !------------!------------!
0     !        !     more  name  bytes   !
0     !        !------------!------------!
0     !        !     more  name  bytes   !
0     !        !------------!------------!--------------!
0     !        ! vector storage descriptor for NT recs  !
0     !        !------------!------------!--------------!
0     !        ! vector storage descriptor for SAT recs !
0     !        !------------!------------!--------------!
0     !        ! vector storage descriptor for LVT recs !
0     !        !------------!------------!--------------!
0     !        !   number of SAT entries !
0     !        !------------!------------!
0     !        !   number of LVT entries !
0     !        !------------!------------!
0     !
0     !   The reason for using our own structure here,
0     !   (instead of a BLOCK), is because we access
0     !   MC records with RST-pointers.
0     !--
0
0     LITERAL
0                                       ! MC records are fixed-size.
0             RST_MC_SIZE     = 57;     ! Each one takes this many bytes.
0
0     STRUCTURE
0             RST_MC [ off, pos, siz, ext; N=1, unit=1 ] =
0                     [ N * RST_MC_SIZE ]
0
1                     BEGIN
2                     (
2                     EXTERNAL LITERAL TBK$_RST_BEGIN;
2                     RST_MC + TBK$_RST_BEGIN
1                     ) + off*unit
0                     END <pos, siz, ext>
0                     ;
0
0     !+
0     !  MC records have the following fields.
0     !-
0
```

```
0       FIELD
0               MC_FIELD_SET =
0           SET
0               ! **** Some fields (up to NAME_ADDR) must be alligned
0               !      with the corresponding ones in RST_NT structures.
0
0               MC_NEXT         = [  0,0,16,0 ],        ! Next MC record in chain.
0               MC_TYPE         = [  2,0, 8,0 ],        ! DST record type byte.
0                                                       ! Must be DSC$K_DTYPE_MOD
0               MC_IS_GLOBAL    = [  3,0, 1,1 ],        ! 0, for 'normal' MCs.  1 for the
0                                                       !  MC record we 'hang' globals off.
0               MC_IN_RST       = [  3,1, 1,1 ],        ! Whether or not this module
0                                                       !   has been initialized into the RST.
0               MC_IS_MAIN      = [  3,2, 1,1 ],        ! Whether or not this module
0                                                       !   contains the program's transfer
0                                                       !   address.
0               MC_LANGUAGE     = [  3,3, 3,0 ],        ! 3-BIT encoding of the language
0                                                       ! which the module is written in.
0               MC_IS_DYING     = [  3,6, 1,0 ],        ! Vector storage for this MC is
0                                                       !  about to be freed up.
0               MC_not_free             = [  3,7, 1,0 ],        ! Used in NTs only.
0               MC_DST_START    = [  4,0,32,0 ],        ! Record ID of first record for this module.
0               MC_NAMES        = [  8,0,32,1 ],        ! Number of NT records required.
0               MC_NAME_CS      = [ 12,0, 8,0 ],        ! Name of Module is a counted string.
0                                                       ! A dotted reference to this field picks
0                                                       ! up the count, an undotted one
0                                                       ! addresses the counted string.
0               MC_NAME_ADDR    = [ 13,0, 8,0 ],        ! The name string itself.  An undotted
0                                                       ! reference to this field addresses
0                                                       ! only the MC name, a dotted reference
0                                                       ! picks up the 1st character of the name.
0
0               ! *** leave up to byte 27 inclusive for _NAME_ field.
0
0               MC_NT_STORAGE   = [ 28,0, 8,0 ],        ! Vector storage descriptor for NT records.
0                                                       ! A direct reference to this field is
0                                                       !  equivalent to the STOR_LONG_PTRS
0                                                       !  field of the storage descriptor.
0
0               ! *** leave up to byte 34 inclusive for _NT_STORAGE field.
0
0               MC_SAT_STORAGE  = [ 35,0, 8,0 ],        ! Vector storage descriptor for SAT records.
0                                                       ! A direct reference to this field is
0                                                       !  equivalent to the STOR_LONG_PTRS
0                                                       !  field of the storage descriptor.
0
0               ! *** leave up to byte 41 inclusive for _SAT_STORAGE field.
0
0               MC_LVT_STORAGE  = [ 42,0, 8,0 ],        ! Vector storage descriptor for LVT records.
0                                                       ! A direct reference to this field is
0                                                       !  equivalent to the STOR_LONG_PTRS
0                                                       !  field of the storage descriptor.
0
0               ! *** leave up to byte 48 inclusive for _LVT_STORAGE field.
0
0               MC_STATICS      = [ 49,0,32,1 ],        ! Number of SAT records required.
0               MC_LITERALS     = [ 53,0,32,1 ]         ! Number of LVT records required.
```

```
;      0584  0          TES;
;      0585  0
;      0586  0      !+
;      0587  0      ! You declare an occurrence or REF of an MC datum via:
;      0588  0      !-
;      0589  0
;      0590  0      MACRO
;      0591  0              MC_RECORD       = RST_MC[ RST_MC_SIZE, BYTE ] FIELD( MC_FIELD_SET ) %;
```

```
0       !++
0       !  The Name Table (NT) is a set of doubly-linked records
0       !  with the following format:
0       !
0       !         !<byte><byte>!<byte><byte>!
0       !
0       !         !------------!------------!
0       !         !x!flags!type! Next NT    !
0       !         !------------!------------!
0       !         !        DST Pointer      !
0       !         !------------!------------!
0       !         ! back hash  !  forw hash !
0       !         !------------!------------!
0       !         ! first name bytes ! count!
0       !         !------------!------------!
0       !         !     more  name  bytes   !
0       !         !------------!------------!
0       !         !     more  name  bytes   !
0       !         !------------!------------!
0       !         !     more  name  bytes   !
0       !         !------------!------------!
0       !
0       !  Since access to such records will be via so-called RST-pointers,
0       !  (16-bit pointers which we always add a global to before using),
0       !  we define the following structure to localize this implementation
0       !  detail.
0       !++
0
0       LITERAL
0               RST_NT_OVERHEAD = 13,   ! Number of bytes in NT record excluding those
0                                       !    taken up by the name.  (So that this
0                                       !    number + .NT_PTR[ NT_NAMES_CS ] gives
0                                       !    the length of the NT record in bytes.)
0                                       !  (This is solely for the benefit of routines
0                                       !    unlink_nt_recs, add_nt, and add_gst_nt.)
0               RST_NT_SIZE     = 28;   ! A static NT record would take a max # of bytes.
0                                       ! (Dynamically-allocated ones usually take less).
0
0       STRUCTURE
0               RST_NT [ off, pos, siz, ext; N=1, unit=1 ] =
0                       [ N * RST_NT_SIZE ]
0
1                       BEGIN
2                       (
2                       EXTERNAL LITERAL TBK$_RST_BEGIN;
2                       RST_NT + TBK$_RST_BEGIN
1                       ) + off*unit
0                       END <pos, siz, ext>
0                       ;
0
0       !+
0       !  Access to an NT chain is via a 'hash' vector.
0       !  Conceptually, this is a vector of RST-pointers, and
0       !  we define the following macro to declare REFs or occurrences
0       !  of these elements.  (because we may decide
0       !  to change their representation)
0       !-
```

```

MACRO
        NT_HASH_RECORD = VECTOR[1,WORD] %;


!+
!  NT records have the following fields.
!
!  Note that NT_FORWARD must be the first
!  field in the record so that unlink_nt_recs
!  can overlay NT_FORWARD and a given entry
!  in the NT_HASH_VECTOR.
!-

FIELD
        NT_FIELD_SET =
    SET
        ! **** Some fields (up to NAME_ADDR) must be alligned
        !      with the corresponding ones in RST_MC structures.

        NT_FORWARD      = [  0,0,16,0 ],        ! Next NT record in hash chain.
                                                !  FORWARD must be first.  See above.
        NT_TYPE         = [  2,0, 8,0 ],        ! DST record type byte, (from SRM),
                                                ! or unused if NT_IS_GLOBAL.
        NT_IS_GLOBAL    = [  3,0, 1,1 ],        ! Whether or not the symbol is GLOBAL.
        NT_not_free     = [  3,1, 6,0 ],        ! Used in MCs but not in NTs.
        NT_IS_BOUNDED   = [  3,7, 1,0 ],        ! Unsed in NTs only.  => symbol's
                                                !   LB and UB are not 0.
        NT_DST_PTR      = [  4,0,32,0 ],        ! Pointer to associated DST record.
        NT_GBL_VALUE    = [  4,0,32,0 ],        ! Value of symbol when it
                                                ! is bound only to a GST record.
        NT_UP_SCOPE     = [  8,0,16,0 ],        ! Pointer to NT record for symbol
                                                !   that is 'above' this as far as
                                                !   scope is concerned.
        NT_BACKWARD     = [ 10,0,16,0 ],        ! Backward NT hash chain link.
        NT_NAME_CS      = [ 12,0, 8,0 ],        ! Name of symbol is a counted string.
                                                ! A dotted reference to this field picks
                                                ! up the count, an undotted one
                                                ! addresses the counted string.
        NT_NAME_ADDR    = [ 13,0, 8,0 ]         ! The name string itself.  An undotted
                                                ! reference to this field addresses
                                                ! only the MC name, a dotted reference
                                                ! picks up the 1st character of the name.

    TES;


!+
! You define an occurrence or REF to an NT record via:
!-

MACRO
        NT_RECORD       = RST_NT[ RST_NT_SIZE, BYTE] FIELD( NT_FIELD_SET ) %;

```

```
!++
! The Static Address Table (SAT) is a vector of
! fixed-size records (blocks) with the following
! format:
!
!       !<byte><byte>!<byte><byte>!
!
!                    !------------!
!                    ! NT-pointer !
!       !------------!------------!
!       ! lower bound address     !
!       !------------!------------!
!       ! upper bound address     !
!       !------------!------------!
!
! The lower and upper bound address fields contain the
! beginning and ending virtual addresses which were
! bound to the symbol by the linker.
! The NT-pointer field contains an RST-pointer into
! the name table (NT) for the NT entry which corresponds
! to this symbol.
!
! Overall Structure:
!
!       Logically, the SAT is a sequence of fixed-size records
! ordered on the _UB field so that we can search them sequentially.
! Physically the storage is actually discontiguous,
! space being associated with the module the space was allocated
! on behalf of.  Sequentially access to the SAT is that which
! is provided and defined by GET_NEXT_SAT in the following
! manner:
!
!       1) call GET_NEXT_SAT( SL_ACCE_INIT )
!
!          to set up to begin scanning the SAT
!  then
!       2) call ptr = GET_NEXT_SA ( access_type )
!
!          to have 'ptr' set to the next SAT record, where
!          the notion of 'next' is defined by 'access_type'.
!
!       Currently 3 access_types are defined.  _RECS and _SORT both
!  ask for the next sequential record in a logical sense.  (i.e.
!  records marked for deletion are quietly skipped over).  The ending
!  criterion for _RECS access is that there are no more records left,
!  while _SORT access, expected to be used with the 'shell' sort,
!  ends each time like _RECS does but at that time causes
!  the access routine to restore the context which it saved after
!  the last _SORT call so that subsequent _RECS calls scan from
!  where they left off last time.
!  In both cases 0 is returned in 'ptr' when
!  there are no more records for the indicated access type.
!
!       For the type of sequential access we need when moving
!   endangered SAT/LVT records to storage not _DYING,
!   we also define a third access mode called SL_ACCE_FREE.
!   This mode asks for modules _IN_RST AND _IS_DYING to
!   be skipped over so that only pointers to 'safe' records
!   are returned.
!
!       In all cases, the same _INIT code must be used to
!   'start off' the access sequence, and no concurrent accessing
!   is allowed except for the limited type supported via RECS/SORT.
!--

LITERAL
        SL_ACCE_INIT    = 0,    ! See above.   "SL" --> SAT/LVT
        SL_ACCE_RECS    = 1,
        SL_ACCE_SORT    = 2,
        SL_ACCE_FREE    = 3;

!+
! SAT/LVT Correspondence
!
!       While the SAT and LVT are as similar in structure as they
! are now, the two are manipulated by the same routines as much
! as possible.  This will remain OK as long as the fields which
! must correspond still do.  See the "Implicit Inputs" section
! of the common routines for details.
!-

!+
!  SAT records have the following fields.
!-

FIELD
        SAT_FIELD_SET =
    SET
        ! **** The SAT and LVT structures must be alligned so that
        !      the _NT_PTR fields match, and so that the _LB and _VALUE
        !      fields overlap.  The latter must be true only as long
        !      as the two share a common sort routine which relies on
        !      this allignment.  The former must be true as long as
        !      the two share any routines which access SAT_NT_PTR
        !      (COMPRES_SAT_LVT, DELE_SAT_LVT, etc).

        SAT_NT_PTR      = [  0,0,16,0 ],        ! Points to associated NT record.
        SAT_LB          = [  2,0,32,0 ],        ! Lower bound static address.

        SAT_UB          = [  6,0,32,0 ]         ! Upper bound static address.
    TES;

!+
! You declare an occurrence or REF of an SAT datum via
! the macro, SAT_RECORD.  If you want the %SIZE of
! a pointer to such a thing, use %size( SAT_POINTER ).
!-

LITERAL
        RST_SAT_SIZE    = 10;   ! Each SAT record takes this many bytes.

MACRO
        SAT_RECORD      = BLOCK[ RST_SAT_SIZE, BYTE ] FIELD( SAT_FIELD_SET ) %,
;    0817  0             SAT_POINTER     = REF BLOCK[ RST_SAT_SIZE, BYTE ] %;
```

```
!++
! The Literal Value Table (LVT) is a vector of
! fixed-size LVT records each of which has the
! following format:
!
!       !<byte><byte>!<byte><byte>!
!
!                    !------------!
!                    ! NT-pointer !
!       !------------!------------!
!       !      literal value      !
!       !------------!------------!
!
! The value field contains the longword value
! which is bound to the literal.
! The NT-pointer is an RST-pointer to the NT record
! for this symbol.
!
! Overall Structure:
!
!       Logically, the LVT is a sequence of fixed-size records
! ordered on the _VALUE field so that we can search them sequentially.
! Physically the storage is actually discontiguous,
! space being associated with the module the space was allocated
! on behalf of.  Sequentially access to the LVT is that which
! is provided and defined by GET_NEXT_LVT using the same
! control literals and the same mechanisms as are described
! for the SAT, above.
!--

!+
!  LVT records have the following fields.
!-

FIELD
        LVT_FIELD_SET =
    SET
        ! **** The SAT and LVT structures must be alligned so that
        !       the _NT_PTR fields match, and so that the _LB and _VALUE
        !       fields overlap.  The latter must be true only as long
        !       as the two share a common sort routine which relies on
        !       this allignment.  The former must be true as long as
        !       the two share any routines which access SAT_NT_PTR
        !       (COMPRES_SAT_LVT, DELE_SAT_LVT, etc).

        LVT_NT_PTR      = [  0,0,16,0 ],        ! Pointer to associated NT record.
        LVT_VALUE       = [  2,0,32,0 ]         ! Value bound to the literal.
    TES;

!+
! You declare an occurrence or REF of an LVT datum via:
!-

LITERAL
        RST_LVT_SIZE    = 6;    ! Each LVT record takes this many bytes.

MACRO
```

```
;   0875  0             LVT_RECORD      = BLOCK[ RST_LVT_SIZE, BYTE ] FIELD( LVT_FIELD_SET ) %;
```

```
!++
! BLISS uses 'non-standard' DST records to encode
! most of its local symbol information.  These records
! are like most DST records except that the TYPE
! information is variable-sized.
!--

FIELD
        BLZ_FIELD_SET =
    SET
        BLZ_SIZE        = [  0,0, 8,0 ],        ! First byte is record size in bytes.

                        ! The next byte contains DSC$K_DTYPE_Z, or we
                        ! wouldn't be applying this structure to a given
                        ! DST record.

        BLZ_TYP_SIZ     = [  2,0, 8,0 ],        ! Type info takes up this
                                                !  many bytes.
        BLZ_TYPE        = [  3,0, 8,0 ],        ! Which type of type Zero
                                                !  this corresponds to.
        BLZ_ACCESS      = [  4,0, 8,0 ],        ! Access field.
        BLZ_STRUCT      = [  5,0, 8,0 ],        ! Type of STRUCTURE reference.

                ! **** The following only work when BLZ_TYP_SIZ is 3.

        BLZ_VALUE       = [  6,0,32,0 ],        ! DST VALUE field.
        BLZ_NAME_CS     = [ 10,0, 8,0 ],        ! The symbol name is a counted string.
                                                ! A dotted reference to this field
                                                ! picks up the count, an undotted
                                                ! one addresses the counted string.
        BLZ_NAME_ADDR   = [11,0, 8,0 ]          ! The name string itself.  An undotted
                                                ! reference is the address of the name,
                                                ! a dotted one is the 1st character.
    TES;

!+
! You declare a REF to a BLZ_DST datum via:
!-

LITERAL
        BLZ_REC_SIZ     = 38;   ! Each DST record is at most 38 bytes long.

MACRO
        BLZ_RECORD = BLOCK[ BLZ_REC_SIZ, BYTE] FIELD( BLZ_FIELD_SET ) %;


!+
! The type zero sub types,
! as defined in CP0021.MEM,
! must be within the following
! range.
!-

LITERAL

        ! Type Zero Sub-Types:

```

```
;     0933  0              BLZ_LOWEST       = 1,    ! Lowest variable type we support.
;     0934  0
;     0935  0              BLISS_Z_FORMAL   = 1,    ! Description of a ROUTINE formal.
;     0936  0              BLISS_Z_SYMBOL   = 2,    ! A BLISS LOCAL symbol.
;     0937  0
;     0938  0              BLZ_HIGHEST      = 2;    ! Highest variable type we support.
;     0939  0
;     0940  0       ! End of TBKRST.REQ
;     0941  0       !--
```

```
!++
!       TBKGEN.REQ - require file for vax/vms TRACE facility
!
!       MODIFIED BY:    Dale Roedger 29 June 1978
!
!       This file was taken from DBGGEN.REQ on 8 March 1978
!
!       29-JUN-78       DAR     Added literals for COBOL and BASIC.
!--

literal
        tty_out_width   =132,           ! standard TTY output width.
        fatal_bit       =4,             ! mask for fatal bit in error codes
        add_the_offset  =1,             ! add offset to value
        sub_the_offset  =0,             ! subtract offset from value
        upper_case_dif  ='a' - 'A',     ! difference between ASCII representation of upper and lower case
        ascii_offset    =%O'60',        ! offset from numeric value to ASCII value

        !++
        ! ASCII character representations
        !--
        linefeed        =%O'12',        ! ASCII representation of linefeed
        carriage_ret    =%O'15',        ! ASCII representation of carriage return
        asc_at_sign     =%ASCII '@',    ! ASCII representation of an at sign
        asc_clos_paren  =%ASCII ')',    ! ASCII representation of closed parenthesis
        asc_comma       =%ASCII ',',    ! ASCII representation of a comma
        asc_minus       =%ASCII '-',    ! ASCII representation of a minus sign
        asc_open_paren  =%ASCII '(',    ! ASCII representation of open parenthesis
        asc_percent     =%ASCII '%',    ! ASCII representation of a percent sign
        asc_period      =%ASCII '.',    ! ASCII representation of a period
        asc_plus        =%ASCII '+',    ! ASCII representation of a plus sign
        asc_pounds      =%ASCII '#',    ! ASCII representation of a pounds sign
        asc_quote       =%ASCII '''',   ! ASCII representation of a quote character
        asc_space       =%ASCII ' ',    ! ASCII representation of a space
        asc_sq_clo_brak =%ASCII ']',    ! ASCII representation of a closed square bracket
        asc_sq_opn_brak =%ASCII '[',    ! ASCII representation of an open square bracket
        asc_tab         =%ASCII '       ',      ! ASCII representation of a tab
        asc_up_arrow    =%ASCII '^',    ! ASCII representation of an up arrow


        not_an_exc      = 0,            ! line number searching for pc
        trap_exc        = 1,            ! pc of trap searching for line number
        fault_exc       = 2,            ! pc of fault searching for line number
        lookup_exc      = 3;            ! Like TRAP only don't do val_to_sym again.

literal
        !++
        ! names of module types
        !--
        macro_module    = 0,            ! module written in MACRO
        fortran_module  = 1,            ! module written in FORTRAN
        bliss_module    = 2,            ! module written in BLISS
        cobol_module    = 3,            ! module written in COBOL
        basic_module    = 4,            ! module written in BASIC
        pli_module      = 5,            ! module written in PLI
        pascal_module   = 6,            ! module written in PASCAL
        c_module        = 7,            ! module written in C
```

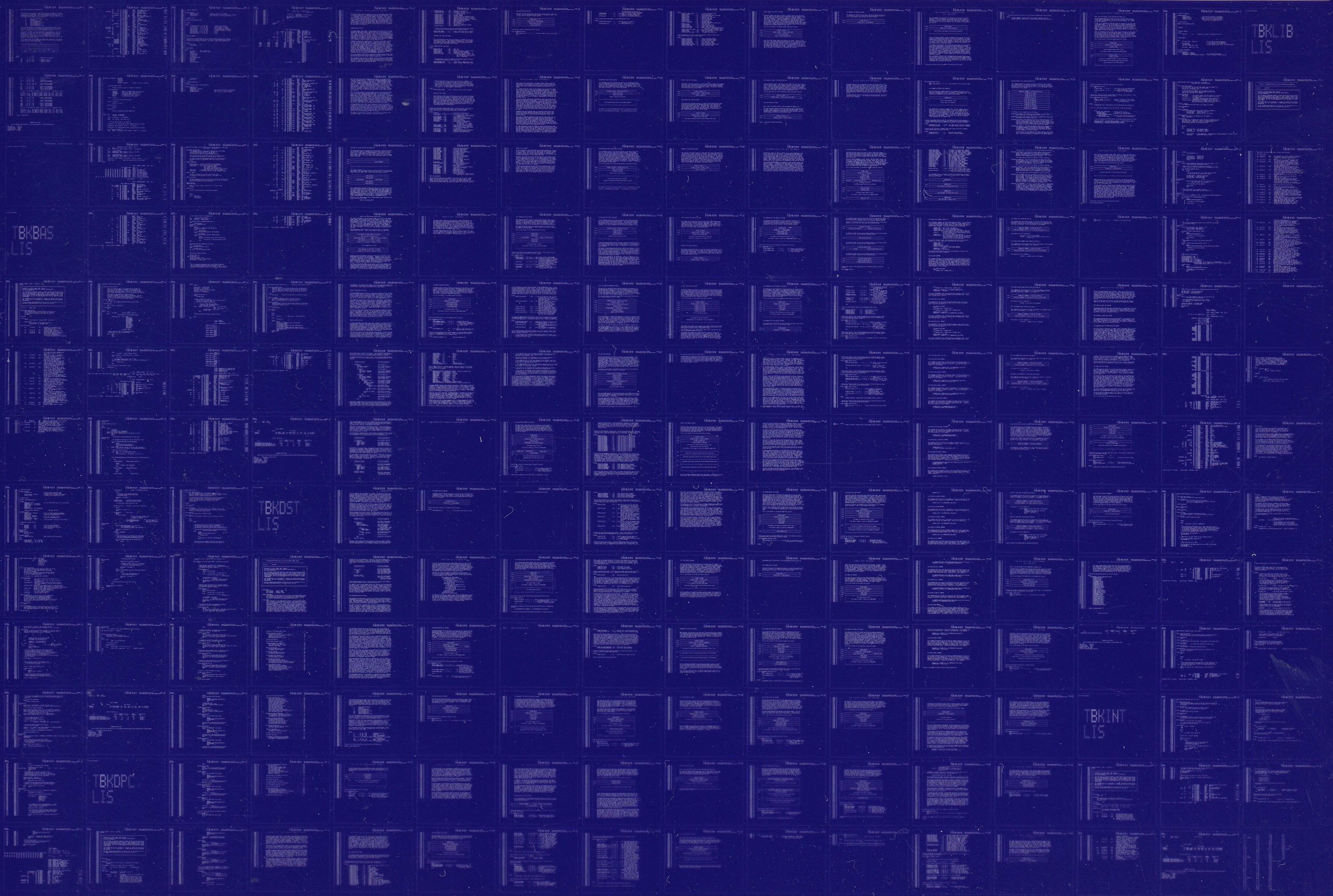
```
;   0999  0            rpg_module      = 8,            ! module written in RPG
;   1000  0            ada_module      = 9,            ! module written in ADA
;   1001  0
;   1002  0
;   1003  0            !++
;   1004  0            ! language names and MAX_LANGUAGE
;   1005  0            !--
;   1006  0            macro_lang      =macro_module,  ! MACRO
;   1007  0            fortran_lang    =fortran_module,! FORTRAN
;   1008  0            bliss_lang      =bliss_module,  ! BLISS
;   1009  0            cobol_lang      =cobol_module,  ! COBOL
;   1010  0            basic_lang      =basic_module,  ! BASIC
;   1011  0            pli_lang        =pli_module,    ! PLI
;   1012  0            pascal_lang     =pascal_module, ! PASCAL
;   1013  0            c_lang          =c_module,      ! C
;   1014  0            rpg_lang        =rpg_module,    ! RPG
;   1015  0            ada_lang        =ada_module,    ! ADA
;   1016  0
;   1017  0            max_language    = 9;            ! languages 0 - 9
;   1018  0
;   1019  0
;   1020  0        !  END OF TBKGEN .REQ
;   1021  0        !--
```

```
!+
!       TRACE Version 1.0 - Kevin Pammett, 8-march-1978
!
!       TBKSER.REQ - definitions file for calling system services
!
!       Added a few macros and literals from DEBUG require files
!       we don't want to drag along with TRACE.
!-


MACRO
        true = 1 %,
        false = 0 %,
        repeat = while(1) do%,

        $fao_stg_count (string) =
                !+
                ! $fao_stg_count makes a counted byte string out of an ASCII string.
                ! This macro is useful to transform an fao control string into the
                ! address of such a string, whose first byte contains the length of
                ! the string in bytes.
                !-
                UPLIT BYTE (%CHARCOUNT (string), %ASCII string)%,

        $fao_tt_out (ctl_string) [] =
                !+
                ! $fao_tt_out constructs a call to fao with a control string,
                ! and some arguments to the control string.
                ! This formatted string is then output to the output device.
                !-
                tbk$fao_out ($fao_stg_count (ctl_string), %REMAINING)%,

        $fao_tt_cas_out (ctl_string_adr) [] =
                !+
                ! $fao_tt_cas_out constructs a call to fao with the address of a
                ! control string, and some arguments to the control string. This formatted
                ! string is then output to the terminal.
                !-
                tbk$fao_out (ctl_string_adr, %REMAINING)%,

        $fao_tt_ct_out (ctl_string) =
                !+
                ! $fao_tt_ct_out constructs a call to fao with a control string.
                ! This formatted string is then output to the terminal.
                !-
                tbk$fao_out ($fao_stg_count (ctl_string))%,

        $fao_tt_ca_out (ctl_string_adr) =
                !+
                ! $fao_tt_ca_out calls fao with the address of a
                ! control string. This formatted string is then output
                ! to the output device.
                !-
                tbk$fao_out (ctl_string_adr)%;


! END OF TBKSER.REQ
```

```
;   1079  0     !--




;                                        COMMAND QUALIFIERS

;       BLISS/LIBRARY=LIB$:TBKLIB.L32/LIST=LIS$:TBKLIB.LIS SRC$:TBKLIB.REQ

; Run Time:          00:06.3
; Elapsed Time:      00:07.6
; Lines/CPU Min:     10308
; Lexemes/CPU-Min:   16203
; Memory Used:  35 pages
; Library Precompilation Complete
```

0401 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY



0402 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY