


```

SSSSSSSS TTTTTTTTT RRRRRRRR MM MM AAAAAA CCCCCCCC RRRRRRRR 000000 SSSSSSSS
SSSSSSSS TTTTTTTTT RRRRRRRR MM MM AAAAAA CCCCCCCC RRRRRRRR 000000 SSSSSSSS
SS      TT      RR      RR MMMM MMMM AA      AA CC      RR      RR 00      00 SS
SS      TT      RR      RR MMMM MMMM AA      AA CC      RR      RR 00      00 SS
SS      TT      RR      RR MM MM MM AA      AA CC      RR      RR 00      00 SS
SS      TT      RRRRRRRR MM MM MM AA      AA CC      RRRRRRRR 00      00 SSSSSS
SSSSSS SS      TT      RRRRRRRR MM MM MM AA      AA CC      RRRRRRRR 00      00 SSSSSS
SS      TT      RR RR MM MM MM AAAAAAAAAA CC      RR RR 00      00 SS
SS      TT      RR RR MM MM MM AAAAAAAAAA CC      RR RR 00      00 SS
SS      TT      RR RR MM MM MM AA      AA CC      RR RR 00      00 SS
SSSSSSSS TT      RR RR MM MM MM AA      AA CC      RR RR 00      00 SSSSSSSS
SSSSSSSS TT      RR RR MM MM MM AA      AA CC      RR RR 00      00 SSSSSSSS

```

```

RRRRRRRR EEEEEEEEE QQQQQQ
RRRRRRRR EEEEEEEEE QQQQQQ
RR      RR EE      QQ      QQ
RR      RR EE      QQ      QQ
RR      RR EE      QQ      QQ
RRRRRRRR EEEEEEEEE QQ      QQ
RRRRRRRR EEEEEEEEE QQ      QQ
RR RR EE      QQ QQ QQ
RR RR EE      QQ QQ QQ
RR RR EE      QQ      QQ
RR RR EE      QQ      QQ
RR RR EEEEEEEEE QQQQ QQ
RR RR EEEEEEEEE QQQQ QQ

```

<BLF/MACROS>

File: STRMACROS.REQ Edit: LEB1034

* This file, STRMACROS.REQ, defines macros for the string facility
to use when manipulating strings.

```
*****
*
* 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.
*
*****
```

*+ AUTHOR: R. Will, CREATION DATE: 22-JAN-79

MODIFIED BY:

```
1-001 - Original. RW 22-JAN-79
1-002 - Made $STR$ALLOCATE manipulate the short string queues
        directly using REMQUE. JBS 13-MAR-1979
1-003 - Added the "discourse on strings", and modified
        $STR$DYN_AL_LEN to correspond to it. JBS 14-MAR-1979
1-004 - Put parens around the body of $STR$DYN_AL_LEN so it can
        be used as a formal of $STR$NEED_ALLOC. JBS 14-MAR-1979
1-005 - Fix the REMQUE instruction in $STR$ALLOCATE. JBS 15-MAR-1979
1-006 - Redo the STR$SHORT_STR structure to improve efficiency and
        handle the case of a string exactly 240 long. JBS 16-MAR-1979
1-007 - Fix some bugs in long strings. JBS 16-MAR-1979
1-008 - Put the body of $STR$NEED_ALLOC in parens so it can be used
        as an expression. JBS 19-MAR-1979
1-009 - Add macros to pick up current length and pointer for strings.
        RW 02-APR-79
1-010 - Change FILL_CHAR to STR$K_FILL_CHAR. JBS 09-APR-1979
1-011 - Make $STR$ALLOCATE handle 0-length strings. JBS 15-APR-1979
1-012 - Signal using the newly-defined STR messages. JBS 16-MAY-1979
1-013 - A null string has length 0. JBS 22-MAY-1979
1-014 - Change string linkages to start with STR$. JBS 04-JUN-1979
```

- 1-015 - Fix the allocation of space for STR\$\$SHORT STR. JBS 11-JUN-1979
- 1-016 - Change all of the literals to start with STR\$. JBS 21-JUN-1979
- 1-017 - Change the allocation macro to loop on the REMQUE until it succeeds. JBS 21-JUN-1979
- 1-018 - Change BAS\$\$SCALE to call a routine. RW 26-JUN-79
- 1-019 - Change call to STR\$MOVQ_R1. JBS 25-JUL-1979
- 1-020 - Undo edit 13. A string with length 0 may still have space allocated to it. RW 17-SEP-1979
- 1-021 - Remove the PRINT statement, for the new BLISS compiler. JBS 02-OCT-1979
- 1-022 - Add macros for class and dtype. Make macros for length and addr suitable for fetch or store. 29-Oct-79
- 1-023 - Remove \$BAS\$\$SCALE macro. 29-Oct-79
- 1-024 - Change linkage name for STR\$MOVQ_R1 to track STRLNK.REQ. JBS 31-OCT-1979
- 1-025 - Add string interlocking. JBS 01-NOV-1979
- 1-026 - String cleanup, use descriptor accessing macros. RW 8-Nov-79
- 1-027 - String speedup, no signalling inside macros (except debugging portion of interlock macros). RW 8-Jan-1980
- 1-028 - Remove string interlocks. RW 19-Feb-1980
- 1-029 - Add macro \$STR\$CHECK_STATUS to map LIB\$ statuses to STR\$ signals, and signal the fatal errors.
Add macros \$STR\$GET_LEN_ADDR to extract the length and address of the first data byte of a string from any supported class of string descriptor.
RKR 3-MAY-1981
- 1-030 - Improve \$STR\$GET_LEN_ADDR so as to not declare a named variable to hold the return status. This typically frees a register from the calling routine. SBL 28-Sep-1981
- 1-031 - Rewrite \$STR\$GET_LEN_ADDR to use STR\$ANALYZE_SDESC_R2 rather than LIB\$ANALYZE_SDESC_R3. STR\$ANALYZE_SDESC and STR\$ANALYZE_SDESC_R2 do not return status, so macro no longer returns status. RKR 19-OCT-1981.
- 1-032 - Rewrite \$STR\$GET_LEN_ADDR to use STR\$ANALYZE_SDESC_R1. RKR 18-NOV-1981.
- 1-033 - Declare LIB\$STOP external where necessary. SBL 30-Nov-1981
- 1-034 - Take away NOVALUE attribute for LIB\$STOP in the \$STR\$SIGNAL_FATAL macro so that it matches the declarations in the .B32 files.
LEB 9-May-1983

--
!<BLF/PAGE>

Discourse on strings:

A string descriptor consists of a length field, a data type field, a class field, and a pointer. These are named the DSC\$W_LENGTH, DSC\$B_DTYPE, DSC\$B_CLASS and DSC\$A_POINTER fields, respectively. The user of the string package uses these fields to indicate the number of bytes in the string, the data type of the string, its allocation class, and the address of its first byte. The STR facility provides subroutines to help the user manipulate these strings.

Two classes of strings are supported: fixed-length and dynamic. Fixed-length strings have class field equal to DSC\$K_CLASS_S. Allocation of the data area pointed to by DSC\$A_POINTER is not under the control of the descriptor. Typically, the data area is allocated on the stack and will be deallocated when the routine which allocated it returns to its caller.

Dynamic strings have class field equal to DSC\$K_CLASS_D. In these strings the STR facility allocates and deallocates the data area pointed to by DSC\$A_POINTER. Thus these strings can be allocated by a routine and returned to the routine's caller.

From the user's point of view there is only one kind of dynamic string, as described above, but the STR facility distinguishes two kinds, for efficiency. Any string whose length is less than 240 bytes is considered a "short string". The STR facility allocates short strings in multiples of 8 bytes, and a request for a string of less or equal to 240 bytes is satisfied by a previously allocated string long enough to handle it. If the list of such strings is empty, more are obtained from virtual storage. A short string can be decreased in length (by the STR facility) by simply shortening the DSC\$W_LENGTH field, since the string will still be classified as a short string. When a short string is allocated, 4 extra bytes are allocated, two of which hold a count of the number of bytes which the user can access (always a multiple of 8, and always at least as large as he requested). The other two bytes are unused. They are present so that the user's string will start on a longword boundary, for efficiency.

When the user has no more use for a short string, the STR facility does not deallocate it but instead puts it on a list of strings of its length, so that a later request can re-use the string space.

A request to allocate a string longer than 240 bytes is satisfied by simply allocating it from free storage. Such a "long string" has no header bytes, so it cannot be shortened, since its allocated length is kept only in the DSC\$W_LENGTH field. When the user is done with a "long string" the STR facility returns its data to the virtual memory pool so it can be used again later, either for another string or for some other purpose, such as for holding information about an opened file.

The routines in the STR facility are AST-reentrant, so the user may allocate and deallocate strings both at non-AST level and at AST

level. However, the process of copying data from one string to another may be interrupted, and when the interrupt resumes the MOVC class of instructions do not re-examine the descriptor to see if the string has moved, so it is the user's responsibility not to re-allocate a string at AST level which might be in use at non-AST level. Within the STR facility strings are interlocked against destructive use at AST level, so the user will get error messages in many cases of this kind of string misuse, though not all.

String interlocking is implemented using a queue of descriptors. Before reallocating a string (or, in general, doing any writing into a string) the STRS routines check the queue to be sure that the string is not on it. Also, any string passed as a parameter to a STRS routine is placed on the queue while the STRS routine is using it. This allows the STRS routines to be confident that the length and pointer fields of a string descriptor passed to them will not change because of an AST reallocating the string. In order for the user to get this same level of protection, any string which could be reallocated at AST level may be accessed at non-AST level only by first making a copy of the string, by calling STRSCOPY_DX. If an AST tries to reallocate the string while it is being copied, an error will be signalled.

<BLF/PAGE>

ST

!+

!-

LI

!+

!-

FI

!+

!-

!+

!-

!+

!-

!+

!-

LI

!+

!-

!+

!-

LI

!+

!-

!+

!-

LI

!+

!-

LI

MACPOS:

The following are macros and literals used by the STRS
RTL modules. The macro names are of the form
 \$STR\$verb for macros that perform an action
 \$STR\$noun for macros that return a value
 \$STR\$condition for macros that check a condition and return T or F

MACRO

This macro is used to declare local storage for a descriptor

```
$STR$DESCRIPTOR =  
  BLOCK [8, BYTE] %,
```

This macro returns the length (current for VARYING) of a string of
any class or dtype. \This macro will have to add a case statement
if any string classes are added whose current length is not
in the same position as for fixed length strings\

```
$STR$LENGTH (DESCRIPTOR) =  
  DESCRIPTOR [DSC$W_LENGTH] %,
```

This macro returns the data type of a string.

```
$STR$DTYPE (DESCRIPTOR) =  
  DESCRIPTOR [DSC$B_DTYPE] %,
```

This macro returns the class of a string

```
$STR$CLASS (DESCRIPTOR) =  
  DESCRIPTOR [DSC$B_CLASS] %,
```

This macro returns the pointer to the string data. \This macro will have
to add a case statement if any string classes are supported whose
pointer in the descriptor does not point to the data\

```
$STR$POINTER (DESCRIPTOR) =  
  DESCRIPTOR [DSC$A_POINTER] %,
```

This macro exchanges the length and pointer fields of 2 descriptors.
It is designed to take the length and pointer from a local temp descriptor
(descriptor_1) and exchange them with the length and pointer of a
destination parameter descriptor. Even though at the point of execution
of this macro the interlock mechanism is in use to prevent anyone else
from modifying the destination descriptor, it would also provide ASI
reentrancy for a reader of the destination string if a way could be found
to move the length and pointer in one instruction.

```
$STR$EXCH_DESCS (DESCRIPTOR_1, DESCRIPTOR_2) =  
  BEGIN  
  
  LOCAL
```

```

$STR$TEMP_DESC : $STR$DESCRIPTOR;
$STR$LENGTH ($STR$TEMP_DESC) = .$STR$LENGTH (DESCRIPTOR_2);
$STR$POINTER ($STR$TEMP_DESC) = .$STR$POINTER (DESCRIPTOR_2);
XIF %BLISS (BLISS32)
XTHEN
  BEGIN
    EXTERNAL ROUTINE
      STR$$MOVQ_R1 : STR$$JSB_MOVQ;

    $STR$type (DESCRIPTOR_1) = .$STR$type (DESCRIPTOR_2);
    $STR$class (DESCRIPTOR_1) = .$STR$class (DESCRIPTOR_2);
    STR$$MOVQ_R1 (DESCRIPTOR_1 [0, 0, 0, 0], DESCRIPTOR_2 [0, 0, 0, 0]);
  END;
XELSE
  BEGIN
    $STR$LENGTH (DESCRIPTOR_2) = .$STR$LENGTH (DESCRIPTOR_1);
    $STR$POINTER (DESCRIPTOR_2) = .$STR$POINTER (DESCRIPTOR_1);
  END;
XFI

$STR$LENGTH (DESCRIPTOR_1) = .$STR$LENGTH ($STR$TEMP_DESC);
$STR$POINTER (DESCRIPTOR_1) = .$STR$POINTER ($STR$TEMP_DESC);
END;

```

%,

+ This macro is used to check for the occurrence of overlap of 2 operands. If overlapping must be checked for more than 2 operands, this macro must be used to check all possible combinations of 2 overlapping. It is used on VAX only if there is more than one source string and a destination string. It is used on other machines for any routine that has source and destination strings.

- This macro returns a TRUE if the 2 operands do overlap and a FALSE if they do not.

```

$STR$OVERLAP (POINTER_1, LENGTH_1, POINTER_2, LENGTH_2) =
  BEGIN
    IF POINTER_1 LSSA POINTER_2
    THEN
      (POINTER_2 LSS (POINTER_1 + LENGTH_1))
    ELSE
      (POINTER_1 LSS (POINTER_2 + LENGTH_2))
  END

```

%,

+ This macro checks to see if allocation is needed because enough space is not currently allocated to the destination. \It should be changed to cause allocation for overlapping operands on non VAX systems. This means the input needs to be descriptors instead of lengths.\

- This macro returns a TRUE if allocation is needed and a FALSE if not.


```

!-
$STR$NEED_ALLOC (SIZE_NEEDED, ALLOCATED_LEN) =
  BEGIN
    IF (ALLOCATED_LEN GEQU STR$K_MAXSIZESTR)
    THEN
      (SIZE_NEEDED NEQU ALLOCATED_LEN)
    ELSE
      (SIZE_NEEDED GTRU ALLOCATED_LEN)

  END
%.
!+
This macro returns the allocated length of a dynamic string.
!-
$STR$DYN_ALLOC_LEN (DESCRIPTOR) =
  BEGIN
    IF (.$STR$POINTER (DESCRIPTOR) EQLU 0)
    THEN
      0
    ELSE
      IF (.$STR$LENGTH (DESCRIPTOR) GTRU STR$K_MAXSIZESTR)
      THEN
        .$STR$LENGTH (DESCRIPTOR)
      ELSE
        BEGIN
          LOCAL
            STRING_BLOCK : REF BLOCK [, BYTE] FIELD (STR$SHORT_FIELD);

          STRING_BLOCK = .$STR$POINTER (DESCRIPTOR);
          .STRING_BLOCK [STR$W_ALLOC_LEN]
        END

      END
    END
  END
%.
!+
This macro allocates the requested amount of space to a string
and fills the length field of the descriptor with the requested length
the value of the macro is either STR$NORMAL or STR$INSVIRMEM
!-
$STR$ALLOCATE (LENGTH, DESCRIPTOR_ADDR) =
  BEGIN
    LOCAL
      RETURN_STATUS;                ! value of the macro

    EXTERNAL ROUTINE
      STR$$INIT : NOVALUE;          ! Initializes the queues

    EXTERNAL
      STR$$V_INIT;                 ! Initialization flag
  END
!+

```

! If the short string queues need initialization, initialize them.

```
IF ( NOT .STR$$V_INIT) THEN STR$$INIT ();
```

+ Initialize the value of the macro

```
RETURN_STATUS = STR$_NORMAL;
```

+ If the requested length of the string is short enough, get space from the short queues. Otherwise call LIB\$GET_VM for space.

```
IF (LENGTH LEQU STR$_MAXSIZSTR)
THEN
  BEGIN
```

+ The requested size is short, access the short queues.

```
BUILTIN
  REMQUE;
```

```
LOCAL
  REMQUE_ADDR,
  TEMP;
```

```
EXTERNAL ROUTINE
  STR$$ALLOC_SHORT;          ! Get more space for short queues
```

```
EXTERNAL
  STR$$Q_SHORT_0 : STR$$SHORT_STR [STR$_NUM_SH_QS];    ! The short queues
```

+ If the requested size is zero, just produce a descriptor for the null string.

```
IF (LENGTH EQLU 0)
THEN
```

```
  TEMP = 0
```

```
ELSE
```

```
  BEGIN
```

```
    LOCAL
```

```
      ALLOC_DONE;
```

```
      REMQUE_ADDR = STR$$Q_SHORT_0 [LENGTH, 0];
```

```
    DO
```

+ There is no more space in the requested queue, and we haven't previously failed while trying to allocate space. Allocate more space for it.

```
      IF (NOT (REMQUE (..REMQUE_ADDR, TEMP)))
```

```

        THEN
            ALLOC_DONE = 1
        ELSE
            BEGIN
                ALLOC_DONE = 0;
                RETURN_STATUS = STR$$ALLOC_SHORT (LENGTH);
            END
        UNTIL (.ALLOC_DONE OR (NOT .RETURN_STATUS));
    END;

```

↑ Store in the descriptor a pointer to the data area of the allocated string.

```

        IF .RETURN_STATUS
        THEN
            BEGIN
                $$STR$POINTER (DESCRIPTOR_ADDR) = .TEMP;
                $$STR$LENGTH (DESCRIPTOR_ADDR) = LENGTH;
            END
        ELSE
            BEGIN

```

↑ The requested length is too long for the short queues. Allocate space from virtual storage to hold the string.

```

        EXTERNAL ROUTINE
            LIB$GET_VM;                ! Get virtual storage

        EXTERNAL LITERAL
            STR$_INSVIRMEM;            ! Error code

        RETURN_STATUS = LIB$GET_VM (%REF (LENGTH), $$STR$POINTER (DESCRIPTOR_ADDR));

        IF ( NOT .RETURN_STATUS)
        THEN RETURN_STATUS = STR$_INSVIRMEM
        ELSE $$STR$LENGTH (DESCRIPTOR_ADDR) = LENGTH;

        END;

```

↑ Return the status

```

        .RETURN_STATUS
    END

```

%,

↑ This macro allocates temporary string space for fixed length string computation. Optimization should be done. Perhaps if the requested length is less than some N then the space could be allocated on the stack, otherwise in heap storage. Currently always gives heap storage\

```

    $$STR$ALLOC_TMP (LENGTH, DESCRIPTOR_ADDR) =
    $$STR$ALLOCATE (LENGTH, DESCRIPTOR_ADDR) %,

```

!+ This macro deallocates the space allocated to the string parameter
-

```

$STR$DEALLOCATE (DESCRIPTOR_ADDR) =
  BEGIN
  LOCAL
  RETURN_STATUS;          ! status of deallocate

  RETURN STATUS = STR$ NORMAL;
  IF (.$STR$POINTER (DESCRIPTOR_ADDR) NEQU 0)
  THEN
  BEGIN

```

!+ If the string is small, put on the appropriate short queue. Otherwise
- call LIB\$FREE_VM to return it to free storage.

```

  IF (.$STR$LENGTH (DESCRIPTOR_ADDR) LEQ STR$K_MAXSIZESTR)
  THEN
  BEGIN
    EXTERNAL
      STR$$SHORT_Q : STR$$SHORT_STR [STR$K_NUM_SH_QS];

    BUILTIN
      INSQUE;

    LOCAL
      INSQUE_ADDR,
      ALLOC_LENGTH,
      STRING_BLOCK : REF BLOCK [, BYTE] FIELD (STR$$SHORT_FIELD);

    STRING_BLOCK = .$STR$POINTER (DESCRIPTOR_ADDR);
    ALLOC_LENGTH = .STRING_BLOCK [STR$W_ALLOC_LEN];
    INSQUE_ADDR = STR$$SHORT_Q [ALLOC_LENGTH, 0];
    INSQUE (.$STR$POINTER (DESCRIPTOR_ADDR), ..INSQUE_ADDR);
  END
  ELSE
  BEGIN
    EXTERNAL ROUTINE
      LIB$FREE_VM;

    EXTERNAL LITERAL
      STR$_FATINTERR;

    RETURN STATUS = LIB$FREE_VM (
      %REF (.$STR$LENGTH (DESCRIPTOR_ADDR)),
      $STR$POINTER (DESCRIPTOR_ADDR));

    IF ( NOT .RETURN_STATUS) THEN RETURN_STATUS = STR$_FATINTERR;

  END;
END;

```

.RETURN_STATUS

END

%,

+ This macro is the counterpart of \$STR\$ALLOC_TMP. It releases the temporary string space used in fixed length string computation. If the space is on the stack (ie. the length in the descriptor is less than N) no action need be taken. However if the temporary space is in heap storage, (length greater than N) the storage must be released. \currently always uses heap\

```
$STR$DEALLOC_TMP (DESCRIPTOR_ADDR) =
  $STR$DEALLOCATE (DESCRIPTOR_ADDR) %,
```

+ This macro is used to take a return status from STR\$\$COPY_R_R8 or other status filled by STR routines and signal the fatal errors and just continue on success, qualified success or warning. It is to be used only after all interlocks are removed.

```
$STR$SIGNAL_FATAL (STATUS) =
  IF (NOT STATUS) THEN
  IF (.BLOCK [STATUS, ST$$V_SEVERITY] EQLU ST$$K_SEVERE)
  THEN BEGIN
    EXTERNAL ROUTINE LIB$STOP;
    LIB$STOP (.STATUS);
  END; %,
```

+ \$STR\$CHECK_STATUS inspects the current status. If it finds it to be one of the fatal LIB\$ status codes, it signals the corresponding STR\$ code. Some LIB\$ statuses are converted to the corresponding STR\$ statuses. Unknown status are left alone.

```
$STR$CHECK_STATUS (RETURN_STATUS) =
  BEGIN
  EXTERNAL ROUTINE
    STR$$CHECK_STATUS_R2 : STR$$CHECK_STATUS_LINKAGE ;

  RETURN_STATUS = STR$$CHECK_STATUS_R2 ( .RETURN_STATUS ) ;
  END
  %,
```

+ This macro is the mechanism by which the string routines derive the length and address of the 1st data byte of any supported class of string descriptor. It is isolated here as a macro to that it can be modified (e.g. enhance to speed up the extraction of some classes of descriptors at the expense of others) by simply recompiling all of the string routines and not having to modify their sources.

```
$STR$GET_LEN_ADDR ( DESCRIPTOR_ADDR, LENGTH, DATA_ADDR ) =
```

```
BEGIN          ! of macro
EXTERNAL ROUTINE
  STR$ANALYZE_SDESC_R1 : STR$ANALYZE_SDESC_JSB_LINK NOVALUC ;

!+
!- special-case the classes of descriptors in Version 2
IF .DESCRIPTOR_ADDR [DSC$B_CLASS] LEQU DSC$K_CLASS_D
THEN
  BEGIN
    LENGTH = .DESCRIPTOR_ADDR [ DSC$W_LENGTH ] ;
    DATA_ADDR = .DESCRIPTOR_ADDR [DSC$A_POINTER] ;
  END
ELSE
  !+
  !- JSB to STR$ANALYZE_SDESC_R1, computing LENGTH and DATA_ADDR.
  BEGIN
    STR$ANALYZE_SDESC_R1 ( .DESCRIPTOR_ADDR ;
                          LENGTH, DATA_ADDR);
  END
END          ! of macro
```

```
!<BLF/PAGE>
```

All interlocking macros are disabled pending further study.

Problem with the commented out code include

1. if AST interrupts while interlocks are set and AST does unwind, there is no handler to remove interlocks, so interlock queue points into garbage.
2. If AST interrupts in BASIC RUN command while interlocks are set and AST does unwind of a BASIC frame which owns one of the interlocked strings, it will try to free the string, which will signal interlocked which will be caught by BASIC RUN handler which will try to unwind, hence infinite loop.

The following macros are used to support string interlocking.

\$STR\$INTERLOCK does the declarations,
 \$STR\$INTERLOCK_WRITE interlocks a string that is to be written,
 \$STR\$INTERLOCK_READ interlocks a string that is to be read, and
 \$STR\$INTERLOCK_CLEAR releases the interlock on a string.

\$STR\$INTERLOCK takes one argument: the maximum number of strings which can be interlocked.

The other three macros take two arguments: the address of the descriptor to be operated on and its number. Numbers can range from 0 to the maximum number of strings minus 1.

All strings to be written should be interlocked first, followed by all strings to be read. Strings must be cleared in the reverse of the order in which they were interlocked.

There is extra consistency checking for debugging purposes which can be turned on and off by using the following DEBUG switch

```
COMPILETIME STR$$K_DEBUG = 0;           ! 1->do extra checking
                                         ! 0->no checking
```

```
MACRO
  $STR$INTERLOCK (NUM_STRINGS) =
  LOCAL
  %IF STR$$K_DEBUG %THEN
    $STR$QUEUED_COUNT,
  %FI
    $STR$INTERLOCK_CONTROL_BLOCKS : VECTOR [NUM_STRINGS*4, LONG];
  EXTERNAL ROUTINE
    STR$$UNWDEQ;           ! Dequeues strings on unwind
  EXTERNAL
    STR$$Q_INTLK : VECTOR [2, LONG];
  LITERAL
    $STR$MAX_STRINGS = (NUM_STRINGS);
  %IF STR$$K_DEBUG %THEN
    $STR$QUEUED_COUNT = 0;
```

```

%FI
    ENABLE
        STR$$UNWDEQ ($STR$QUEUED_COUNT);
    S$$_NORMAL
    %,
    $STR$INTERLOCK_WRITE (DESCRIPTOR_ADDR, STRING_NUMBER) =
        BEGIN
            BUILTIN
                INSQUE;

            LOCAL
                INSQUE_ADDR,
                INTERLOCK_ADDR : REF VECTOR [3, LONG];

%IF STR$$K_DEBUG
%THEN
    IF (((STRING_NUMBER) LSS 0) OR ((STRING_NUMBER) GEQ $STR$MAX_STRINGS))
    THEN
        BEGIN
            EXTERNAL LITERAL
                STR$_FATINTERR;

            LIB$STOP (STR$_FATINTERR);
        END;
%FI

    INTERLOCK_ADDR = (($STR$INTERLOCK_CONTROL_BLOCKS [(STRING_NUMBER)*4] + 7) AND (NOT 7));
    INTERLOCK_ADDR [2] = DESCRIPTOR_ADDR;

    IF (.STR$$Q_INTLK [0] EQLA 0)
    THEN
        BEGIN
            STR$$Q_INTLK [1] = STR$$Q_INTLK [0];
            STR$$Q_INTLK [0] = STR$$Q_INTLK [0];
        END;

    INSQUE_ADDR = STR$$Q_INTLK [1];

    IF (NOT (INSQUE (INTERLOCK_ADDR [0], ..INSQUE_ADDR)))
    THEN
        BEGIN
            LOCAL
                SRCH_STATUS;

            EXTERNAL ROUTINE
                STR$$SRCH_INTLK;

%IF STR$$K_DEBUG %THEN
    $STR$QUEUED_COUNT = .STR$QUEUED_COUNT + 1;
%FI
%FI

```



```

!!      SRCH_STATUS =
!!      STR$$SRCH_INTLK (DESCRIPTOR_ADDR)
!!      :
!!      IF ( NOT .SRCH_STATUS) THEN LIB$STOP (.SRCH_STATUS);
      END
    ELSE
      BEGIN
%IF STR$$K_DEBUG %THEN
      $$STR$QUEUED_COUNT = .$$STR$QUEUED_COUNT + 1;
%FI
      STR$_NORMAL
      END
    END
  $$$_NORMAL
  %
  $$STR$INTERLOCK_READ (DESCRIPTOR_ADDR, STRING_NUMBER) =
  BEGIN
    BUILTIN
      INSQUE;

    LOCAL
      INSQUE_ADDR,
      INTERLOCK_ADDR : REF VECTOR [3, LONG];

%IF STR$$K_DEBUG
%THEN
      IF (((STRING_NUMBER) LSS 0) OR ((STRING_NUMBER) GEQ $$STR$MAX_STRINGS))
      THEN
        BEGIN
          EXTERNAL LITERAL
            STR$_FATINTERR;

          LIB$STOP (STR$_FATINTERR);
        END;
%FI

    INTERLOCK_ADDR =
      (($$STR$INTERLOCK_CONTROL_BLOCKS [(STRING_NUMBER)*4] + 7)
      AND ( NOT 7));
    INTERLOCK_ADDR [2] = DESCRIPTOR_ADDR;

    IF (.STR$$Q_INTLK [0] EQLA 0)
    THEN
      BEGIN
        STR$$Q_INTLK [1] = STR$$Q_INTLK [0];
        STR$$Q_INTLK [0] = STR$$Q_INTLK [0];
      END;

    INSQUE_ADDR = STR$$Q_INTLK [1];
    INSQUE (INTERLOCK_ADDR [0], ..INSQUE_ADDR);
%IF STR$$K_DEBUG %THEN

```

```

%FI      $STR$QUEUED_COUNT = . $STR$QUEUED_COUNT + 1;
%FI
      END
      $$$_NORMAL
%$
      $STR$INTERLOCK_CLEAR (DESCRIPTOR_ADDR, STRING_NUMBER) =
      BEGIN
      BUILTIN
      REMQUE;

      LOCAL
      REMQUE_ADDR,
      INTERLOCK_ADDR : REF VECTOR [3, LONG],
      TEMP : REF VECTOR [3, LONG];

%IF STR$$K_DEBUG
%THEN
      EXTERNAL LITERAL
      STR$ FATINTERR;
      IF (((STRING_NUMBER) LSS 0) OR ((STRING_NUMBER) GEQ $STR$MAX_STRINGS))
      THEN
      LIB$STOP (STR$_FATINTERR);
%FI

      IF (.STR$$Q_INTLK [0] EQA 0)
      THEN
      BEGIN
      STR$$Q_INTLK [1] = STR$$Q_INTLK [0];
      STR$$Q_INTLK [0] = STR$$Q_INTLK [0];
      END;

      REMQUE_ADDR = STR$$Q_INTLK [1];
      REMQUE (.REMQUE_ADDR, TEMP);
%IF STR$$K_DEBUG %THEN
      $STR$QUEUED_COUNT = . $STR$QUEUED_COUNT - 1;

      IF (. $STR$QUEUED_COUNT LSS 0) THEN LIB$STOP (STR$_FATINTERR);
%FI

      INTERLOCK_ADDR =
      (( $STR$INTERLOCK_CONTROL_BLOCKS [(STRING_NUMBER)*4] + 7)
      AND ( NOT 7));

%IF STR$$K_DEBUG %THEN
      IF (.TEMP NEQA .INTERLOCK_ADDR) THEN LIB$STOP (STR$_FATINTERR);
      IF (.TEMP [2] NEQA DESCRIPTOR_ADDR) THEN LIB$STOP (STR$_FATINTERR);
%FI

      END
      $$$_NORMAL
%$
!<BLF/PAGE>

```

!+ Define a default fill character to be used as pad in fixed length strings
!-

LITERAL
STR\$K_FILL_CHAR = %C' ';

!+ Define Dynamic String Control Block (pointed to by descriptor).
!- Actually the descriptor (DSC\$A_POINTER) points to where the string is
!- stored in the control block and the header is before that (negative offset).
!-

FIELD
STR\$SHORT_FIELD =
SET

!+ Offset to word containing allocated length. Actually # of bytes following
!- which is also max. size of string that can be held in the control block.
!- In other words this count does not include the space taken up by itself.
!-

STR\$W_ALLOC_LEN = [-2, 0, 16, 0]
TES;

!+ Number of bytes in the control block header.
!-

LITERAL
STR\$K_HED_LEN = 2;

!+ no. of extra bytes that can be in string because next header doesn't
!- take up entire longword. (ie. residue is no. extra bytes
!- beyond n*STR\$K_ALL_QUA in string area).
!-

LITERAL
STR\$K_RESIDUE = %UPVAL - STR\$K_HED_LEN;

!+ Power of 2 of allocation unit (8)
!- MAINTENANCE NOTE: Some advantage is taken in the code that the
!- size of the Q header is the same as the quanta. Thus no shifting
!- is needed in order to perform INSQUE and REMQUE, only adding offsets.
!-

LITERAL
STR\$K_ALL_POW = 3;

!+ Number of bytes in allocation quanta
!-

LITERAL
STR\$K_ALL_QUA = 1^STR\$K_ALL_POW;

```

|+
| Number of short queues, one queue for each allocation quanta.
| With current parameter settings, string area
| sizes are 8, 16, 24, ..., 240.
|-

```

```

LITERAL
STR$K_NUM_SH_QS = 30;

```

```

|+
| Max. size of string which can fit in short Qs.
|-

```

```

LITERAL
STR$K_MAXSIZESTR = ((STR$K_NUM_SH_QS)*STR$K_ALL_QUA);

```

```

|+
| Sizes of descriptors for all string types
|-

```

```

LITERAL
STR$K_SIZE_FIX = 2, ! descriptor size for fixed length strings
STR$K_SIZE_VARY = 3, ! descriptor size for varying strings
STR$K_SIZE_DYN = 2; ! descriptor size for dynamic strings

```

```

|+
| The following structure is used to access (and define) the queue
| of short strings. It is vector of INSQUE/REMQUE quadwords. Each
| quadword represents a queue of potential strings which can be
| allocated to a requesting program.
|-

```

```

STRUCTURE
STR$$SHORT_STR [I, SIDE; N] =
  [N*%UPVAL*2]
  (STR$$SHORT_STR + (SIDE + ((I - 1) AND ( NOT (STR$K_ALL_QUA - 1)))));
!
  End of file STRMACROS.REQ

```


RTLMACB32 REQ	STRMACROS REQ	RTLOOBBG REQ	RTLPSECT REQ	STRLNK REQ	RTLMACMAR MAR	LIBASCEBC LIS	LIBASTINP LIS	LIBBINTRE LIS	LIBCHAR LIS
					LIBDEF FOR	LIBANASTR LIS	LIBASNMBX LIS	LIBBBCCI LIS	
					LIBABUPCA LIS	LIBADDP LIS			
					SIGDEF FOR	LIBASCTIM LIS			
					LIBTABMAC MAR	LIBATTACH LIS			
					LIBAZEREV LIS	LIBBBSST LIS	LIBCALLG LIS	LIBCLICAL LIS	
					LIBADDP LIS				