TABLE OF CONTENTS

SDL2 LANGUAGE SYNTAX

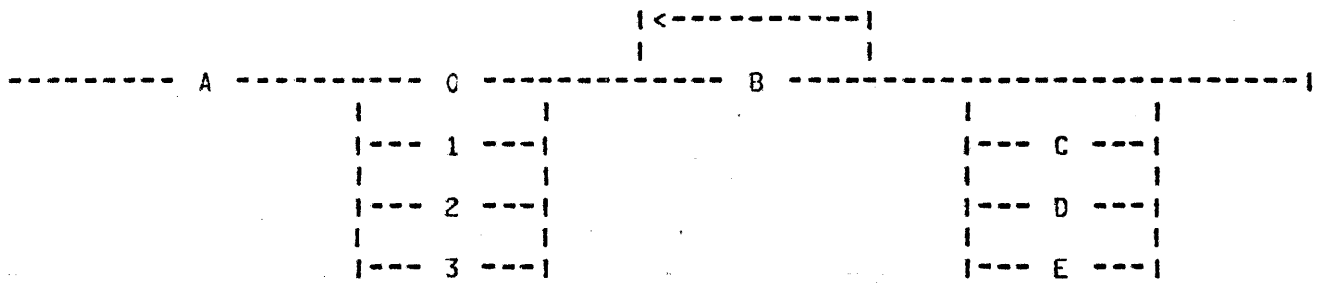
The syntax for the SDL2 language is described with syntax diagrams.

A syntax diagram consists of a main line which contains all required elements of the syntax. The main line may have other lines above and below it. Lines above the main line indicate loops back in the syntax, and lines below the main line indicate alternate productions. Continuation from one line of a diagram to another is represented by a right arrow, ">", at the end of the current line and the beginning of the next line. A complete syntax diagram is terminated by a vertical bar, "|".

If an element appears on the main line of the syntax diagram, with no lines below it, then that element is required. If an element appears on the main line of the diagram, but other elements appear in lines directly below it, ther one of those elements is required. If an element appears in a line below the main line, and the main line abcve it is empty, then the element is optional.

Elements which are in uppercase, or are special characters, are terminal symbols and must appear literally in the final production. Lowercase elements reference a syntax diagram of that name.

Example:



where

A09	is valid
A188	is valid
A2888C	is valid
A0	is invalid

STRUCTURE OF AN SDL2 PROGRAM

---------- Declarations --------- Body ----------|

The structure of an SDL2 program must follow the sequence described in the syntax graphs. All variables and data types must be declared before the procedures, and all procedures must be declared before the main executable part of the program (Body).

Program execution begins with the first statement in the program Body.

TOKENS

SDL2 recognizes the following token types: identifiers, bit strings, fixed numbers, character strings and special characters.

Comments are allowed anywhere in the input stream, except within quoted strings. There are two forms of comments. The first form begins with a % and terminates with the end of that line. The second form begins with /* and terminates with */. A comment enclosed in /* */ may span multiple lines.

Blanks, special characters, comments and end-of-lines are all treated as separators, and so cannot be embedded within a token.

IDENTIFIERS

An SDL2 identifier may contain a maximum of 72 characters. Valid characters include upper-case letters (A-Z), lower-case letters (a-z), numerals (0-9), and the underscore (_). Identifiers must begin with a letter, and are case-sensitive.

FIXED NUMBERS

A string consisting entirely of numerals, without any bracketing characters, is treated as a fixed number.

Example:

```
12366
400
```

CHARACTER STRINGS

Character strings are bracketed by the character ". If a quote is desired within a quoted string, then two quotes ("") must be used.

Example:

```
"ABCDEF"
"KEYWORD ""THEN"" IS OUT OF CONTEXT"
```

BIT STRINGS

Bit strings are bracketed by the character @. The string may be either binary, quartal, octal or hexadecimal, where the number of bits per digit is specified within parenthesis. The default is hexadecimal (4 bits per digit).

Example:

```
@0AFF@
@(1)0100@
@(2)0122@
```

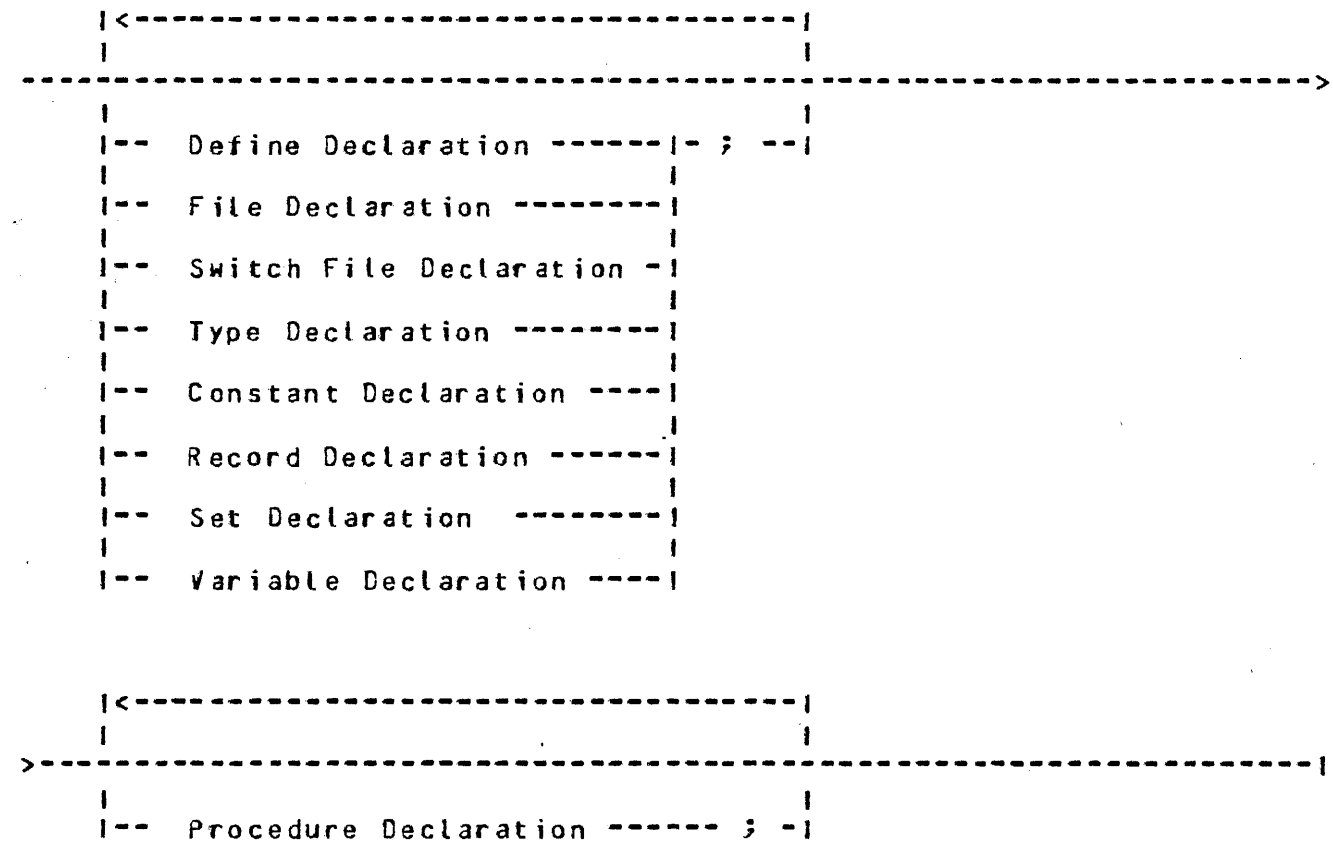
SPECIAL CHARACTERS

SDL2 recognizes the following special characters.

&	.	,	<	@	[
$	:	+	>	"	]
(	;	-	*	'	
)	!	=	/	#	

DECLARATIONS

DECLARATIONS



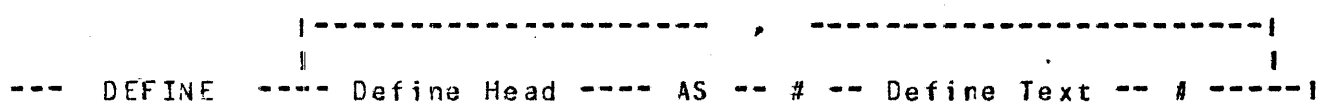
All identifiers must be declared before they can be referenced, except within the define text of a define declaration, where they must be declared before the expansion of the define.

Scope of an Identifier

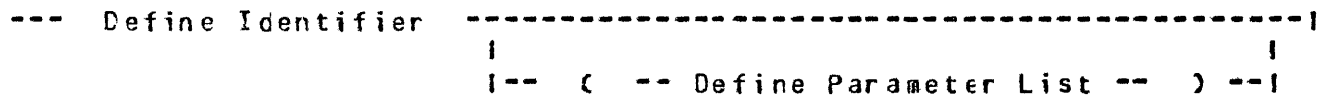
For global declarations, the scope of an identifier is all the subsequent declarations, nested procedures and the main body of the program, except for any nested procedures that declare another version of the indentifier.

For a declaration within a procedure, the scope of an identifier is that procedure and any nested procedures, except for any that declare another version of the identifier.

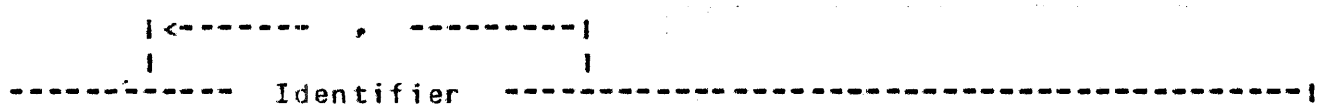
DEFINE DECLARATION



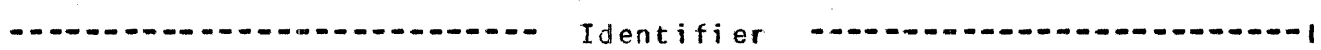
Define Head



Define Parameter List



Define Identifier



The DEFINE declaration allows a string of text to be allocated an identifier. Whenever this identifier is encountered in the source program it is replaced by the text (except in a define declaration). Define expansion is also turned off when the compiler is expecting an identifier in any of the following: Declaration identifier, Define Identifier, Procedure Identifier, Formal Parameter Identifier, Do Group name.

A define declaration may have parameters. When the define text is invoked there must be a text string for every parameter. This string will replace all occurences of the define parameter identifier in the define text.

Example:

```
DEFINE
  FALSE      AS #0#,
  TRUE       AS #1#,
  ARRAY_SIZE AS #1024#,

  MAX(X,Y) AS # IF X>Y
                THEN X
                ELSE Y#;
```

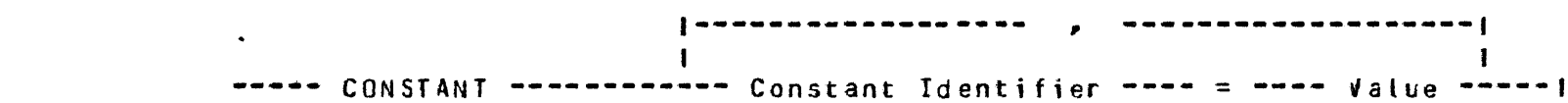
FILE DECLARATION

See section 6 - INPUT/OUTPUT

SWITCH FILE DECLARATION

See section 6 - INPUT/OUTPUT

CONSTANT DECLARATION



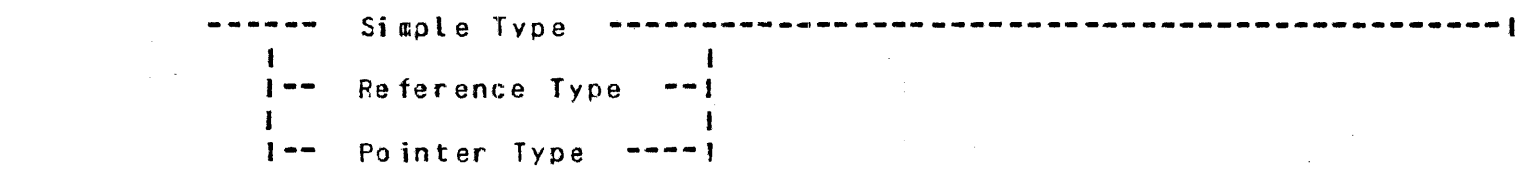
A constant declaration introduces an identifier as a synonym for a constant. Value may be either a character or bit literal, an expression which evaluates to a fixed constant, or a previously defined constant.

Example:

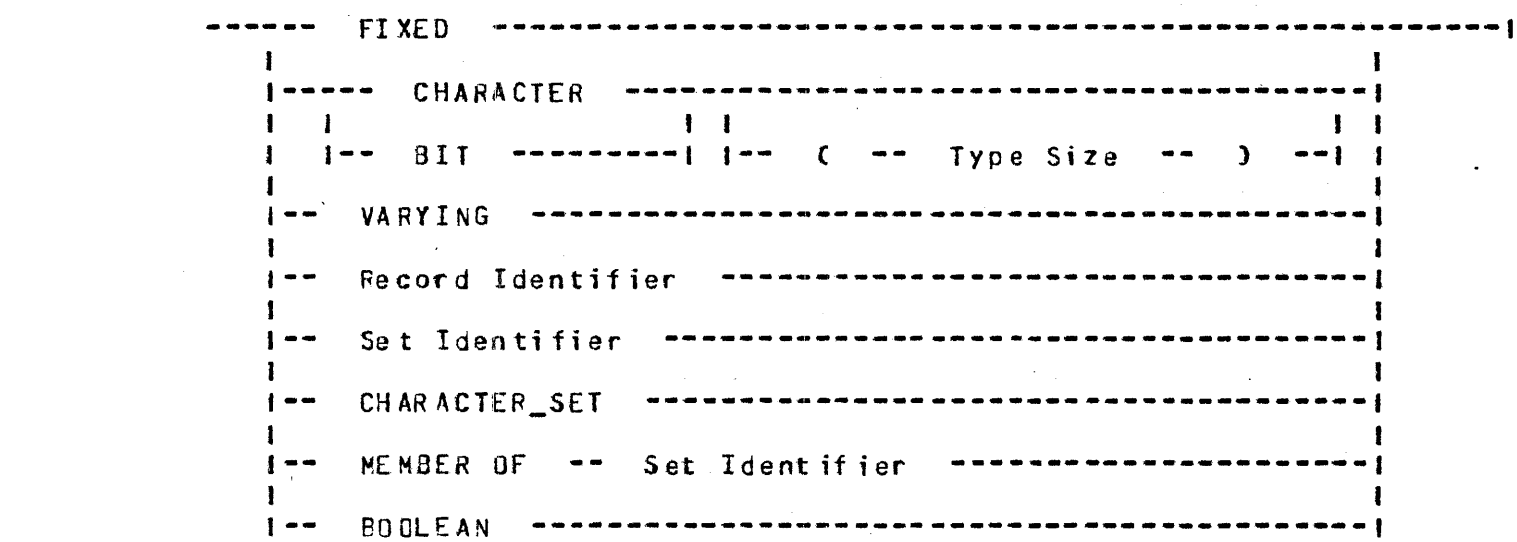
```
        CONSTANT    TEN   = 10,
                    FIFTY = 5 * TEN,
                    LEVEL = "11.0";
```

TYPES

Type



Simple Type



The type BIT will allocate a field of <Type Size> bits. It can have a maximun size of 65535 bits. <Type Size> can only be omitted in the <Type> part of a formal parameter declaration, the <Type> part of a procedure declaration, or a Reference Type declaration, indicating that the size is dynamic.

The type CHARACTER will allocate a field of <Type Size> characters. Each character occupies 8 bits. It can have a maximum size of 8191 characters. <Type Size> can only be omitted in the <Type> part of a formal parameter declaration, the <Type> part of a procedure declaration, or a Reference Type declaration, indicating that the size is dynamic.

The type FIXED will allocate a 24 bit field, where the high orcer bit is interpreted as the sign bit (0 = positive, 1 = negative). Negative values are represented in their two's complement form, and can have a range of -(2**23)+1 to (2**23)-1.

The type VARYING can only be used in a procecure oeclaration and a fcrmal parameter declaration. It indicates that both the type and length are dynamic.

The type <Record Identifier> will allocate a field with the length of the sum of the fields in the Record Declaration, and make available the field names for field selection.

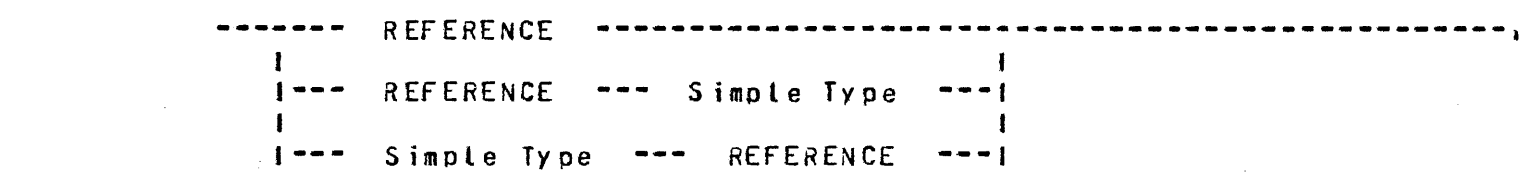
The type <Set Identifier> will allocate a field with a length equal to the number of members declared in the Set Declaration (in bits).

The type CHARACTER_SET is a predefined Set Declaration with 256 members, the EBCDIC character set.

The type <MEMBER OF> will allocate a field with a length equal to number of bits required to represent the set member with the largest ordinal value.

The type BOOLEAN is a predefined Set Member Declaration. It is defined as a member of a set containing two members, TRUE and FALSE.

Reference Type



Example:

```
DECLARE
   TEXT REFERENCE CHARACTER,
   SOURCE CHARACTER (80);
REFER TEXT TO SOURCE;
```

If a reference type is declared in a Variable Declaration, a descriptor is allocated for it on the Descriptor Stack. It originally has no data associated with it, but can be REFERed to some data space.

If a field is declared to have a type of reference in a Record Declaratation, 96 bits are allocated. If it is REFERed to a field in static memory, a copy of the descriptor is storec into the field. If it is REFERed to a field in dynamic memory, a symbolic form of the descriptor is stored in the field. The selection of a field that is a reference type will cause an automatic dereference; the descriptor can only be accessed with a REFER statement.

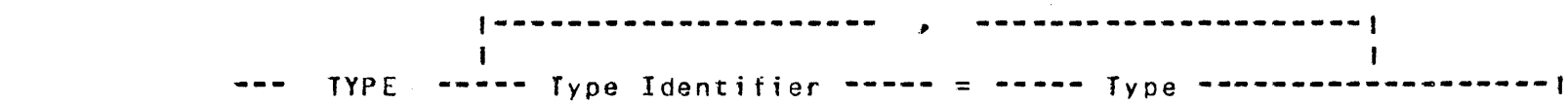
Pointer Type

----- POINTER -- Simple Type ----------

The type of a pointer must be a fixed length type. A field of 24 bits will be allocated to contain the address of the refered data space. There is no protection if a pointer is refered into dynamic memory and that page gets rolled out of memory. Its main use is for the MCP. Reference to a pointer will cause an automatic dereference to the field pointed to by the pointer.

```
RECORD RS_NUCLEUS
     .
     .
     .
     MCP_BIT      BIT(1),
RECORD HINTS_FORM
     .
     .
     .
     FIRST_Q      POINTER RS_NUCLEUS,
     .
     .
     .
     MCP_RSN      POINTER RS_NUCLEUS,
     .
     .
     .
DECLARE HINTS     HINTS_FORM;

HINTS.MCP_RSN.MCP_BIT:=TRUE;
```

TYPE DECLARATION

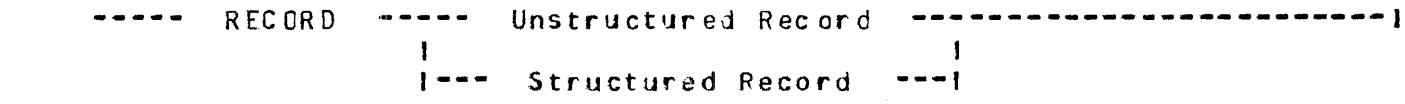


The TYPE Declaration allows an identifier to appear as a Type inside a Variable Declaration, Procedure Declaration, or Type Declaration. Only fixed length types are allowed.

Example:

```
TYPE      STATE_INDICATOR = MEMBER OF STATE_SPACE,
          ADDRESS         = BIT (24),
          STATE_VECTOR    = (8) STATE_INDICATOR;

DECLARE   STATE     STATE_INDICATOR,
          VECTOR    STATE_VECTOR,

IF VECTOR (I) IN TERMINAL_STATES THEN TERMINATE (I);
```

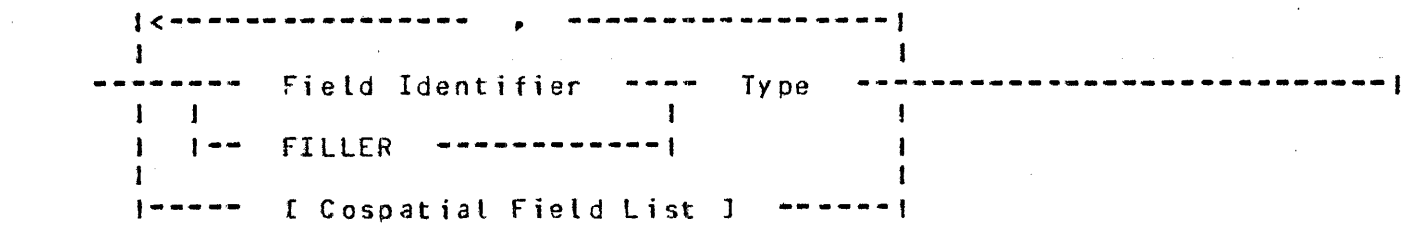
RECORD DECLARATION



The Record Declaration is a convenient way of mapping the layout of a structure. It allows variables and fields within other records to be given structure. (See Selector, and WITH statement for referencing of subfields).

Unstructured Record

Field List



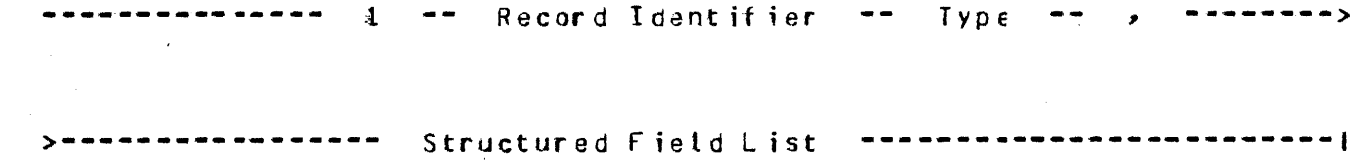
Cospatial Field List

Fields may be declared cospatial, that is each field list remaps the other cospatial field lists, and the length will always be the length of the longest field list.

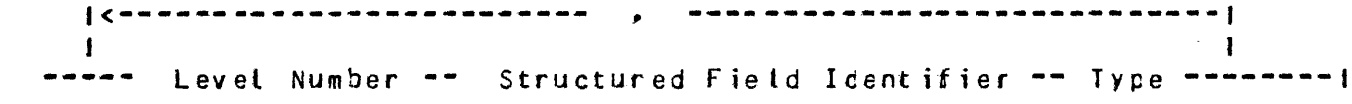
Example:

```
RECORD SYSTEM_DESCRIPTOR_FORM
  IN_USE                    BIT(1),
  MEDIA                     BIT(1),
  LOCK                      BIT(1),
  IN_PROCESS                BIT(1),
  INITIAL                   BIT(1),
  FILE_OBJECT               BIT(1),
  DK_FACTOR                 BIT(3),
  SEG_PG                    BIT(7),
  TYPE                      BIT(4),
  [ADDR                     BIT(36),
  !FILLER                   BIT(12),
  CORE                      BIT(24),
  ],
  LEN                       BIT(24);

  DECLARE
     SYSTEM_DESCRIPTOR  SYSTEM_DESCRIPTOR_FORM;

IF SYSTEM_DESCRIPTOR.MEDIA
     THEN READ_TO_MEMORY;
```

Structured Record



Structured Field List



Structured Field Identifier

Remaps Object

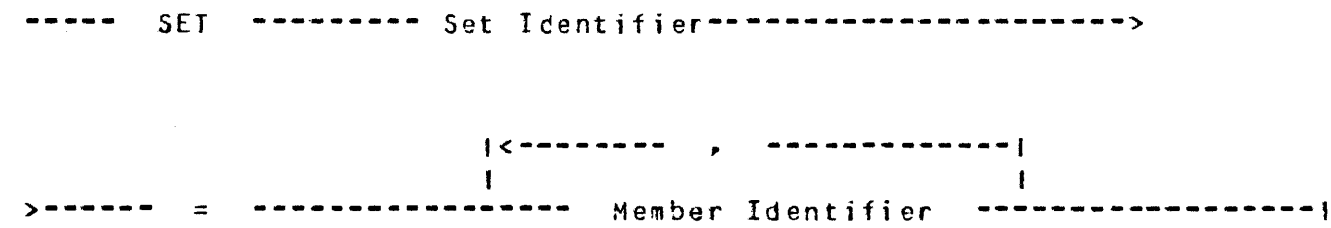
------- Identifier ----------

Example:

```
RECORD 01 MEMORY_LINK BIT(MEMORY_LINK_SIZE),
            02 ML_DISK                          DSK_ADDR,
            02 ML_GROUP                         BIT(47),
              03 ML_POINTER                     ADDRESS,
              03 ML_JOB_NUMBER                  BIT(16),
              03 ML_TYPE                        BIT(6),
              03 ML_SAVE                        BIT(1),
            02 ML_SIZE                          BIT(24),
            02 ML_PRIORITY_FIELD                BIT(30),
              03 ML_DK_INTERVAL                 BIT(10),
              03 ML_CURRENT_DK_INT              BIT(10),
              03 ML_INCOMING_PRIORITY           BIT(5),
              03 ML_RESIDENCE_PRIORITY          BIT(5),
                04 ML_RP_WHOLE                 BIT(4),
                04 ML_RP_FRACTION              BIT(1),
            02 ML_FRONT                         BIT(24),
            02 ML_BACK                          BIT(24),
            02 ML_USAGE_BITS                    BIT(2),
              03 ML_PREVIOUS_SCAN_TOUCH         BIT(1),
              03 ML_CURRENT_SCAN_TOUCH          BIT(1);
```

The structured record declaration is included in SDL2 only as a means of easier conversion of the SDL PL/I style structures.

SET DECLARATION



The Set Declaration is used to define the Members of a set. Each member is given an ordinal value, in descerding order. The last member will have a value of zero.

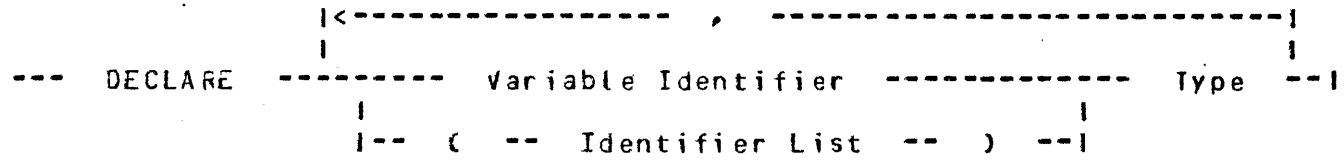
CHARACTER_SET is a pre-defined SET declaration of 256 members.

See also MEMBER OF in type declarations.

Example:

```
SET  BOOLEAN = TRUE, FALSE;
DECLARE
   STATE     MEMBER OF BOOLEAN,
   NUMERICS  CHARACTER_SET,
   CH        CHARACTER(1); % COULD BE MEMBER OF CHARACTERSET.
NUMERICS := ["0", "1", "2", "3", "4", "5", "6," "7", "8", "9"];

IF CH IN NUMERICS
  THEN STATE:= TRUE;
```

VARIABLE DECLARATION



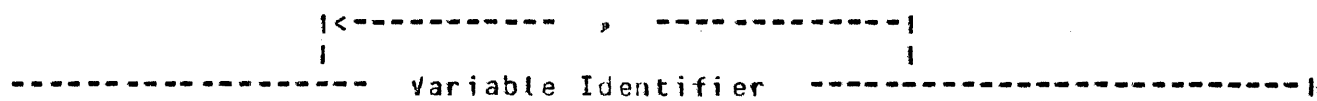
The DECLARE statement allocates memory storage for variables and defines their type.

Static variables will be allocated space in memory and will not have a descriptor.

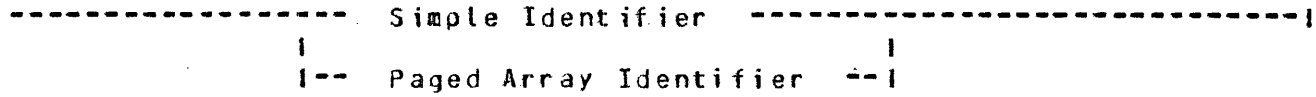
Dynamic variables will be allocated a descriptor, and space will be allocated when an ALLOCATE is executed if it is at lexic level 0, or on procedure entry if nested in a procedure. No explicit declaration is required for dynamic variables, except PAGED arrays. If the type size and/or the array bound is * at lexic level 0, or an expression (non constant) at any other lexic level, it will be considered a dynamic declaration. Lexic level 0 dynamic variables must be allocated in the global code. (See also dynamic parameters to procedures).

REFERENCE variables do not have space allocated, but get a descriptor allocated that may be REFERed to some data space.

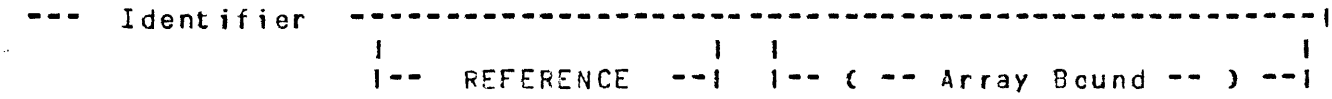
Identifier List



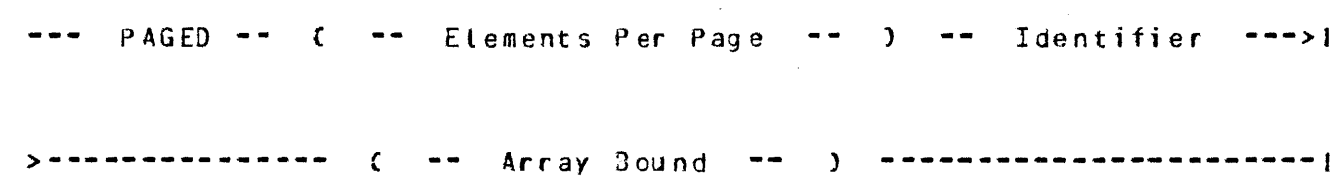
Variable Identifier



Simple Identifier



Paged Array Identifier

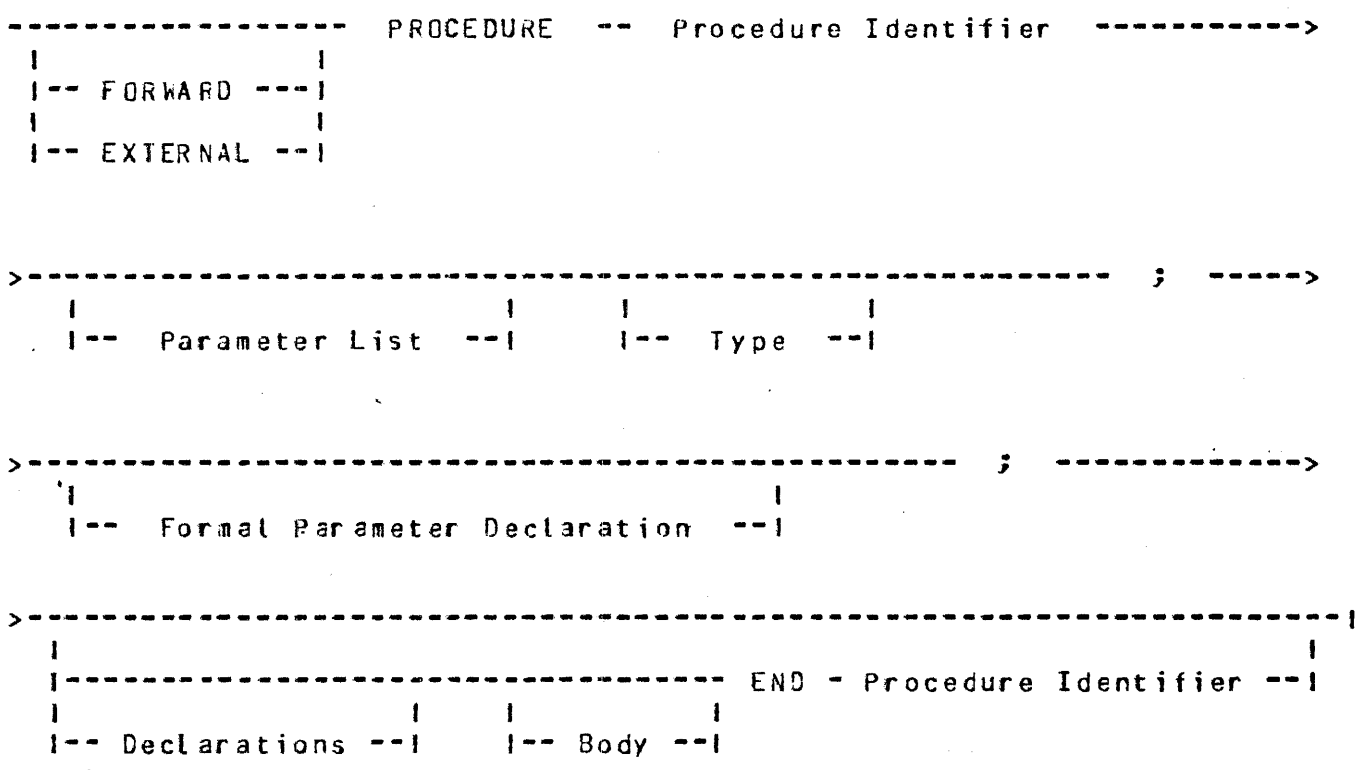


Elements Per Page must be a power of two, between 2 and 32768.
Array Bound can range from 1 to 65535.

Example:

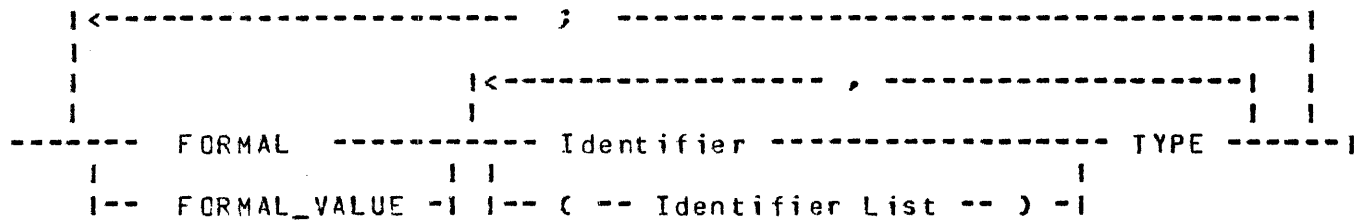
```
DECLARE
  I                    FIXED,
  (J,K)                CHARACTER(10),
  A (10)               FIXED,
  C                    CHARACTER(*),
  R REFERENCE          FIXED,
  PAGED (16) P (1024) FIXED;
```

PROCEDURE DECLARATION

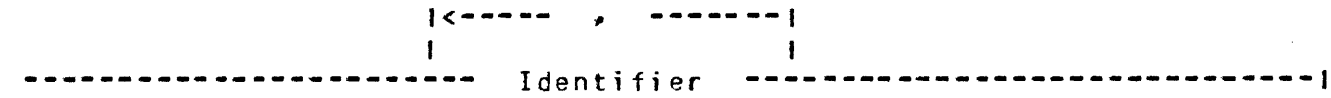


Parameter List

Formal Parameter Declaration



Identifier List



The Procedure Declaration is used to define a program fragment and to associate it with an identifier, so that it can be activated by a Call Statement or Typed Procedure Call.

Declarations within a procedure only have the scope of that procedure (i.e. when the END Procedure Identifier is encountered they are no longer accessible to the programmer). If a procedure needs to be called before it is declared, a 'FORWARD' declaration must be made to define it and its formal parameters. See 'Separate Compilation' for use of 'EXTERNAL'.

Procedures may be nested to a maximum of 15 levels.

The order of declaration of formal parameters must match the order in <Parameter List>.

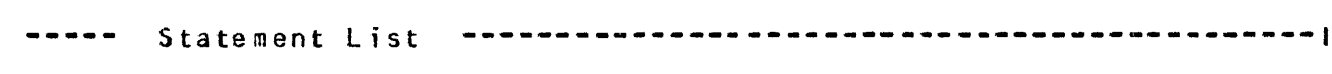
If a parameter is declared FORMAL_VALUE, a copy of the data will be made, and the original data will be preserved. An Array may not be declared FORMAL_VALUE.

Example:

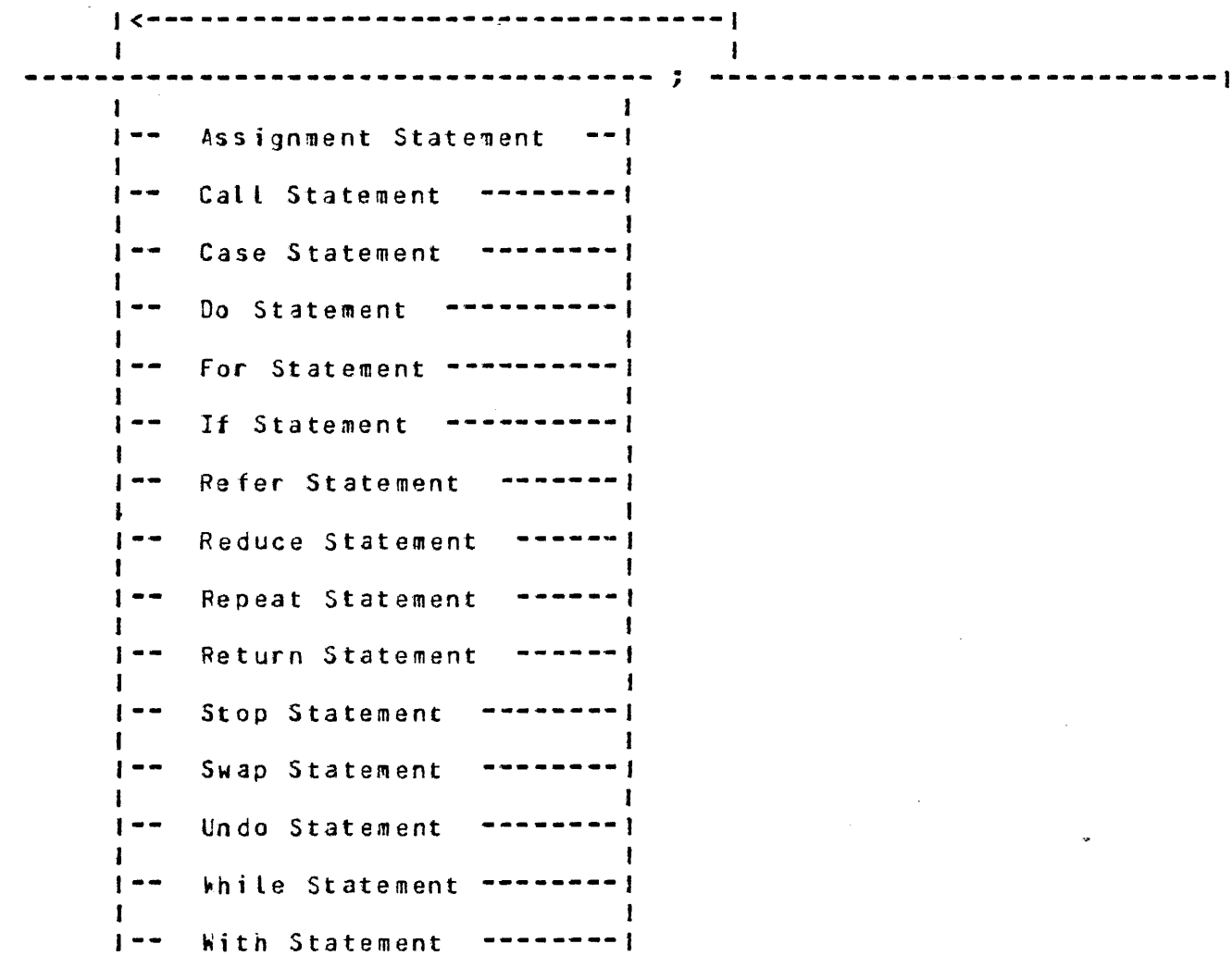
```
FORWARD PROCEDURE A(P);
  FORMAL P FIXED;

PROCEDURE B(P);
  FORMAL P FIXED;
  .
  .
  A(P);
  .
  .
END B;

PROCEDURE A(P);
  FORMAL P FIXED;
  .
  .
  B(P);
  .
  .
END A;

PROCEDURE MAX(X,Y) FIXED;
  FORMAL (X,Y) FIXED;
  IF X>Y
    THEN RETURN X;
    ELSE RETURN Y;
END MAX;
```

STATEMENTS

BODY



STATEMENT LIST



ASSIGNMENT STATEMENT

```
----- Address Generator ---- := ---- Expression ----------!
```

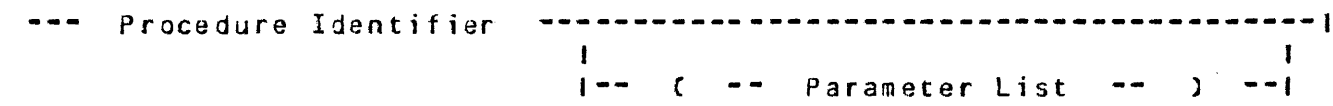
The Assignment Statement is used to store a value into a variable.

See also Put File Attribute in INPUT/OUTPUT section.

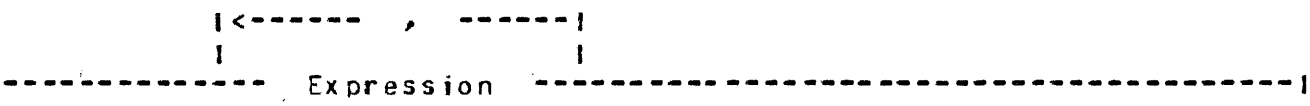
Example:

```
DECLARE
  (A, B, C)    FIXED;

A:= B + C;
```

CALL STATEMENT



Parameter List

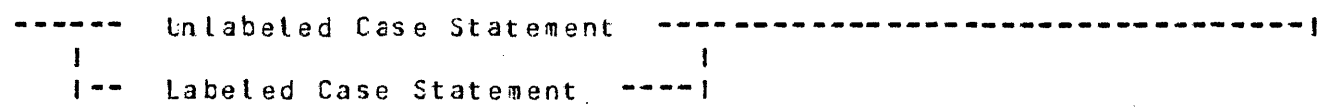


The Call Statement is used to activate a procedure that has been declared with a Procedure Declaration. There must be an Expression in the Parameter List for every Formal Parameter declared.

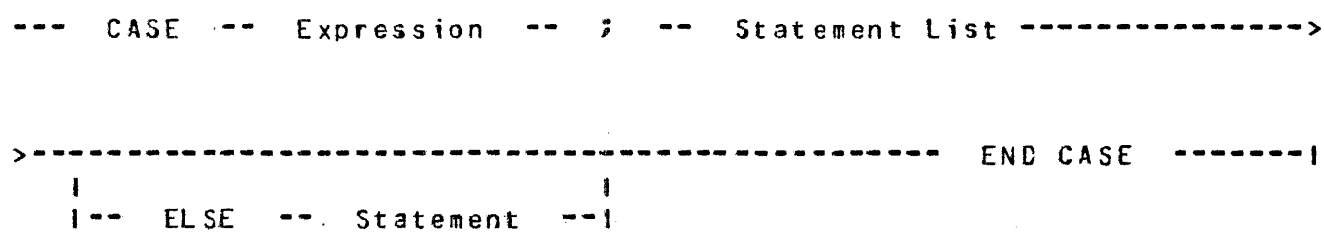
Example:

```
A:= MAX(I,J);
```

CASE STATEMENT



Unlabeled Case Statement

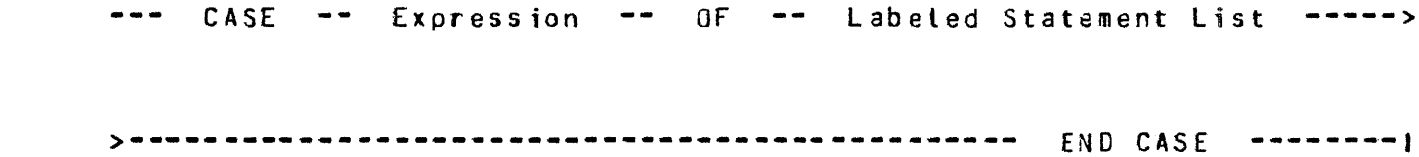


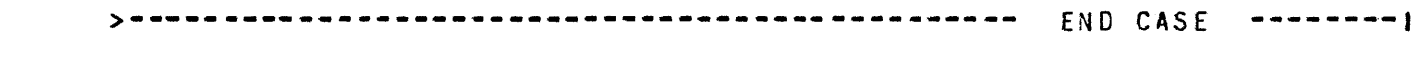
The value of Expression is used as an index into Statement List. The Statement selected will be executed and control passed to the statement following END CASE, unless an UNDO or RETURN is executed. If there are N statements in the list then the range of the expression may be 0 to N-1. If the value of Expression is out of range, the Statement following ELSE will be executed and control then passed to the Statement following END CASE (unless an UNDO or RETURN statement was executed in the ELSE Statement). If Expression is out of range and there is no ELSE Statement, a run time error will be generated.

Example:

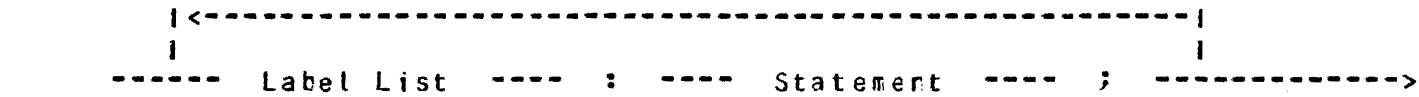
```
DECLARE
  HEX         CHARACTER(1),
  I           FIXED;
CASE I;
  HEX:= "0";
  HEX:= "1";
  HEX:= "2";
  HEX:= "3";
  HEX:= "4";
  HEX:= "5";
  HEX:= "6";
  HEX:= "7";
  HEX:= "8";
  HEX:= "9";
  HEX:= "A";
  HEX:= "B";
  HEX:= "C";
  HEX:= "D";
  HEX:= "E";
  HEX:= "F";
END CASE;
```

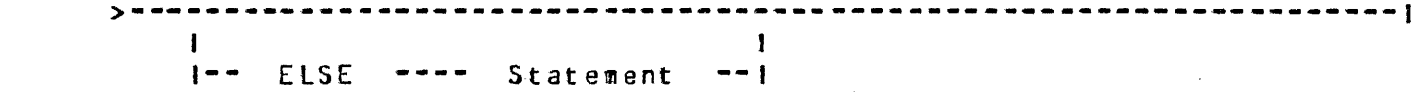
Labeled Case Statement





Labeled Statement List





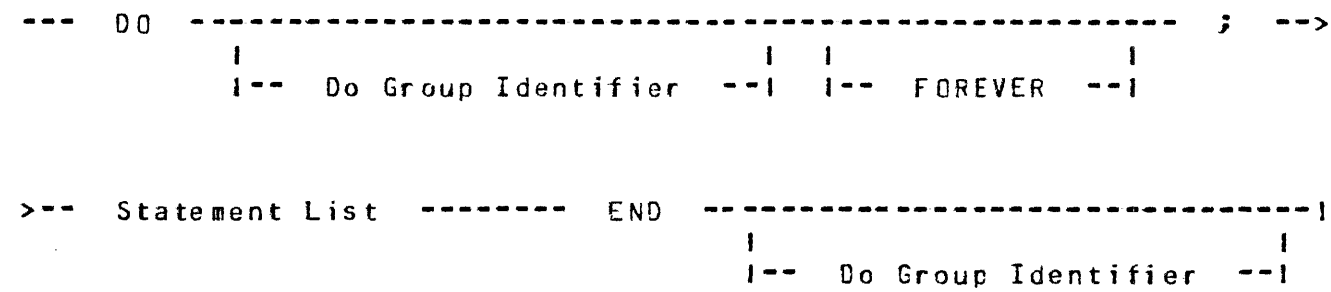
Label List

The labelec case statement selects for execution that statement whose label is equal to the value of Expression. After execution of the selected statement control passes to the statement following END CASE. If no statement has a label that is equal to the value of Expression, the Statement following ELSE is executed and then control passed to the statement following END CASE (unless an UNDO or RETURN statement was executed in the ELSE statement). If no ELSE clause is present, a run time error is generated. The value of label constants can be within the range 0 to 255, and may be of any type.

Example:

```
DECLARE
  CH         CHARACTER(1),
  CH_TYPE    MEMBER OF TYPE_SET;

CASE CH OF
  "0","1","2","3","4","5","6","7","8","9":
     CH_TYPE:= NUMERIC;
  ";":
     CH_TYPE:= SEMICOLON;
  ",":
     CH_TYPE:= COMMA;
  ELSE
     CH_TYPE:= UNKNOWN;
END CASE;
```

DO STATEMENT

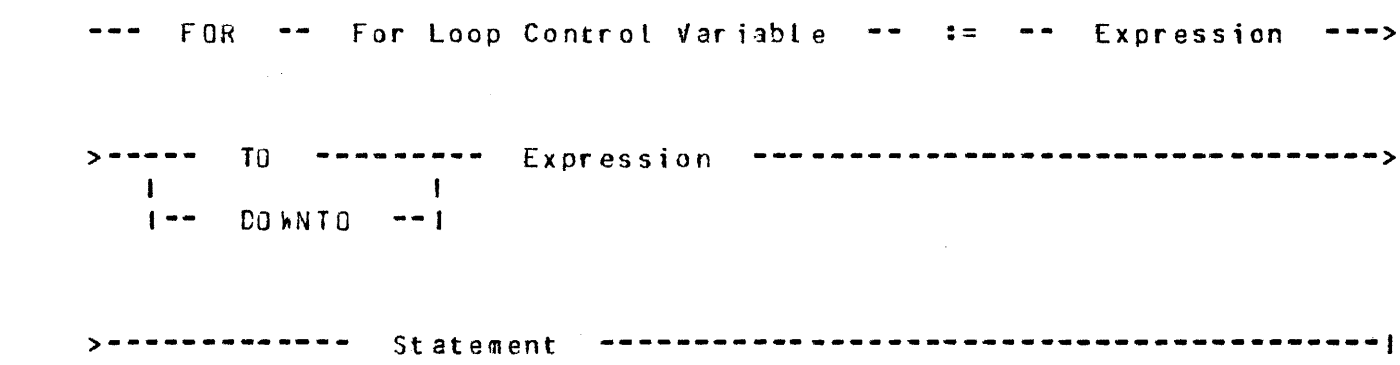


The DO group allows a list of statements to be grouped together. If FOREVER is present control will be passed back to the first statement from the end of the statement list. The DO group can be exited with an UNDO or RETURN statement.

Example:

```
DECLARE
  I                FIXED,
  A (ARRAY_SIZE) FIXED;

I:= 0;
DO INIT_A FOREVER;
  IF I = ARRAY_SIZE THEN
    DO;
      I:= 0;
      UNDO INIT_A;
    END;
  A(I):= I;
  I:= I + 1;
END INIT_A;
```

FOR STATEMENT

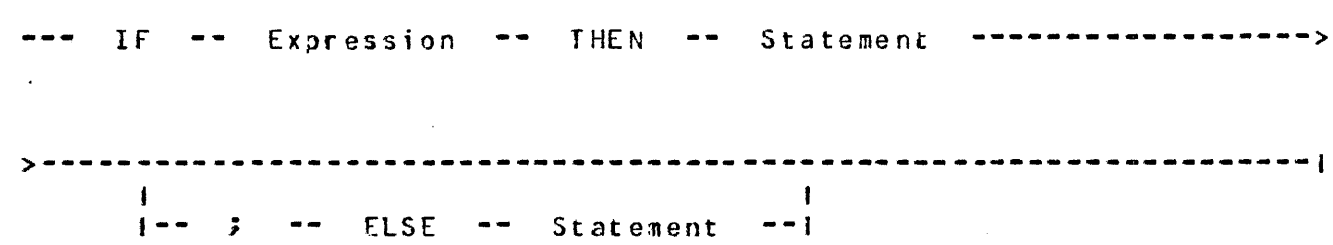


The For Loop Control variable must be a simple variable and of type FIXED. If the TO (DOWNTO) option is used the For Loop is exited when the For Loop Control variable is greater than (less than if DOWNTO) the Limit Expression. The exit condition is tested at the top of the loop.

Example:

```
DECLARE
  I                FIXED,
  A (ARRAY_SIZE)   FIXED;

FOR I:= 0 TO ARRAY_SIZE - 2 A(I):= I;
FOR I:= ARRAY_SIZE - 1 DOWNTO 1 A(I):= I;
```

IF STATEMENT



The Statement following THEN is executed if Expression yields a true result. If it is false, then either no statement or the statement following ELSE is executed.

When using nested IF statements an ELSE will be matched to the closest previously unmatched THEN.

Example:

```
DECLARE
  MAX       FIXED,
  VALUE_1   FIXED,
  VALUE_2   FIXED;

IF VALUE_1 >= VALUE_2 THEN
  MAX:= VALUE_1;
ELSE
  MAX:= VALUE_2;
```

REFER STATEMENT

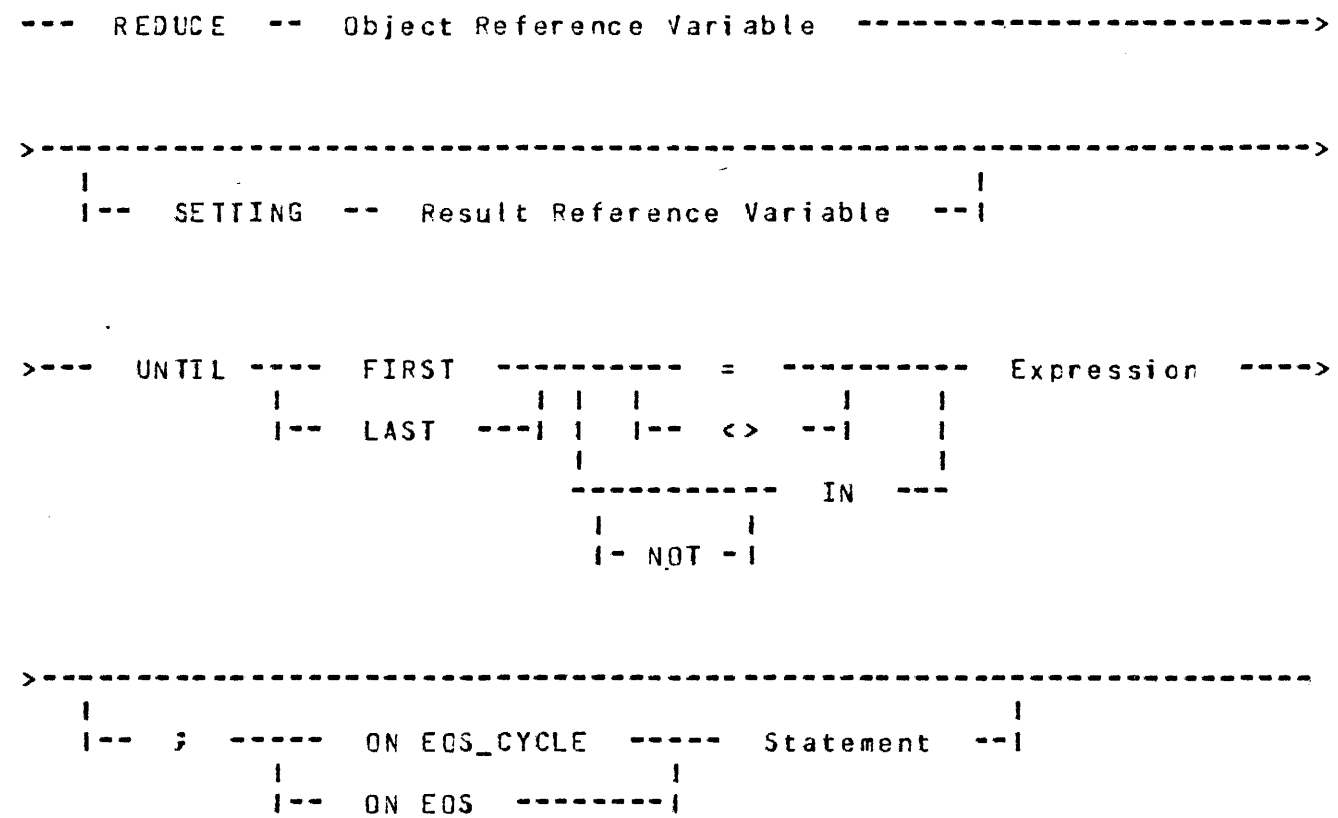
----- REFER -- Referent -- TO -- Address Generator ---------|

The REFER statement is used to change the address (and, except for pointers, length and type) of a reference variable, reference field, pointer variable or pointer field to map the data of another variable.

Example:

```
REFER TEXT TO BUFFER.TEXT;
```

REDUCE STATEMENT



The REDUCE statement is an efficient methoc of scanning character strings. The execution of a REDUCE statement does not change any data, the reference variables are adjusted tc point at a substring of the original string. The REDUCE statement scans from left to right if FIRST is specified and from right to left if LAST is specified. After execution of the REDUCE statement the Object Reference Variable is left describing the substring of the orignal Object Reference Variable that meets the condition of the reduction.

If the SETTING option is specified the Result Reference Variable will describe the substring of the original string that did not meet the condition of the reduction.

If the condition of the reduction is = or <>, the Expression must be a character string. It should be noted that only the first three characters of the Expression are compared to the string, but the length will be used to test for end of strirg conditions. If the condition of the reduction is IN, the Expression must be a SET of characters (See SETS). If the ON EOS_CYCLE or EOS condition is specified, Statement will be executed when the length of Object Reference Variable has been reduced tc zero.

After completion of Statement, control will be passed to the next statement if the ON condition was EOS, or back to the REDUCE Statement if the ON condition was EOS_CYCLE unless an UNDO or RETURN was executed in the ON Statement.

Example:

```
DECLARE
  NUMERICS            CHARACTER_SET
  ALPHAS              CHARACTER_SET
  SOURCE_TEXT         CHARACTER(72),
  TEXT      REFERENCE CHARACTER,
  TOKEN     REFERENCE CHARACTER;

PROCEDURE NEXT_RECORD;
  READ(SOURCE, SOURCE_TEXT);
  REFER TEXT TO SOURCE_TEXT;
END NEXT_RECORD;

REDUCE TEXT UNTIL FIRST <> " ";                                    % A
  ON EOS_CYCLE NEXT_RECORD;
IF SUBSTR(TEXT,0,1) IN NUMERICS THEN
  REDUCE TEXT SETTING TOKEN UNTIL FIRST NOT IN NUMERICS;           % B
ELSE IF SUBSTR(TEXT,0,1) IN ALPHAS THEN
  REDUCE TEXT SETTING TOKEN UNTIL FIRST NOT IN ALPHAS;             % C
ELSE DO SPECIAL_CHARACTER;
     REFER TOKEN TO SUBSTR(TEXT,0,1);
     REFER TEXT TO SUBSTR(TEXT,1);
     END SPECIAL_CHARACTER;
```

Assume SOURCE_TEXT contains " IF X = 999 THEN " and TEXT has been refered to SOURCE_TEXT. After execution of the REDUCE at line %A, TEXT will describe "IF X = 999 THEN ". As the first character of TEXT is in the set ALPHA, the REDUCE at line %C, will be selected. After execution of this statement, TEXT will describe " X = 999 THEN " and TOKEN will describe "IF".

REPEAT STATEMENT

----- REPEAT ----- Statement List ----- UNTIL ----- Expression -----|

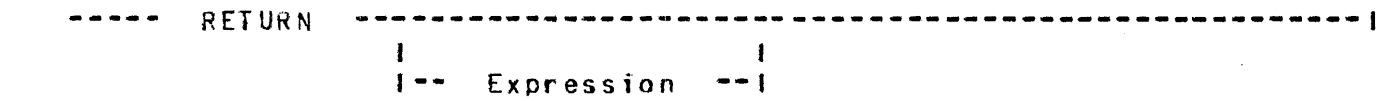
Statement List will be executed repetitively until Expression returns a true result. The exit condition is tested at the end of each loop, so Statement List will be executed at least once.

Example:

```
REPEAT

  DO_SOMETHING;
  IF END_CONDITION
    THEN DONE := TRUE;

UNTIL DONE;
```

DO STMT THEN TEST

RETURN STATEMENT

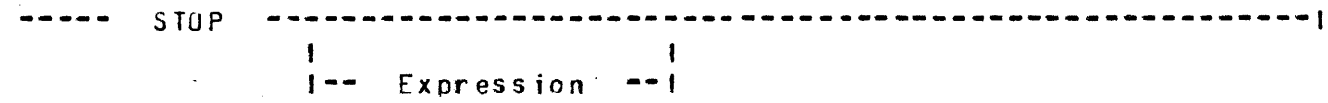


The RETURN statement allows an explicit return from a procedure. If the procedure is typed then Expression must be present and will be the value returned from the procedure. If the procedure is not typed then the presence of Expression is an error. An implicit return is generated by the compiler at the end of every procedure. If the procedure is typed a value matching the procedure's type is returned (0:if FIXED, BIT or RECORD, and a null string if CHARACTER).

Example:

```
PROCEDURE P;
  .
  RETURN;
  .
END P;

PROCEDURE Q FIXED;
  .
  RETURN 0;
  .
END Q;
```

STOP STATEMENT

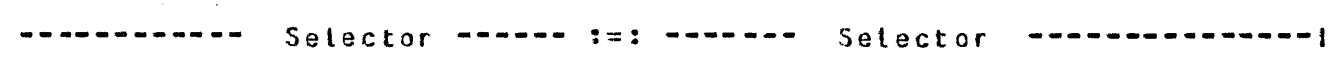


The STOP statement will generate a terminate communicate to the MCP to end the execution of the program. Expression is intended for use by compilers to communicate the number of syntax errors to the MCP.

Example:

```
STOP;
STOP SYNTAX_ERROR_COUNT;
```

SWAP STATEMENT

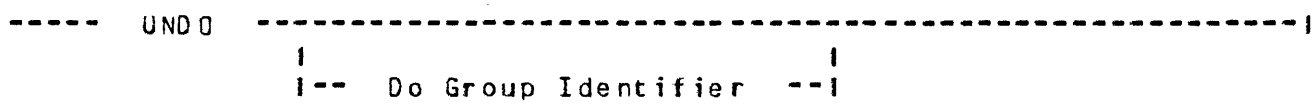


The swap statement will exchange the values of the two fields.

Example:

```
BUFFER_1 :=: BUFFER_2;
```

UNDO STATEMENT



The UNDO statement transfers control to the end of the current DO group, or to the end of the DO group specified by Group Identifier.

Example:

```
DECLARE
  I                FIXED,
  A (ARRAY_SIZE) FIXED;

I:= 0;
DO INIT_A FOREVER;
  IF I = ARRAY_SIZE THEN UNDO INIT_A;
  A(I):= I;
  I:= I + 1;
END INIT_A;
```

test & then do

WHILE STATEMENT

Statement will be executed repetitively while the Expression yields a true result. Expression is evaluated before each iteration.

Example:

```
DECLARE
  I                FIXED,
  A (ARRAY_SIZE)   FIXED;

I:= 0;
WHILE I < ARRAY_SIZE
  DO;
  A(I):= I;
  I:= I + 1;
  END;
```

WITH STATEMENT

The type of Selector must be RECORD. The WITH Statement opens a new scope containing the field identifiers of the Record type of Selector.

Within Statement a field of selector is selected by specifying only its field name, without preceeding it with the entire Selector expression.

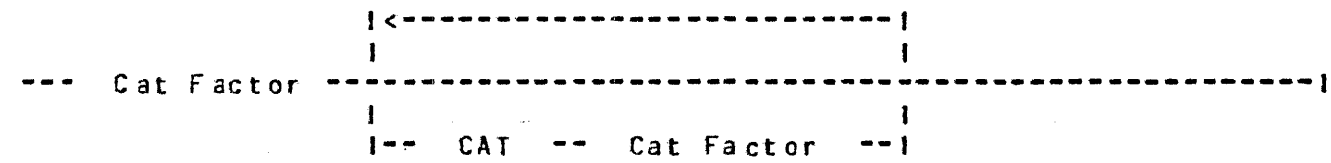
Example:

```
        P(I).FIELD_1 :=X;
        P(I).FIELD_2 :=Y;
        P(I).FIELD_3 :=Z;
is equivalent to:
        WITH P(I) DO;
          FIELD_1 :=X;
          FIELD_2 :=Y;
          FIELD_3 :=Z;
          END;


        WITH A, B, C
is equivalent to:
        WITH A WITH B WITH C
```

EXPRESSIONS

Expressions are the rules for calculating a value using constants, variables, functions and operations. The conventional rules of left to right evaluation and operator precedence are observed, as indicated by the syntax graphs.

Expression

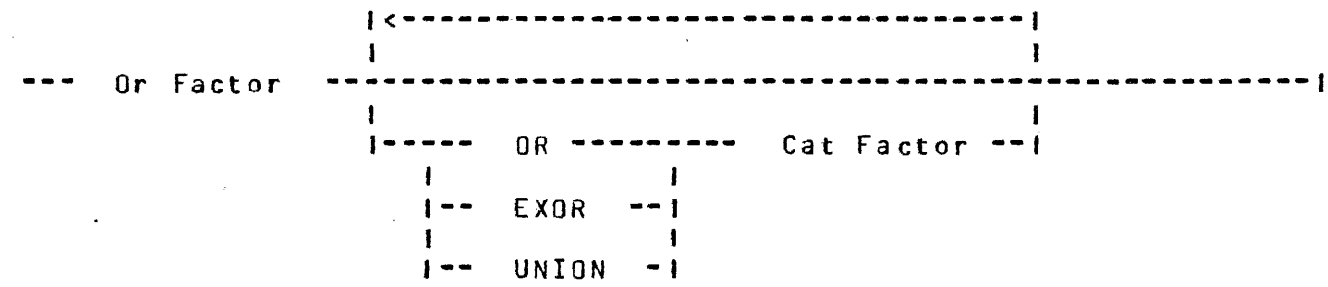


The CAT operator will concatenate two fields together as one field. The resulting type will be character if both fields are character; for all other combinations it will be bit.

Example:

```
BUFFER:= DECIMAL(LINE_NUMBER,4) CAT ":" CAT TEXT CAT ":";
```

Cat Factor



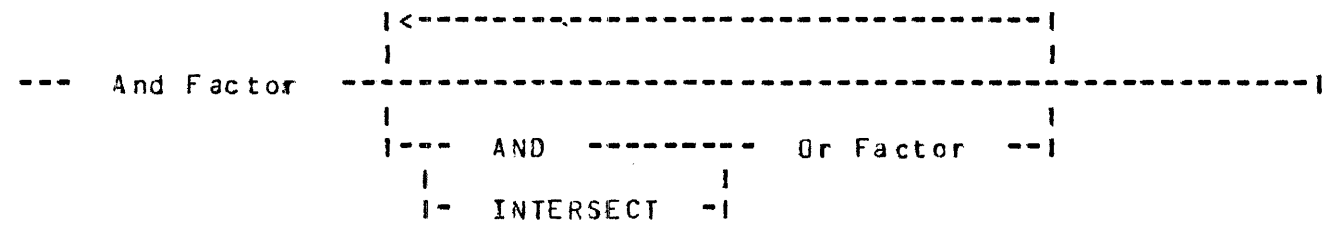
The OR/EXOR operation performs a logical OR/EXOR of two operancs. UNION performs a set union of two operands.

The result will always be of type bit.

Example:

```
DECLARE
        (ALPHAS,
         NUMERICS,
         ALPHA_NUMERICS)  CHARACTERS_SET;
ALPHAS:= [ "0" TO "9" ];
NUMERICS:= [ "A" TO "I", "J" TO "R", "S" TO "Z" ];
ALPHA_NUMERICS:= ALPHAS UNION NUMERICS;
```

Or Factor



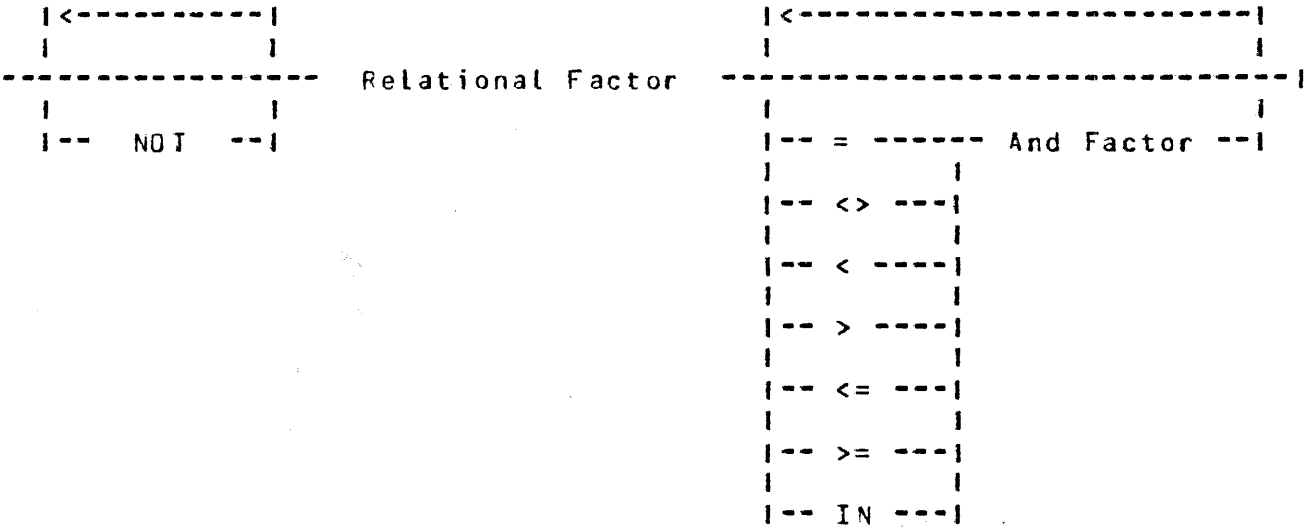
The AND operation performs a logical AND of two operands. The INTERSECT operation performs a set intersection of two operands.

The result will always be of type bit.

Example:

```
SET SUITS = CLUBS, DIAMONDS, HEARTS, SPADES;

DECLARE
       (PLAYER_A, PLAYER_B, MATCHING_SUITS) SUITS
MATCHING_SUITS:= PLAYER_A INTERSECT PLAYER_B
```

And Factor

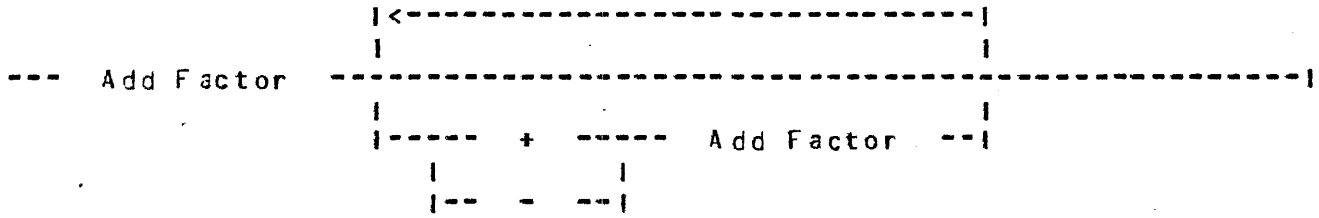


The relational operators leave a result that has a value of 1 if the condition is true or 0 if the condition is false. The not operator will perform a logical not on one operand, and will always have a type of bit.

Example:

```
A = B
```

Relational Factor

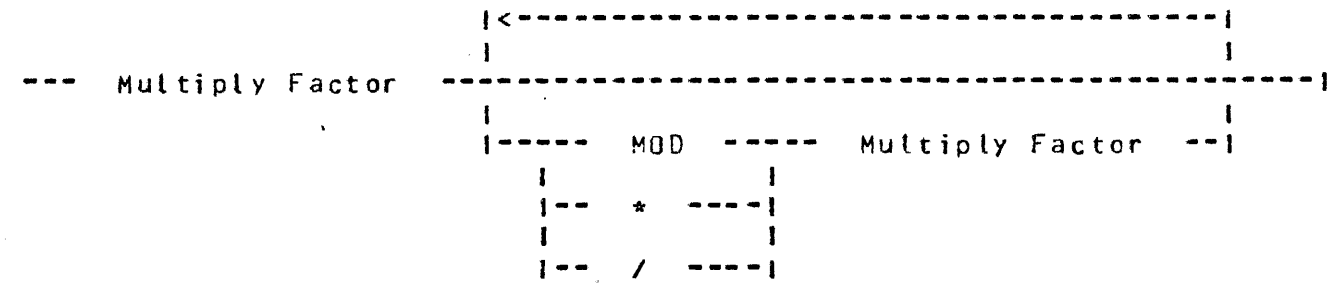


The + and - operations perform 24 bit, two's compliment arithmetic on two operands and result in a value of type FIXED.

Example:

```
A + B
A - B
```

Add Factor

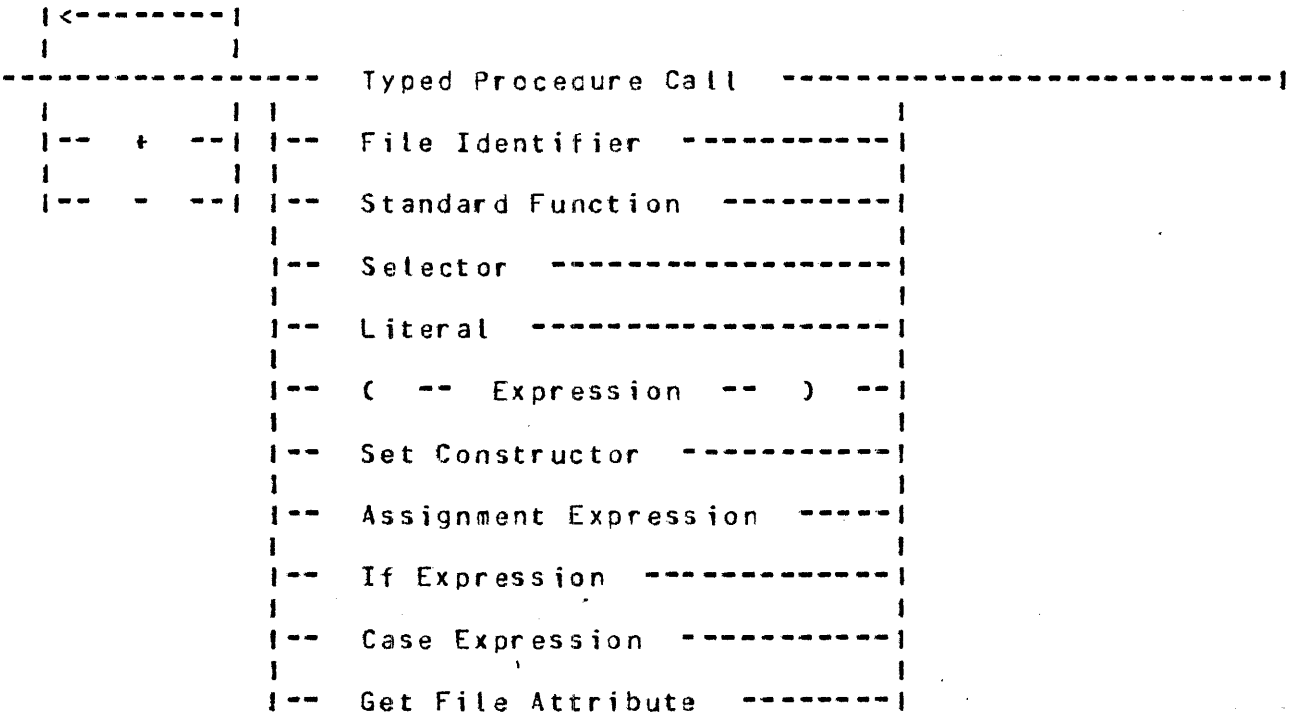


The MOD * and / operators perform 24 bits, two's compliment arithmetic on two operands and result in a value of type FIXED. The result will be an integer result with the remainder being ignored for division (i.e. 9/4 = 2). The MOD operation is division resulting in the integer value of the remainder (i.e. 9 MOD 4 =1).

Example:

```
A MOD B
A  *  B
A  /  B
```

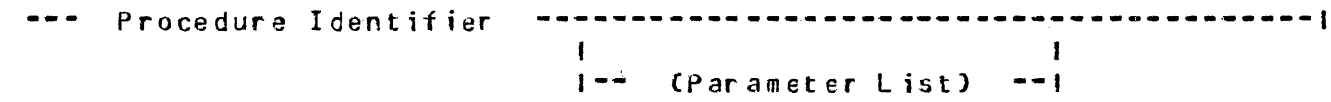
Multiply Factor



The unary - operation will negate the value of an operand. The unary + does not change the operand.

See INPUT/OUTPUT Section for Get File Attribute.

Typed Procedure Call

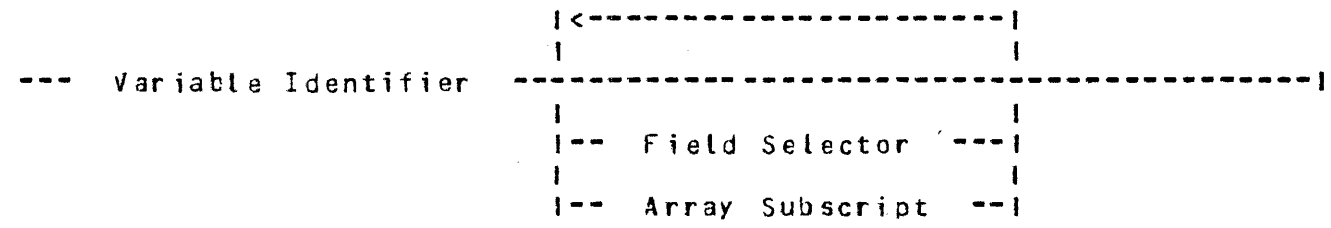


The Typed Procedure Call is used to call a procedure that returns a value, and to use that value in further Expression evaluation.

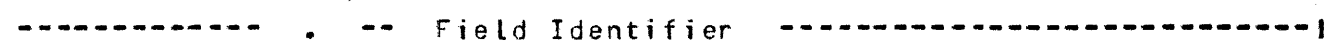
Example:

```
MAXIMUM := MAX(X,Y);
```

Selector



Field Selector



Array Subscript

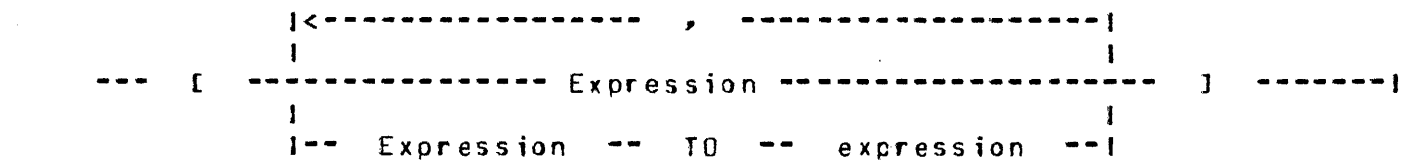
A Selector is an address generator that selects a sub-field of a variable.

If an array is referenced with a subscript that is less than 0 or greater than or equal to the array bound a run time error will be generated.

Example:

```
RECORD R1
  LEN          FIXED,
  OFFSET       FIXED;

RECORD R2
  IDS (10)     R1,
  TYPE         FIXED;

DECLARE
  A (10)       R2;

A(I).IDS(J).LEN:= 0;
```

Set Constructor



The Set Constructor expression is used to build sets from their members. The form 'Expression TO Expression' will include all members of the set bounded by the two expressions. If the first expression is greater than the second it denotes an empty set. Remember that set members are declared in descending order.

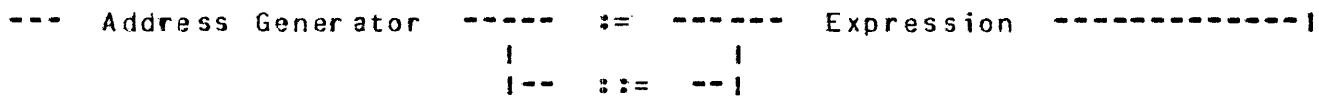
Example:

```
DECLARE
    NUMERICS            CHARACTER_SET,
    ALPHAS              CHARACTER_SET,
NUMERICS:= ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"];
ALPHAS:= ["A" TO "I", "J" TO "R", "S" TO "Z"];
```

Example:

```
SET STATES = FINAL, MIDDLE, INITIAL;

DECLARE
    STATE_SET           STATES,
    STATE               MEMBER OF STATES;
STATE_SET := [ INITIAL TO FINAL ];
WHILE STATE IN STATE_SET COMPUTE_STATE;
```

Assignment Expression



The Assignment Expression stores the value of Expression into Address Generator and leaves either the value (if := is used) or the Address Generator (if ::= is used) as an operand for further expression evaluation.

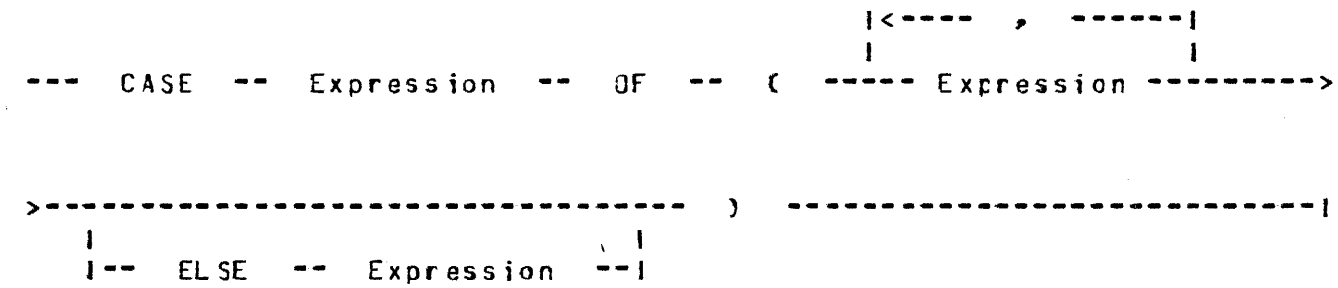
Example:

```
I:= -1;
DO FOREVER;
     A(I:= I+1) :=I;
     IF I = ARRAY_SIZE THEN UNDO;
END;
```

Return (A := B)

Return (A ::= B)

Case Expression

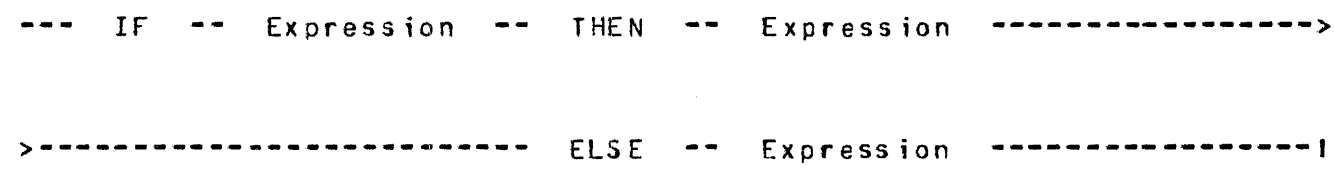


The Case Expression selects an Expression from a list of expressions to be evaluated. The value of CASE Expression is used as an index into the list of Expression. If there are N Expressions in the list the range is from 0 to N-1. If the value of the Expression following CASE is greater than the number of elements in the list, the Expression following ELSE will be used. If Expression is out of range and the ELSE is not present, a run time error will be generated.

Example:

```
I:= CASE X OF ( X+1, X+2, X+3, X+4 );
```

If Expression

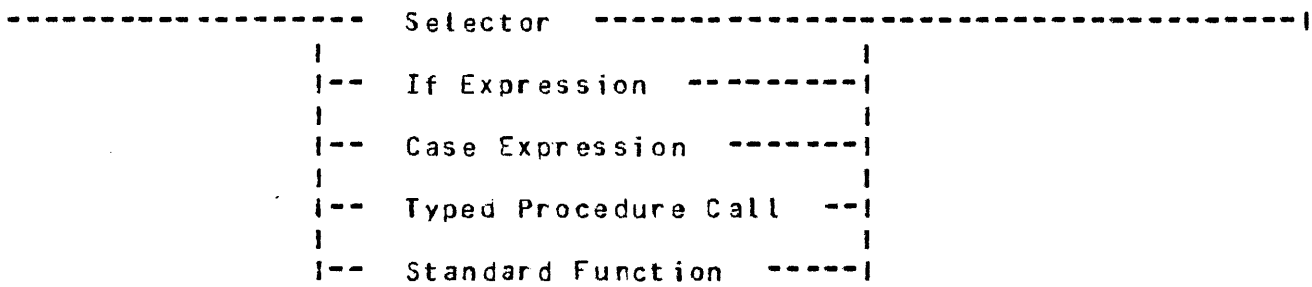


The IF Expression is used to select an Expression for evaluation.

Example:

```
A := IF X<Y THEN X ELSE Y;
```

Address Generator



An If Expression is considered an address generator if both the Expression following THEN and the Expression following ELSE are address generators.

A Case Expression is considered to be an address generator if all elements of the case Expression list are address generators.

A Typed Procedure Call is considered an address generator if its result type is reference.

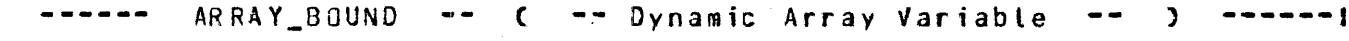
STANDARD PROCEDURES and FUNCTIONS

SDL2 provides a set of procedures and functions to perform commonly required operations.

The Identifiers are not reserved and may be redeclared, in which case the user declared name takes precederce over the pre-defined one.

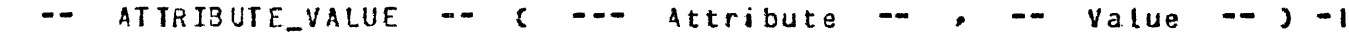
STANDARD FUNCTIONS

Array Bound



Returns the number of elements in Dynamic Array Variable as a FIXED value.

Attribute Value



Attribute may be any file attribute allowed in a Get/Put Attribute statement. Value must be a predefined mnemonic value for that attribute. This function returns a fixed number which is the MCP-assigned number corresponding to the attribute value.

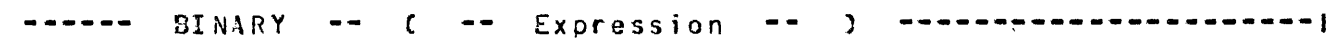
Example:

```
IF INFILE.KIND = ATTRIBUTE_VALJE(KIND,DISK)
  THEN DISPLAY "INPUT IS FROM DISK";
```

Base Register

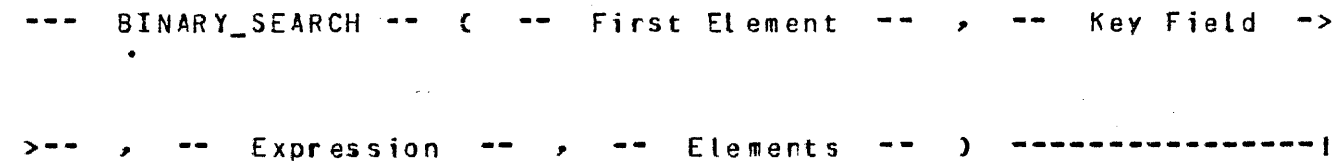
Returns the absolute memory location of the base of the program's data space as a FIXED value.

Binary



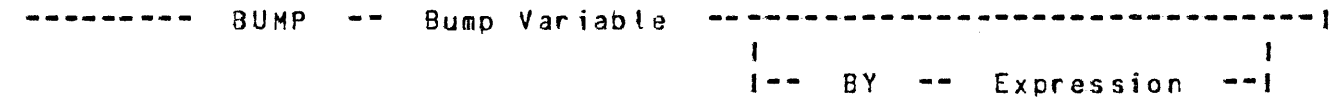
Returns a FIXED value which is the binary representation of Expression. Expression is assumed to be a character string.

Binary Search



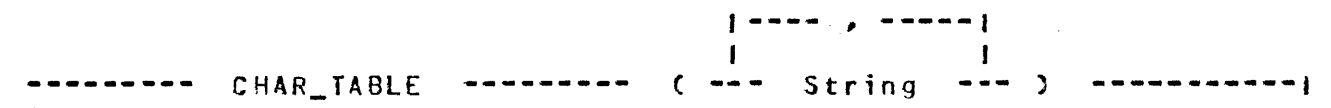
First Element must be an Address Generator and have a type of RECORD. Key Field is a field of the RECORD type of First Element that is used to compare against Expression. Elements is an expression whose value is used to give the bound of the search. The element number (the First Element is 0) of the element whose Key Field is greater than Expression is returned. The Elements must be ordered.

Bump



Bump Variable must be an address generator. If Expression is present Expression is added to Bump Variable, otherwise 1 is added to Bump Variable. The result is stored into Bump Variable and returned. (See also BUMP in Standard Procedures).

Character Table



String should be a quoted string of characters. CHAR_TABLE will return a Character Set whose members include all characters specified in String.

Code Address

```
--------- CODE_ADDRESS -- ( -- Procedure Identifier -- ) ---|
```

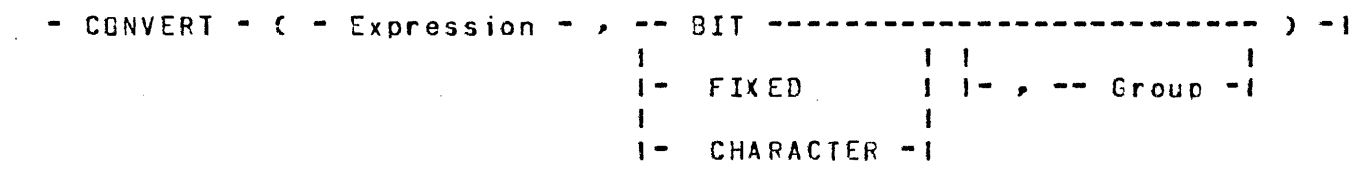
Returns the code address of Procedure Identifier as a 32 bit value. (See 'SDL2 S-LANGUAGE PRODUCT SPECIFICATION for the format of a code address).

Communicate With Gismo

```
------ COMMUNICATE_WITH_GISMO -- ( -- Expression -- ) ------|
```

Transfers control to GISMO passing Expression as a parameter and returns a value. (See 'MCP PRODUCT SPECIFICATION' for formats of parameters and values returned). (See also Communicate With Gismo in Standard Procedures).

Convert



Expesssion is converted to the type specified, according to these rules:

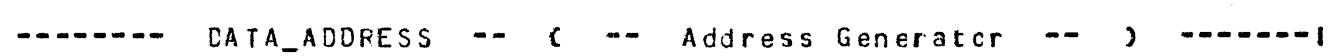
BIT to FIXED:	Up to 24 of the rightmost bits are converted.
FIXED to BIT:	The result is BIT (24).
CHARACTER to FIXED:	Leading blanks and sign are allowed. Up to seven of the rightmost characters are converted.
FIXED to CHARACTER:	The result is eight characters. The sign and any leading zeros are not supressed.
BIT to CHARACTER:	The radix = 2 ** Group.
CHARACTER to BIT:	The radix = 2 ** Group.

Group is only used with BIT to CHARACTER or CHARACTER to BIT conversions. It specifies the number of bits in the bit string which correspond to a character in the character string. The default Group is 4 (hexadecimal).

For example:

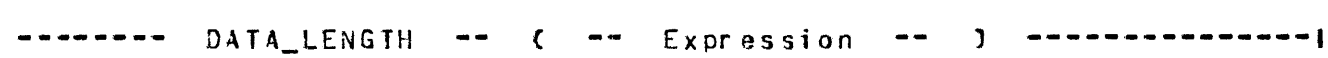
```
CONVERT ("-72581", FIXED)          = -72581
CONVERT (@(3)752@, CHARACTER,4)    = "1E4"
CONVERT (@(1)11011@, FIXED)        = 27
CONVERT ("132", BIT, 2)            = @(2)132@
CONVERT ("132", BIT, 4)            = @(4)132@
CONVERT ("2", BIT)                 = @(4)2@
CONVERT (-4, BIT)                  = @(4)FFFFFC@
CONVERT (-4, CHARACTER)            = "-0000004"
CONVERT (@0A0@, CHARACTER)         = "0A0"
```

Data Address



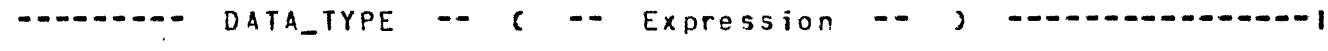
Returns the base relative address of Address Generator as a FIXED value.

Data Length



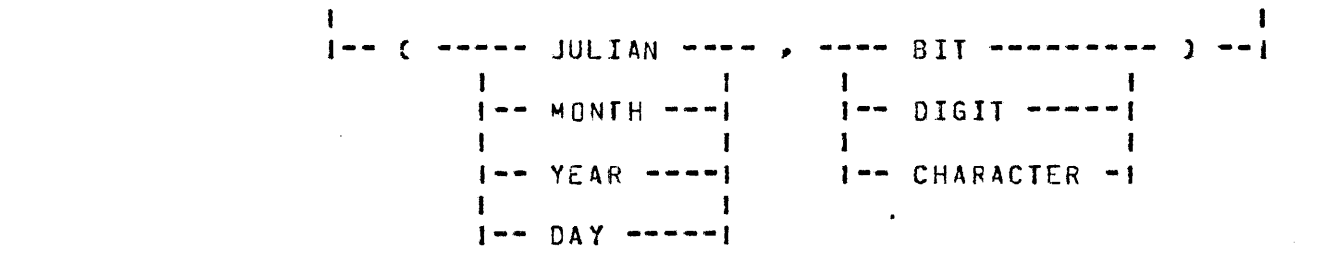
Returns the bit length of Expression as a FIXED value.

Data Type



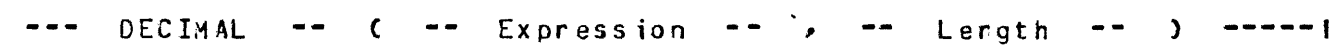
Returns the type bits of Expression. (See 'SDL2 S-LANGUAGE PRODUCT SPECIFICATION').

Date



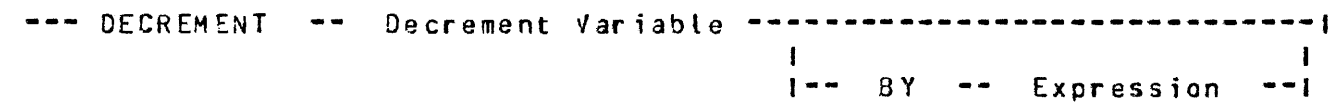
Returns a string containing the current date. The default format is (MONTH, CHARACTER).

Decimal



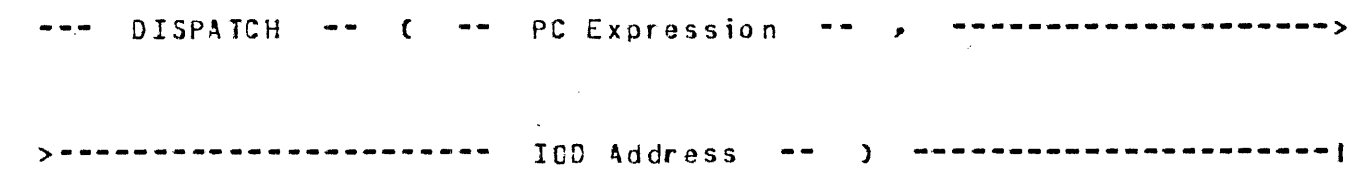
Length is an Expression. Returns a character string of Length characters containing the decimal representation of Expression.

Decrement



Decrement Variable must be an Address Generator. Expression (1 if BY Expression is omitted) is subtracted from Decrement Variable and the result is stored into Decrement Variable and also returned. (See also Decrement in Standard Procedures).

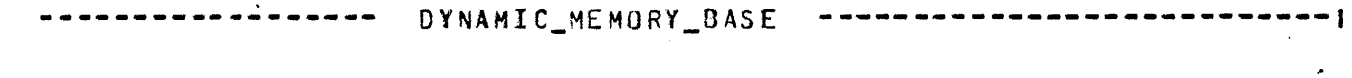
Dispatch



PC Expression should generate a value containing the Port and Channel. (See 'GISMO PRODUCT SPECIFICATION' for layout of the Port and Channel information).

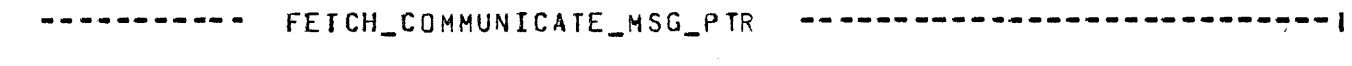
IOD Address should be an Expression whose value is the absolute address of the I/O descriptor to be dispatched. (See 'MCP PRODUCT SPECIIFICATION' for layout of an I/O descriptor).

Dynamic Memory Base



Returns the offset from the base register of the beginning of dynamic memory as a FIXED value.

Fetch Communicate Message Pointer

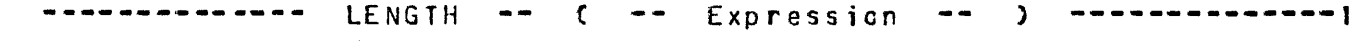


Returns RS.REINSTATE_MSG_PTR. (See 'MCP PRODUCT SPECIFICATION' for layout of the Run Structure and meanings of the values returned).

Hex Sequence Number

Returns a 32 bit value containing the sequence number of the current line in hex digits.

Length

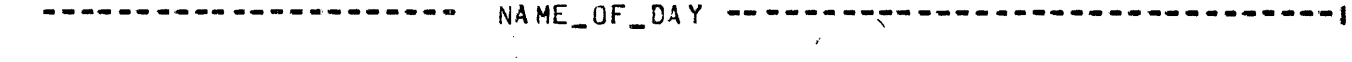


Returns a FIXED value containing the length of Expression. If the type of Expression is characters the value will be the number of characters, otherwise it will be the number of bits.

Limit Register

Returns a FIXED value containing the base to limit size in bits of the program.

Name of Day

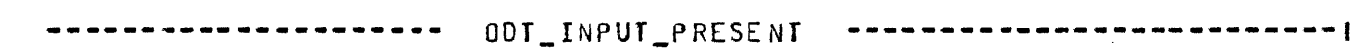


Returns a nine-character string containing the name of the day of the week.

Null

Returns a null character string. (Note: this car be used as a selector).

Odt Input Present

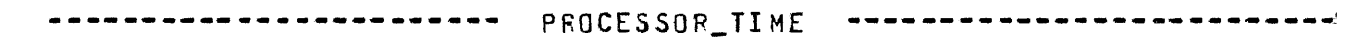


Returns a true value if there are any outstanding messages from an operator accept. (See 'AX' command in the 'MCP PRODUCT SPECIFICATION').

Overlay

See 'MCP PRODUCT SPECIFICATION' for more information. (For MCP use only).

Processor Time

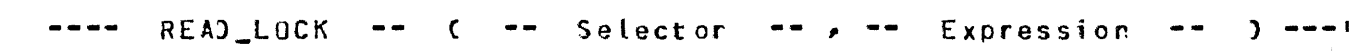


Returns a FIXED value containing the amount of processor time that has been allocated to this program in tenths of a second.

Program Switches

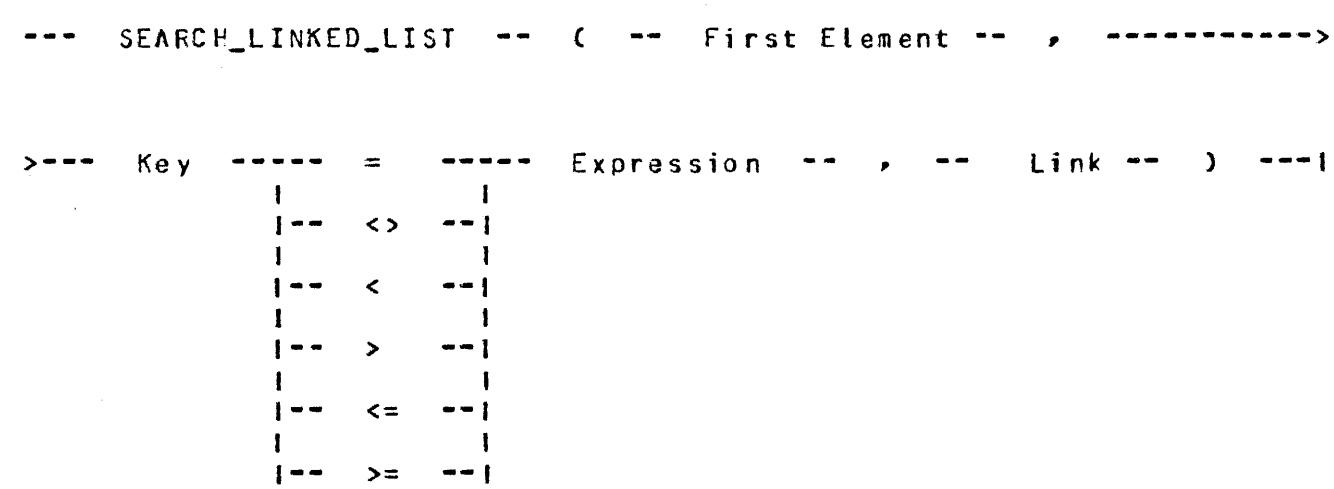
Returns a 40 bit value containing the Program Switches, unless Expression is specified. If Expression is specified, it must evaluate to a value between 0 and 9, and the function will return a 4 bit value containing that program Switch.

Read Lock



Expression is stored into Selector, and the original value of Selector is returned.

Search Linked List



First Element must be an Address Generator with a type of RECORD. Key and Link must be fields within the RECORD type of the First Element. A search is performed starting at First Element until either the relation is true or a link of all F's is encountered. The selected element is returned; if no match was found an address generator with a data address of all F's is returned.

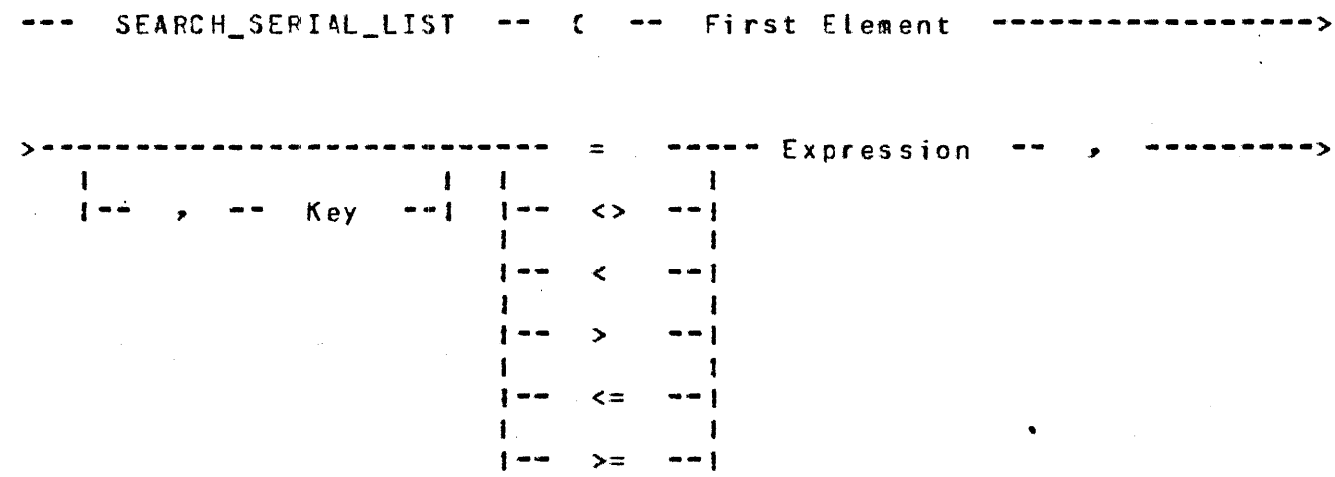
Example:

```
   RECORD  INFO
             BACK_LINK       BIT(24),
             FORWARD_LINK    BIT(24),
             ID              BIT(20),
             DATA            BIT(100);

   DECLARE  FIRST  INFO,
            CURRENT REFERENCE INFO;

REFER CURRENT TO SEARCH_LINKED_LIST (FIRST, ID = PATTERN, FORWARD_LINK)

IF DATA_ADDRESS (CURRENT) = @FFFFFF@
  THEN DISPLAY "PATTERN NOT FOUND";
```

Search SDL Stacks

Lower must be an Expression whose value is the base relative address of a structure. Upper must be an Expression whose value is the base relative address of the end of the structure. The base relative address of a descriptor that points within the bounds of Lower and Upper is returned. If no descriptor is found 0 is returned.

Search Serial List

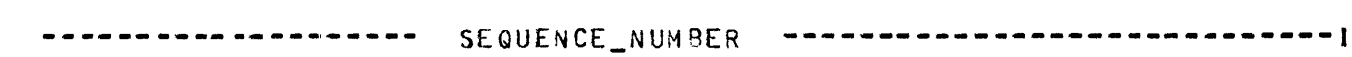


First Element must be an Address Generator and if Key is used must be of type RECORD. Key must be field of the RECORD type of the First Element. If key is not used the complete element is used. Elements is an expression describing the bound of the search. A serial search is performed starting at First Element until either the relation is true or the end of the table is encountered. The element number (First Element is 0) of the selected element is returned. If no match was found Elements is returned.

Example:

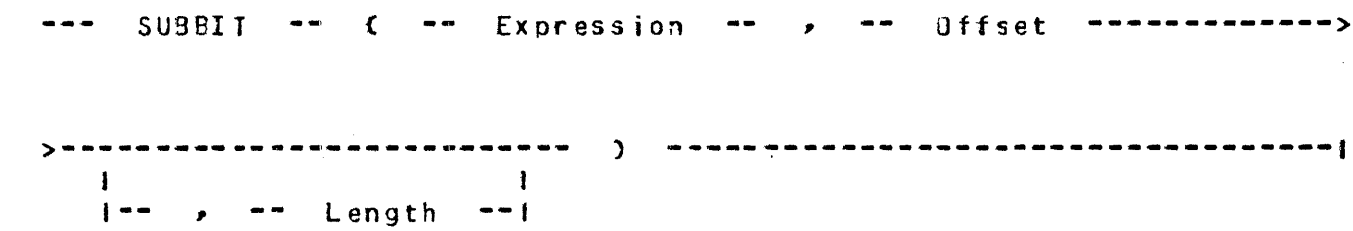
```
RECORD  JOB_INFO
          JOB_NO    BIT(16)
          NAME      CHARACTER(30)
          PRIORITY  BIT(4);
DECLARE JOB(MAX_JOBS)   JOB_INFO,
        I               FIXED;

I := SEARCH_SERIAL_LIST (JOB(0), JOB_NO = THIS_JOB, MAX_JOBS);

IF I = MAX_JOBS
   THEN DISPLAY "JOB NOT FOUND";
```

Sequence Number



Returns an 8 character string containing the value of the sequence number of the current source line.

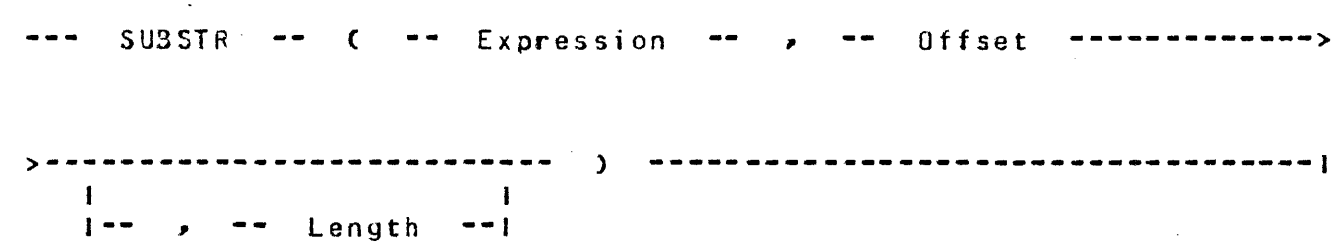
Subbit



Expression is assumed to be of type BIT. Returns a subfield of the value of Expression starting at Offset bits for Length Bits. If Length is not present the length is calculated to be the remaining bits. The type will always be BIT.

(SUBBIT may be used as an Address Generator, in which case Expression must be an Address Generator).

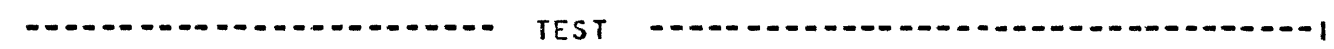
Substr



Expression is assumed to be of type CHARACTER. Returns a subfield of the value of Expression starting at Offset characters for Length characters. If Length is not present the length is calculated to be the remaining characters. The type will always be CHARACTER.

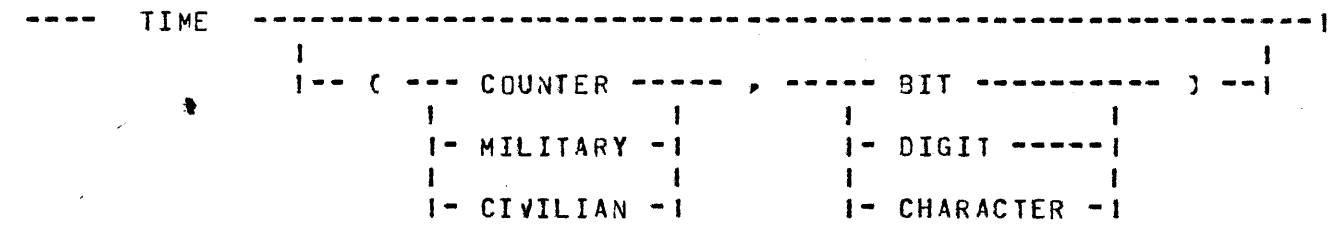
(SUBSTR may be used as an Address Generator, in which case Expression must be an Address Generator).

Test



For compiler and interpreter debugging only. (See also Standard Procedures).

Time



Returns a string containing the current time of day. The default format is (CIVILIAN, CHARACTER).

Timer

---------- TIMER ----------

Returns a FIXED value with the current setting of the TIME REGISTER.

Todays_Date

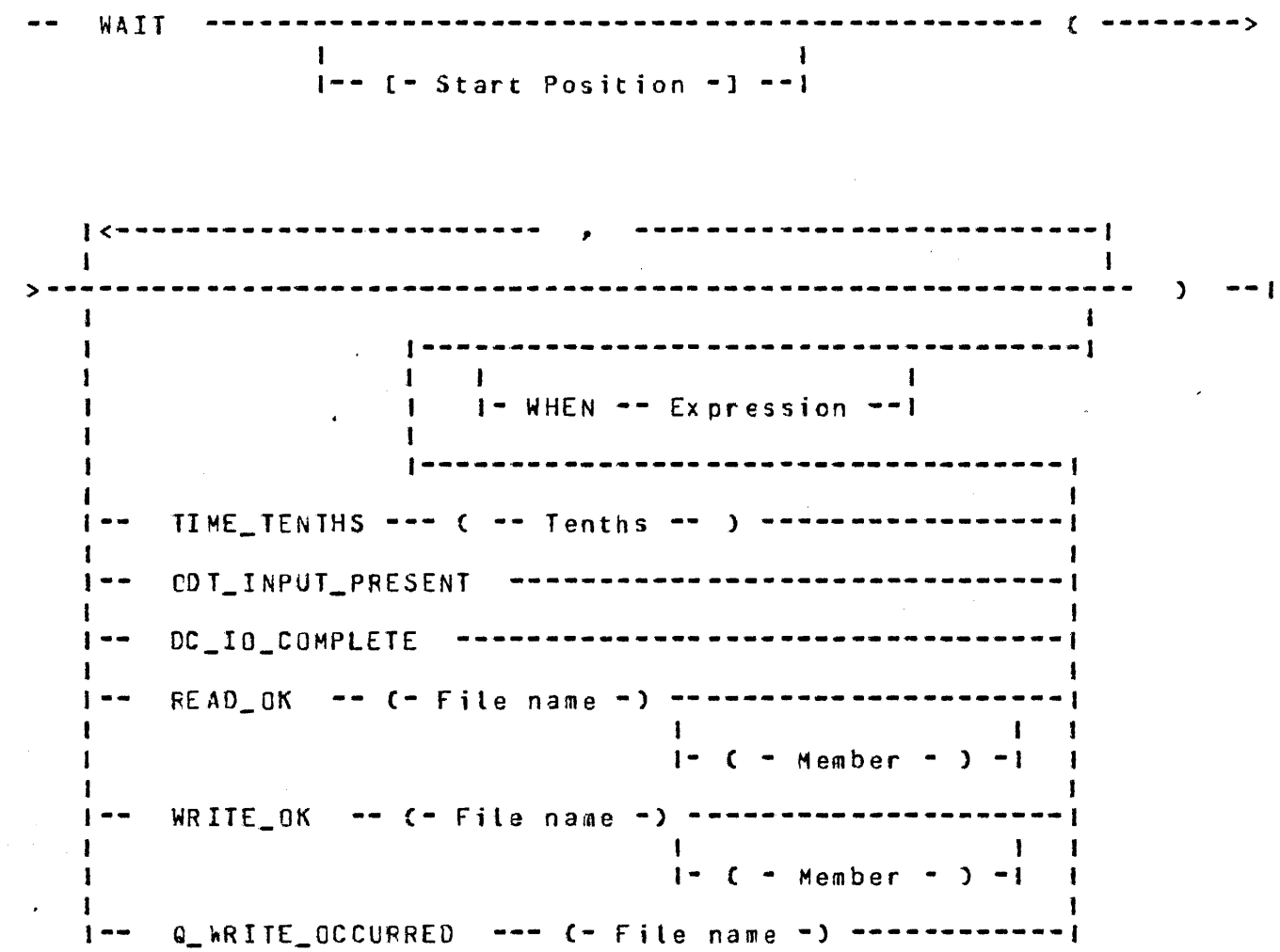
---------- TODAYS_DATE ----------

Returns a 17 character string containing the date of compilation of this program.

The format of the string is:

```
RECORD TODAYS_DATE_FORM
  MONTH              CHARACTER(2),  % 01 TO 12
  SLASH_1            CHARACTER(1),  % LITERALLY "/"
  DAY                CHARACTER(2),  % 01 TO 31
  SLASH_2            CHARACTER(1),  % LITERALLY "/"
  YEAR               CHARACTER(2),  % 00 TO 99
  FILLER             CHARACTER(1),  % LITERALLY " "
  HOUR               CHARACTER(2),  % 01 TO 12
  COLON_1            CHARACTER(1),  % LITERALLY ":"
  MINUTE             CHARACTER(2),  % 00 TO 59
  FILLER             CHARACTER(1),  % LITERALLY " "
  AM_PM              CHARACTER(2);  % LITERALLY "AM" OR "PM"
```

wait

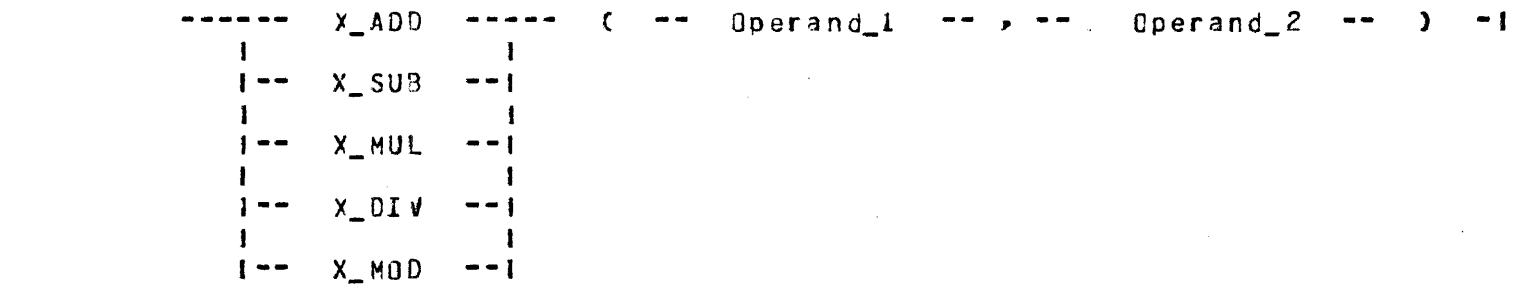


The program will be suspended until one of the wait events becomes true. The value returned is the ordinal position of the event in the wait event list (first is 0). Start Postion can be used to specify which event in the list to test first. The default is to start at event 0.

If WHEN Expression is present the event will be included in the wait list only if expression yields a true result, however a null event will be inserted in the list to maintain the ordinal position of the events.

If TIME_TENTHS is specified, it must be the first event in the list.

Extended Arithmetics



X_ADD adds Operand_1 to Operand_2 and returns the result as type bit.

X_SUB subtracts Operand_2 from Operand_1 and returns the result as type bit.

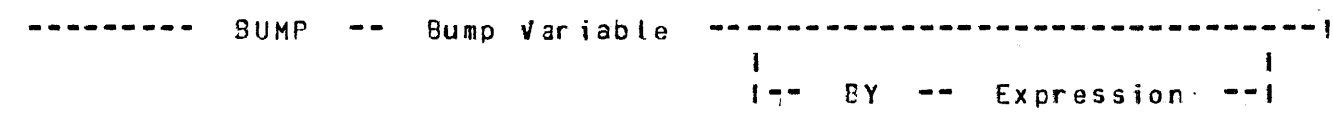
X_MUL multiplies Operand_1 by Operand_2 and returns the result as type bit.

X_DIV divides Operand_1 by Operand_2 and returns the result as type bit.

X_MOD modulo divides Operand_1 by Operand_2 and returns the result as type bit.

STANDARD PROCEDURES

Bump

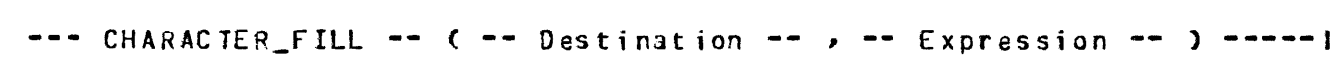


Bump Variable must be an address generator. If Expression is present Expression is added to Bump Variable, otherwise 1 is added to Bump Variable. The result is storec into Bump Variable. (See also BUMP in Standard Expressions).

Call

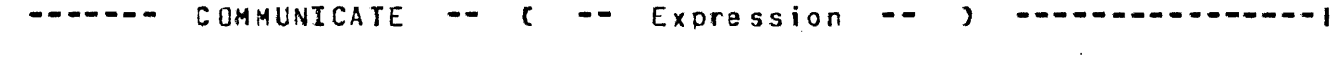
Performs an indirect call. Expression must generate a 32 bit value containing the code address to be called (See 'SDL2 S-LANGUAGE PRODUCT SPECIFICATION' for format of a code address).

Character Fill



Destination must be an Address Generator. The high order 8 bits (first byte) of Expression will be used to fill Destination.

Communicate



Transfers control to the MCP passing Expression as a parameter. (See 'MCP PRODUCT SPECIFICATION' for formats of parameters in a communicate).

Communicate With Gismo

```
------ COMMUNICATE_WITH_GISMO -- ( -- Expression -- ) ------|
```

Transfers control to GISMO passing Expression as a parameter. (See 'MCP PRODUCT SPECIFICATION' for formats of parameters). (See also Communicate With Gismo in Standard Functions).

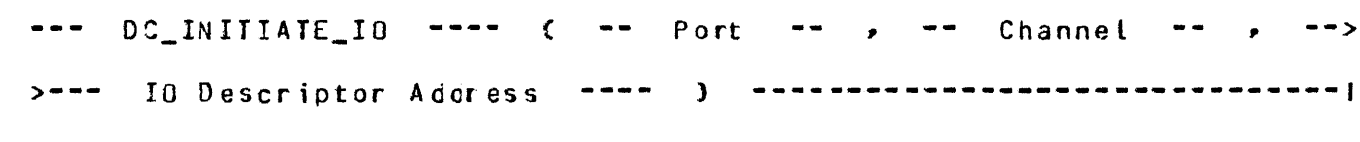
Compile Card Information

```
--- COMPILE_CARD_INFO ----- ( -- Destination -- ) ----------|
```

Destination must be an Address Generator. Information about the program is returned into Destination in the following format:

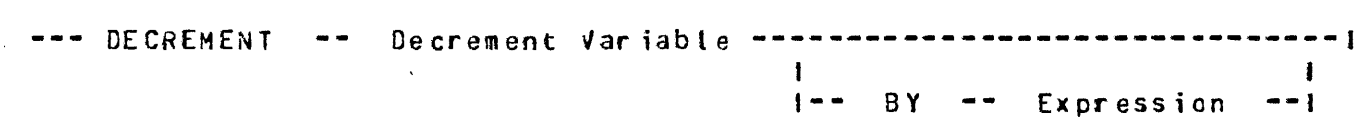
```
RECORD COMPILE_CARD_INFO_FORM
  OBJECT NAME                                  CHARACTER(30),
  EXECUTE TYPE                                 CHARACTER(2),
        %01 = execute
        %02 = compile and go
        %03 = compile for syntax
        %04 = compile to library
        %05 = compile and save
        %06 = go part of compile and go
        %07 = go part of compile and save
  COMPILER PACK IDENTIFIER                     CHARACTER(10),
  INTERPRETER NAME                             CHARACTER(30),
  INTRINSIC NAME                               CHARACTER(10),
  PRIORITY                                     CHARACTER(2),
  SESSION NUMBER                               CHARACTER(6),
  JOB NUMBER                                   CHARACTER(6),
  COMPILER MFID AND FIO                        CHARACTER(20),
  CHARGE NUMBER                                CHARACTER(7),
  FILLER                                       CHARACTER(1),
  DATE AND TIME COMPILED                       BIT(36),
  FILLER                                       BIT(4),
  USERCODE                                     CHARACTER(10),
  PASSWORD                                     CHARACTER(10),
  PARENT JOB NUMBER                            CHARACTER(4),
  PARENT QUEUE IDENTIFIER                      CHARACTER(20),
  LOG SPO                                      CHARACTER(1);
```

Datacomm Initiate I/O



IO Descriptor Address must be the base-relative address of an IO descriptor which will be initiated on the specified Port and Channel. (See 'MCP PRODUCT SPECIFICATION' for more information).

Decrement



Decrement Variable must be an Address Generator. Expression (1 if BY Expression is omitted) is subtracted from Decrement Variable and the result is stored into Decrement Variable. (See also Decrement in Standard Functions).

Disable Interrupts

All interrupts will be disabled until an ENABLE_INTERRUPTS is executed. (For MCP use only).

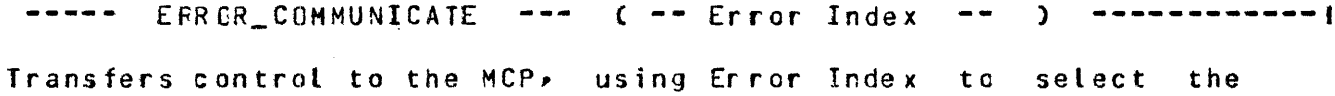
Dump

Generates a memory dump of the program.

Enable Interrupts

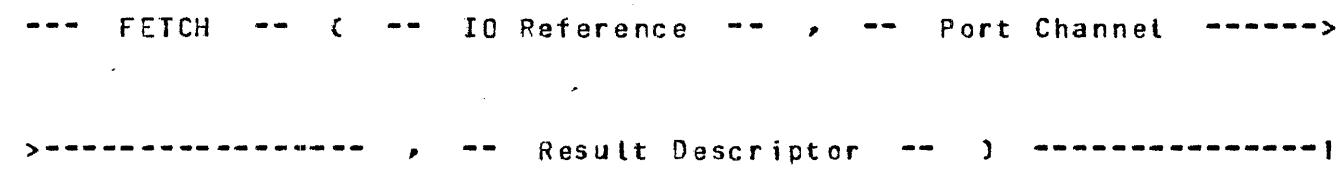
If interrupts are disabled they will be enabled ard normal processing of interrupts will continue. (For MCP use only).

Error Communicate



Transfers control to the MCP, using Error Index to select the appropriate run-time error message. Error Index must be a fixed constant.

Fetch

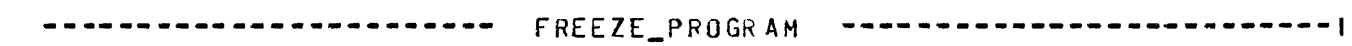


IO Reference must be an Expression whose value is the reference address of an I/O descriptor. Port Channel must be an Address Generator. Result Descriptor must be an Address Gererator.

The Standard Procedure FETCH fetches the result of an I/O operation. If there is a high priority interrupt, then that interrupt will be reported. Otherwise, if IO Reference is non-zero then only an interrupt on an I/O descriptor with that reference address will be reported. If not found then the first interrupt to occur will be reported. The port and channel number will be stored into Port Channel and the address of the result descriptor will be stored into Result Descriptor.

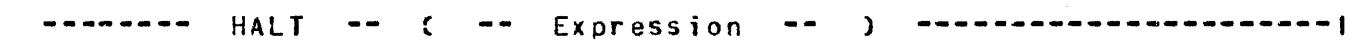
(See 'MCP PRODUCT SPECIFICATION' for more information on interrupts and I/O descriptors).

Freeze Program



This procedure prevents the program from being moved to a different memory location, or from being rolled out of memory.

Halt

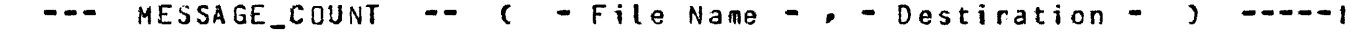


The machine will halt with the low order 24 bits of Expression displayed. (See 'SDL2 S-LANGUAGE PRODUCT SPECIFICATION' for more information).

Hardware Monitor

See 'SDL2 S-LANGUAGE PRODUCT SPECIFICATION' and 'B1000 PROCESSOR PRODUCT SPECIFICATION' for more information.

Message Count



File Name must be the name of a queue file, and Destination must be an Address Generator. The number of messages in the queue will be returned as a fixed number into Destination. If File Name refers to queue file family, an array of values, one for each family member, will be returned into Destination.

Refer Address

--- REFER_ADDRESS -- (-- Ref Variable -- , -- Expression --) ---|

Ref Variable must be a Reference or Pointer Variable. Expression will be stored into the address part of the descriptor for Ref Variable.

Refer Bound

--- REFER_BOUND -- (-- Ref Array -- , -- Expression --) ----|

Ref Array must be a paged array that has not been referenced, or a reference array. Expression will be stored into the array bound part of the descriptor for Ref Array.

Refer Length

--- REFER_LENGTH -- (-- Ref Variable -- , -- Expression --) -|

Ref Variable must be a Reference variable that is not of type RECORD. Expression will be stored into the length part of the descriptor for Ref Variable.

Refer Page Shift

--- REFER_PAGE_SHIFT -- (-- Ref Array -- , -- Expression --) ---|

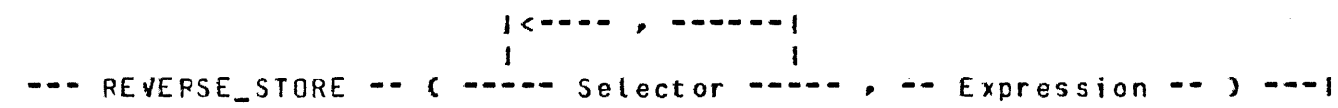
Ref Array must be a paged array that has not been referenced. Expression will be stored into the page shift part of the descriptor for Ref array.

Refer Type

--- REFER_TYPE -- (-- Ref Variable -- , -- Expression --) --|

Ref Variable must be a reference variable that is not of type RECORD. Expression will be stored into the type part of the descriptor for Ref Variable.

Reverse Store



Multiple store operations will be generated, evaluated from left to right.

Example:

```
REVERSE_STORE(A,B,C,1);
```

is the same as

```
A:= B; B:= C; C:= 1;
```

Test

```
------------------------- TEST ---------------------------------
```

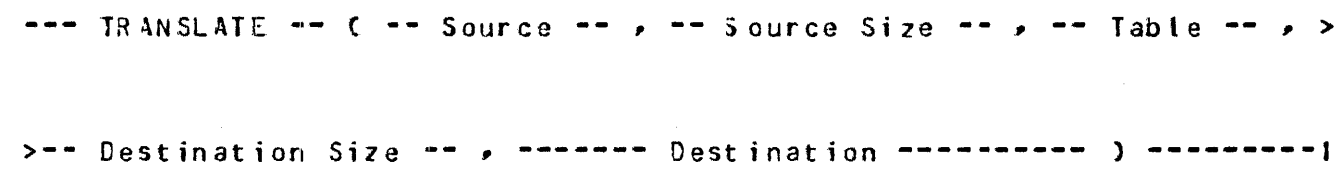
For compiler and interpreter debugging only. (See also Standard Functions).

Thaw Program

```
----------------------- THAW_PROGRAM -----------------------------
```

This procedure unfreezes the program, and allows it to be moved or rolled out of memory. It does not force the program to be rolled out.

Translate



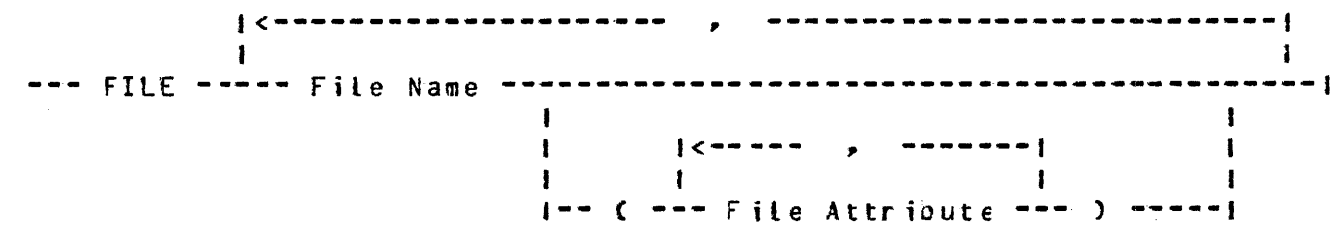
Source and Destination must be Address Generators. Source Size, Table and Destination Size are Expressions. Source is assumed to consist of items of Source Size (in bits). Table and Destination are assumed to consist of items of Destination Size (in bits). Each item in Source is used as a subscript into Table to obtain an item which is placed into Destination in the same ordinal position as the source.

Zip

The Zip statement allows the program to pass control instructions to the MCP. Expression should generate a character string whose value is a valid MCP control command as documented in 'MCP OPERATOPS MANUAL'.

INPUT/OUTPUT

FILE DECLARATIONS

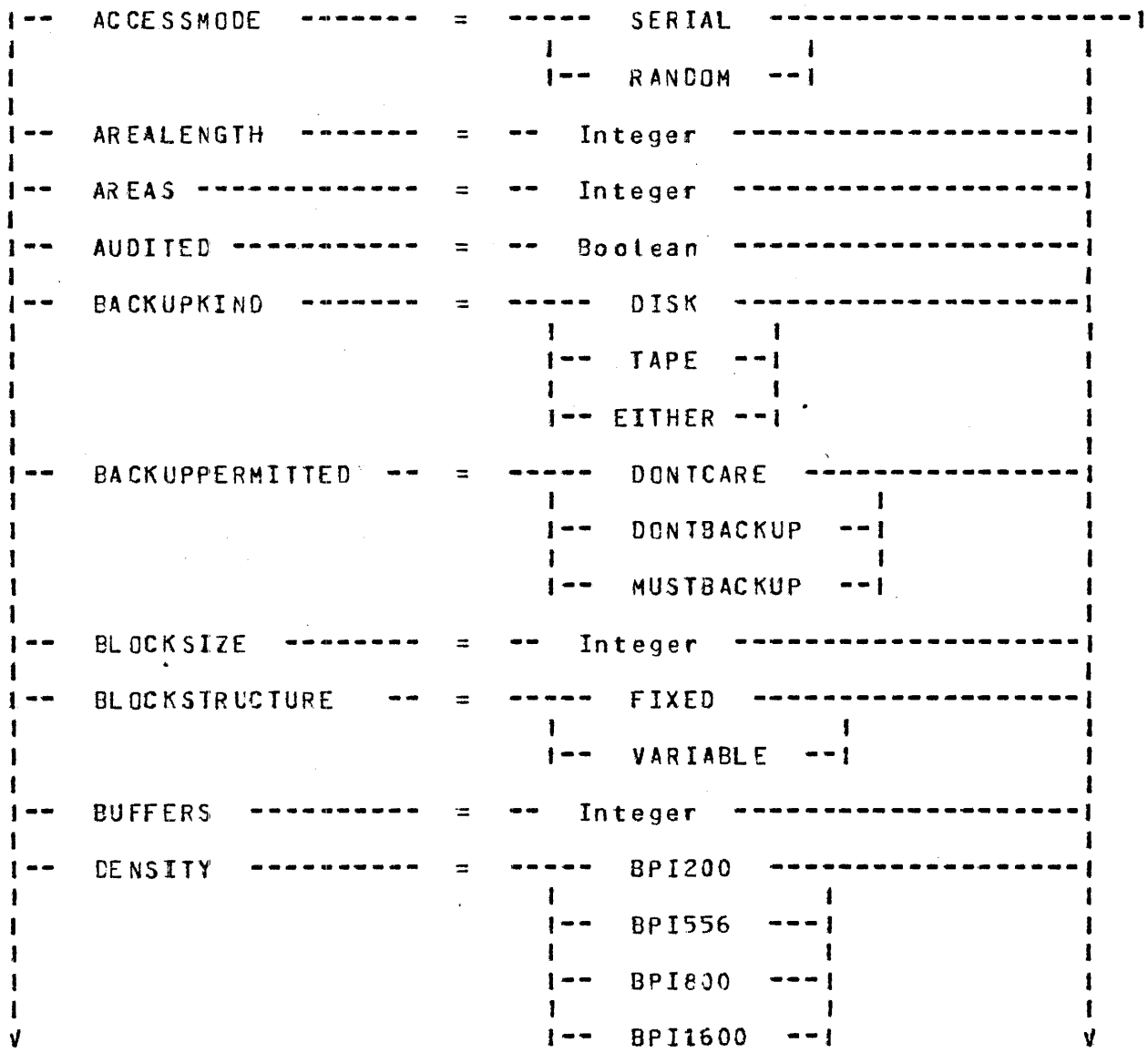


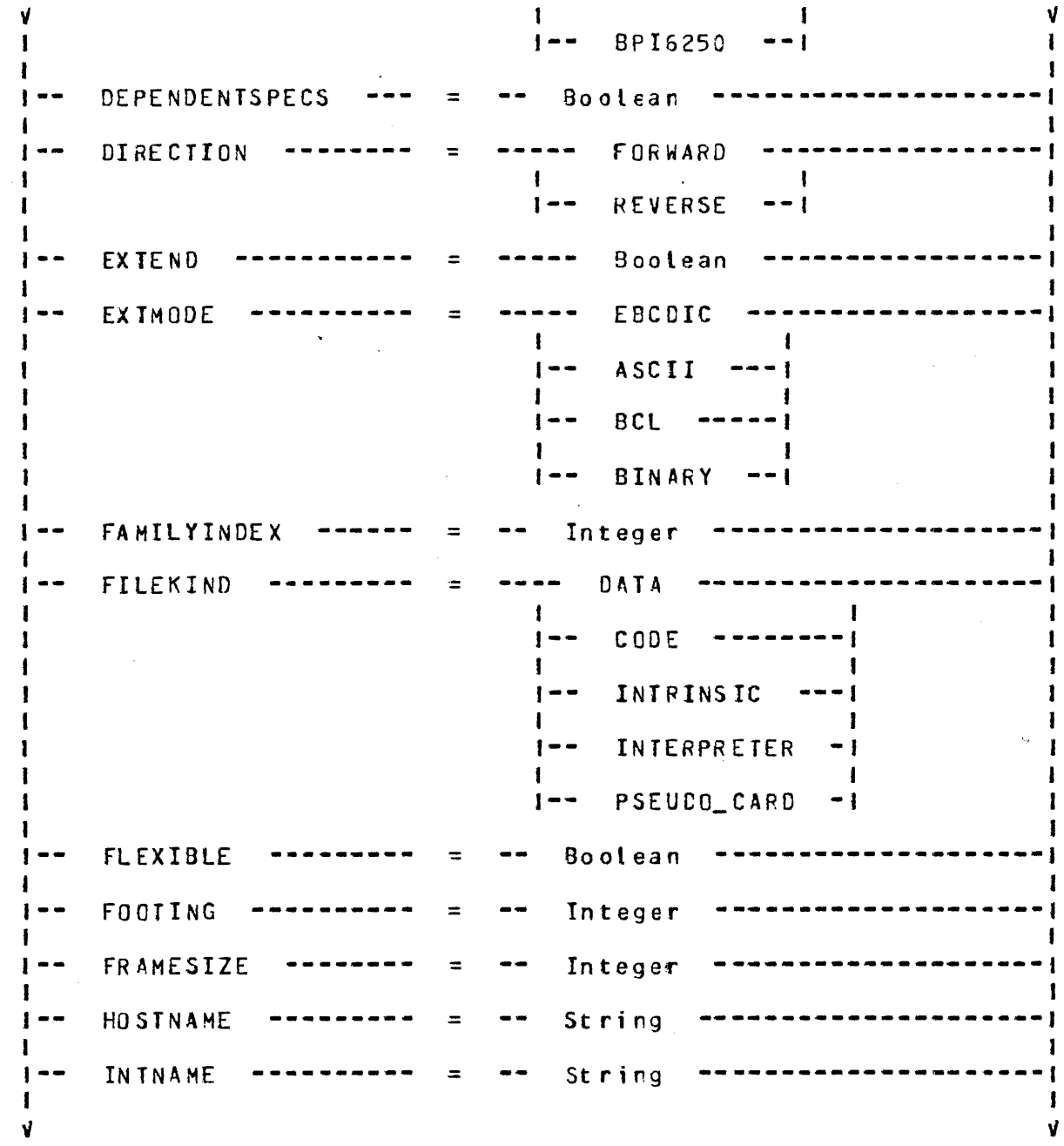
The FILE declaration will generate an FPB with the default file attributes modified by the attribute list. The File Name, when used in an Expression, will yield its FPB number (the first file declared will be 0).

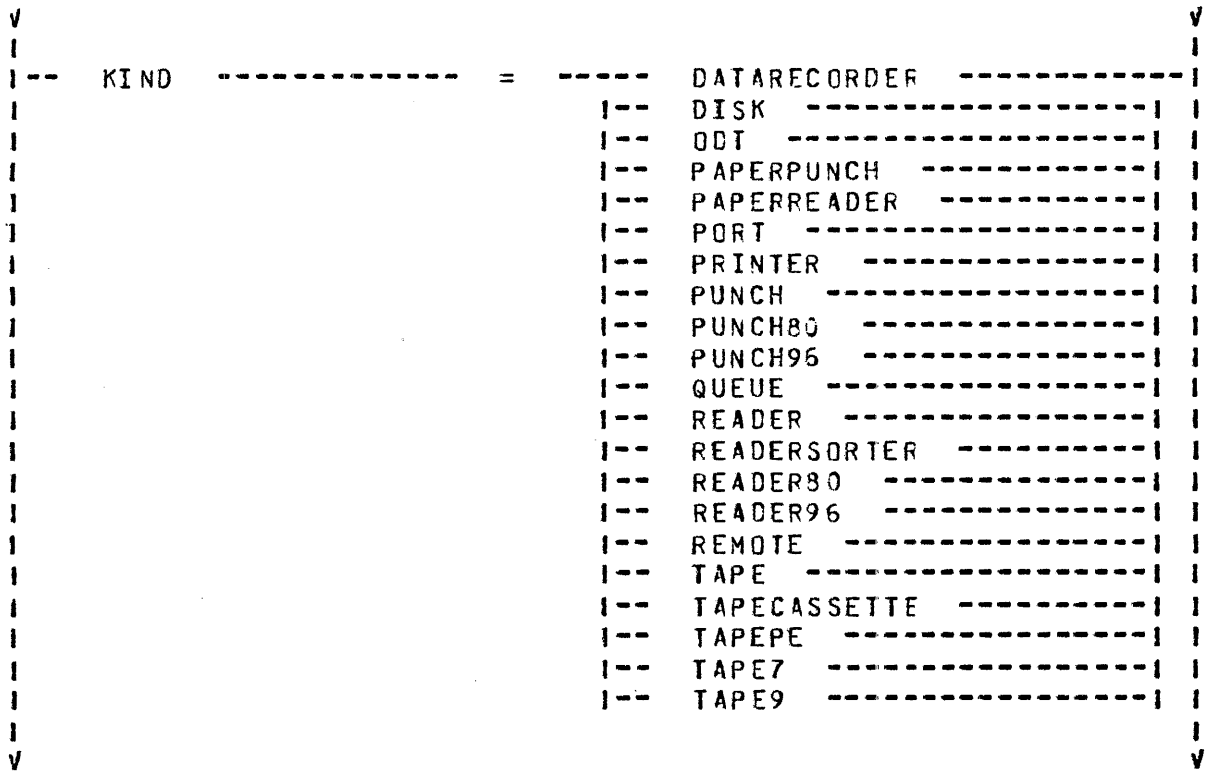
File Attributes

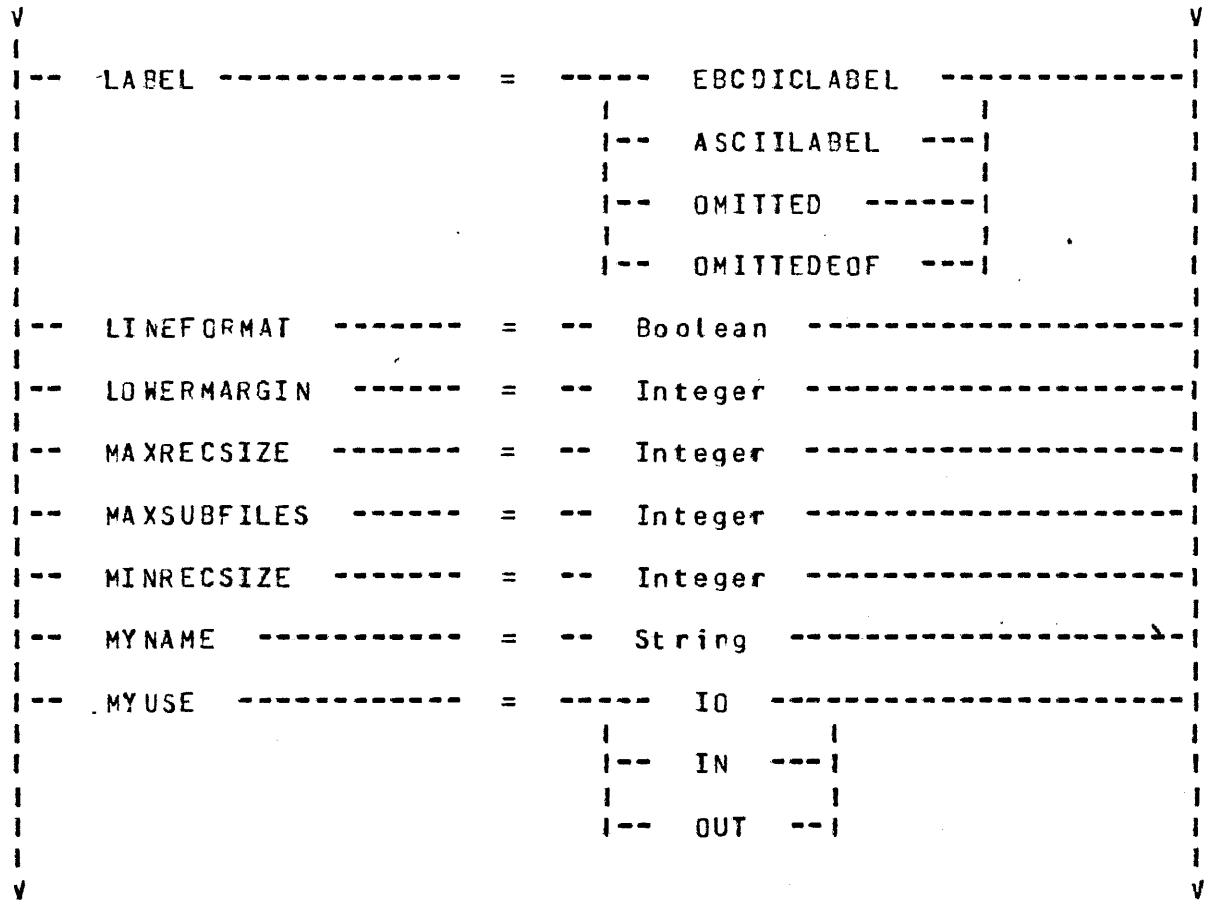
The following list of file attributes are those that are applicable for file declarations.

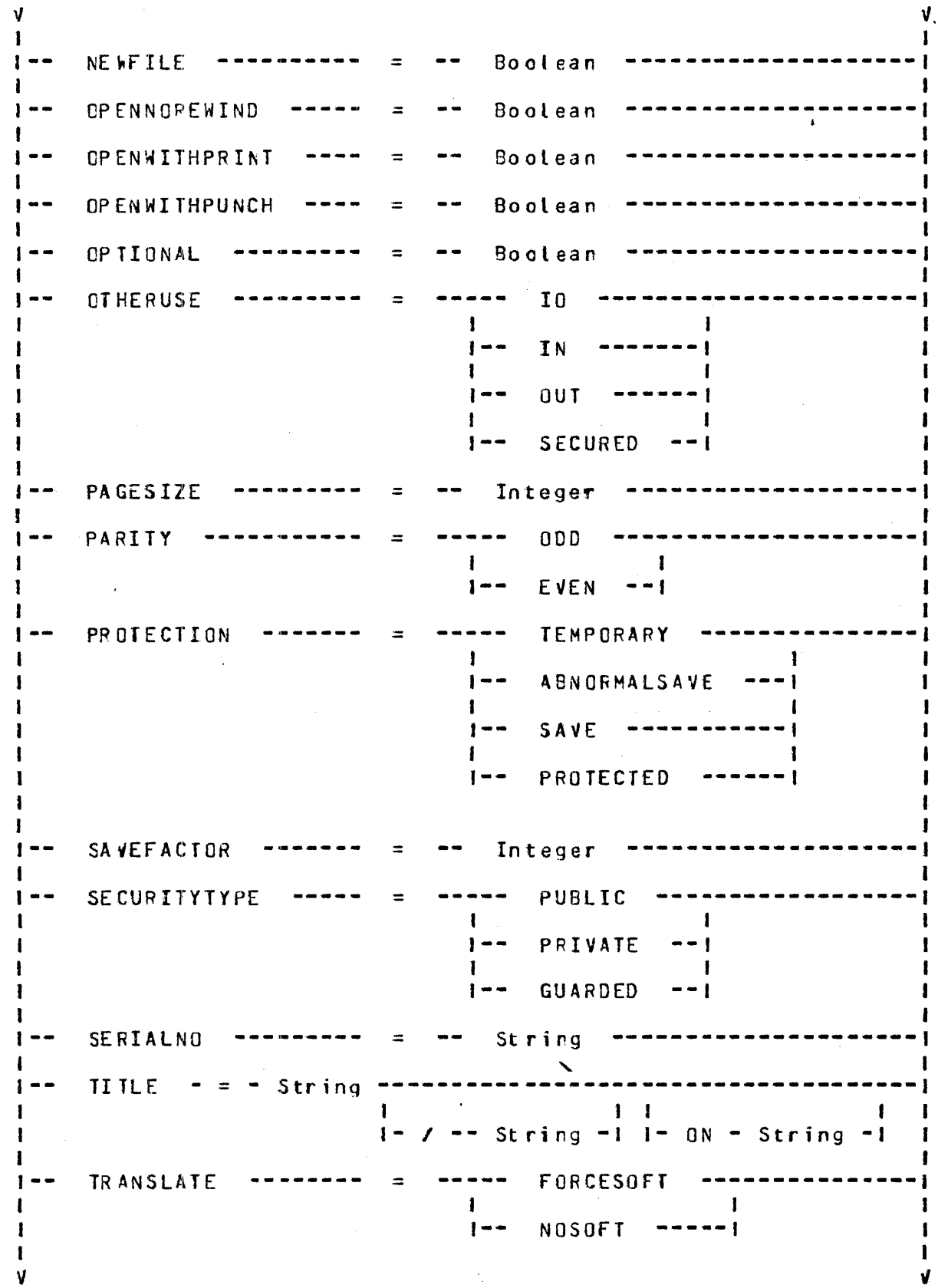
Where an attribute requires a Boolean value, the value must be TRUE or FALSE. Where an attribute requires an integer or string, the value must be a constant. All Attributes which expect mnemonic values will also allow an integer constant. However, caution should be used whenever substituting an integer for a mnemonic, since the integer values in the File Declaration statement are different from those in the Get/Put Attribute statements.

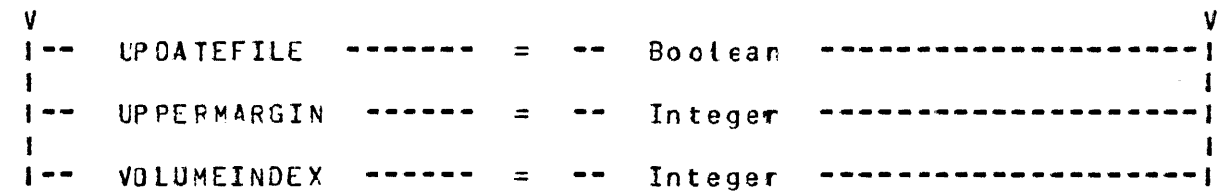



BPI6250
DEPENDENTSPECS = Boolean
DIRECTION = FORWARD
REVERSE
EXTEND = Boolean
EXTMODE = EBCDIC
ASCII
BCL
BINARY
FAMILYINDEX = Integer
FILEKIND = DATA
CODE
INTRINSIC
INTERPRETER
PSEUDO_CARD
FLEXIBLE = Boolean
FOOTING = Integer
FRAMESIZE = Integer
HOSTNAME = String
INTNAME = String


KIND
=
DATARECORDER
DISK
ODT
PAPERPUNCH
PAPERREADER
PORT
PRINTER
PUNCH
PUNCH80
PUNCH96
QUEUE
READER
READERSORTER
READER80
READER96
REMOTE
TAPE
TAPECASSETTE
TAPEPE
TAPE7
TAPE9


LABEL = EBCDICLABEL
ASCIILABEL
OMITTED
OMITTEDEOF
LINEFORMAT = Boolean
LOWERMARGIN = Integer
MAXRECSIZE = Integer
MAXSUBFILES = Integer
MINRECSIZE = Integer
MYNAME = String
MYUSE = IO
IN
OUT


NEWFILE = Boolean
OPENNOREWIND = Boolean
OPENWITHPRINT = Boolean
OPENWITHPUNCH = Boolean
OPTIONAL = Boolean
OTHERUSE = IO
IN
OUT
SECURED
PAGESIZE = Integer
PARITY = ODD
EVEN
PROTECTION = TEMPORARY
ABNORMALSAVE
SAVE
PROTECTED
SAVEFACTOR = Integer
SECURITYTYPE = PUBLIC
PRIVATE
GUARDED
SERIALNO = String
TITLE = String
/ String
ON String
TRANSLATE = FORCESOFT
NOSOFT


UPDATEFILE = Boolean
UPPERMARGIN = Integer
VOLUMEINDEX = Integer

Default File Attributes

This section lists the default values which are used by SDL2 when an attribute is not specificied in a File Declaration statement.

ACCESSMODE = SERIAL

AREALENGTH = blocksize * 100

AREAS = 25

BACKUPPERMITTED = TRUE

BLOCKSIZE = maxrecsize

BUFFERS = 1

FRAMESIZE = 8

INVALID_CHARACTERS = 2 (non-standard attribute)

KIND = DISK

MAXRECSIZE = 80 or 96 if card device,
132 if printer device,
180 in all other cases

MYUSE = IN if input only device,
OUT if output only device,
IO in all other cases

NEWFILE = TRUE if printer device,
FALSE in all other cases

TITLE = internal file name

SWITCH FILE DECLARATION

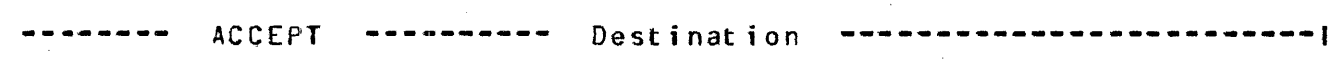
A switch file declaration allows a group of files to be selectively accessed. A switch file name with a subscript may be used anywhere that a file name is allowed, except within a switch file declaration. The files are numbered from 0 to n-1, where n is the number of files in the switch file list. If any of the files have a KIND of PORT or REMOTE, then all the files in the list must be of kind PORT or REMOTE.

Example:

```
FILE USER1 (KIND=DISK),
     USER2 (KIND=DISK),
     USER3 (KIND=DISK),

SWITCH_FILE USER_FILE(USER1, USER2, USER3);

DECLARE
   USER FIXED;
.
.
.
OPEN USER_FILE(USER);
.
.
.
WRITE USER_FILE(USER)  (BUFFER);
```

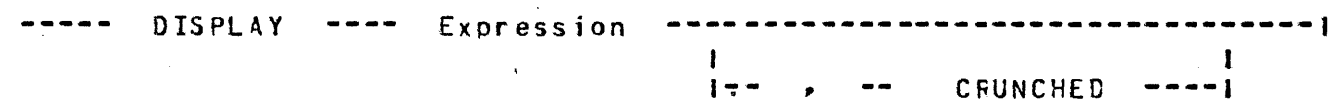
INPUT/OUTPUT STATEMENTS

Accept



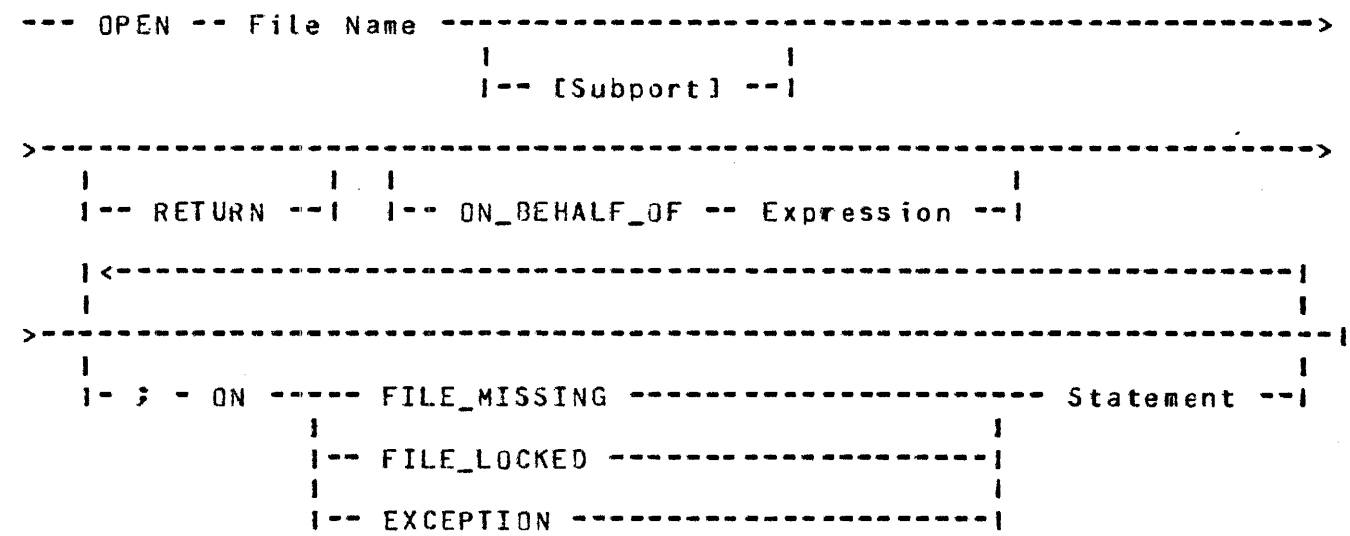
Destination must be an Address Generator. The program will be suspended until the operator responds with an AX message on the ODT. The message entered by the operator will be read into Destination, and then program execution will be resumed.

Display



Expression will be printed on the ODT. If ", CRUNCHED" is specified, the MCP will delete all trailing blanks and substitute one blank for each occurrence of multiple embedded blanks in Expression.

Open

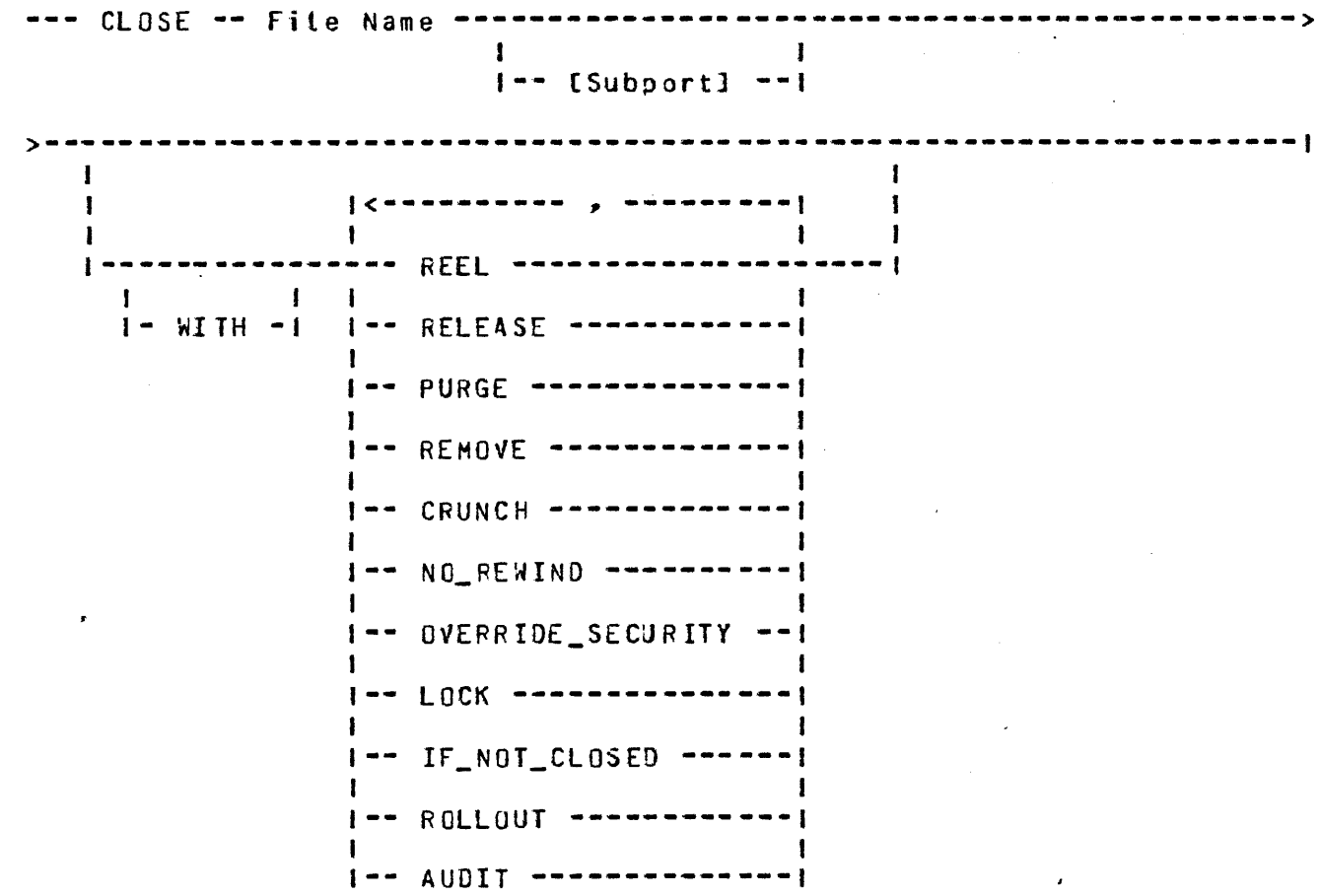


The Open statement allows the programmer to explicitly open a file, and to handle error conditions.

Example:

```
OPEN SOURCE;
  ON FILE_MISSING  SOURCE_PRESENT := FALSE;
```

Close

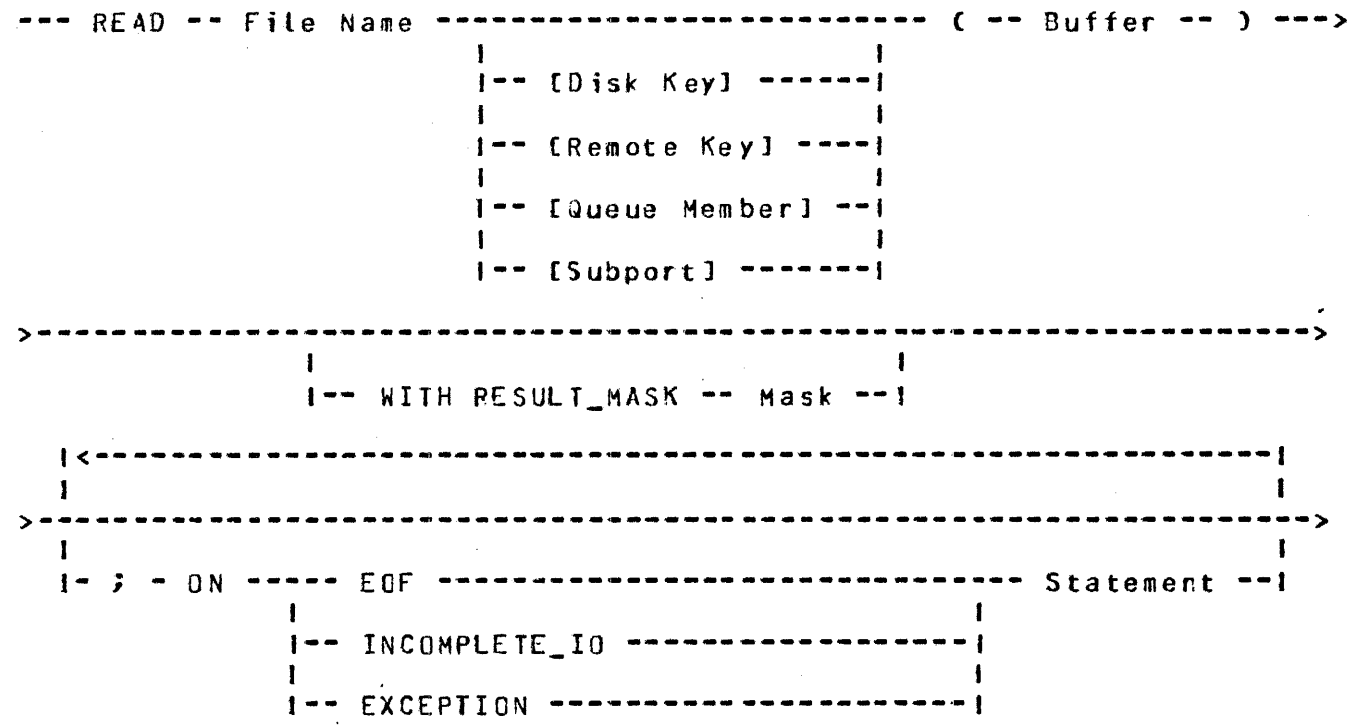


The Close statement allows a programmer to explicitly close a file and specify how it is to be disposed.

Example:

```
CLOSE SOURCE PURGE;
CLOSE NEWSOURCE LOCK, REMOVE;
```

Read

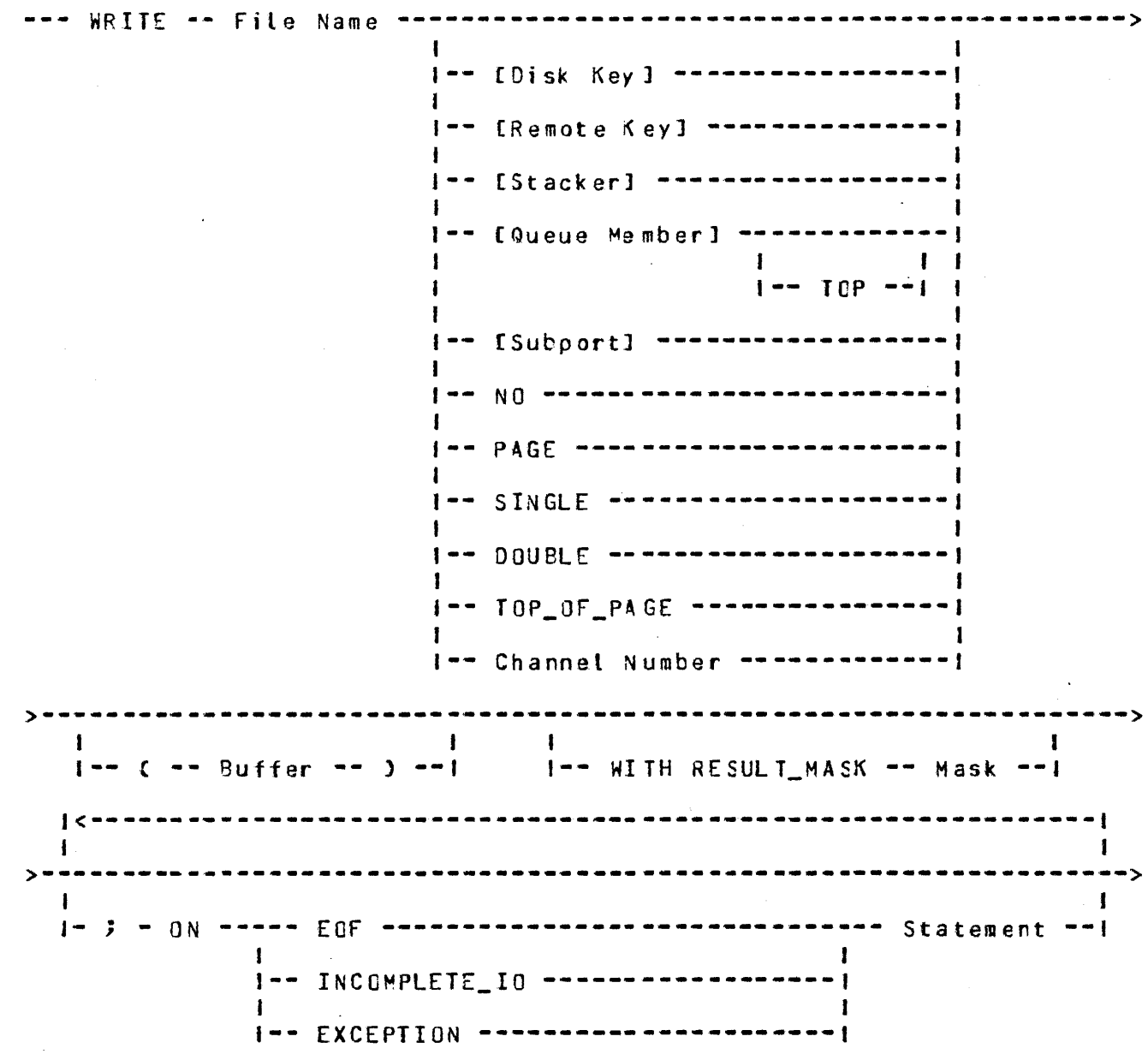


The Read statement allows a record to be read from file. Buffer must be an address generator.

Example:

```
READ RANDOM_FILE [KEY] (BUFFER);
    ON EOF BAD_KEY := TRUE;
```

Write

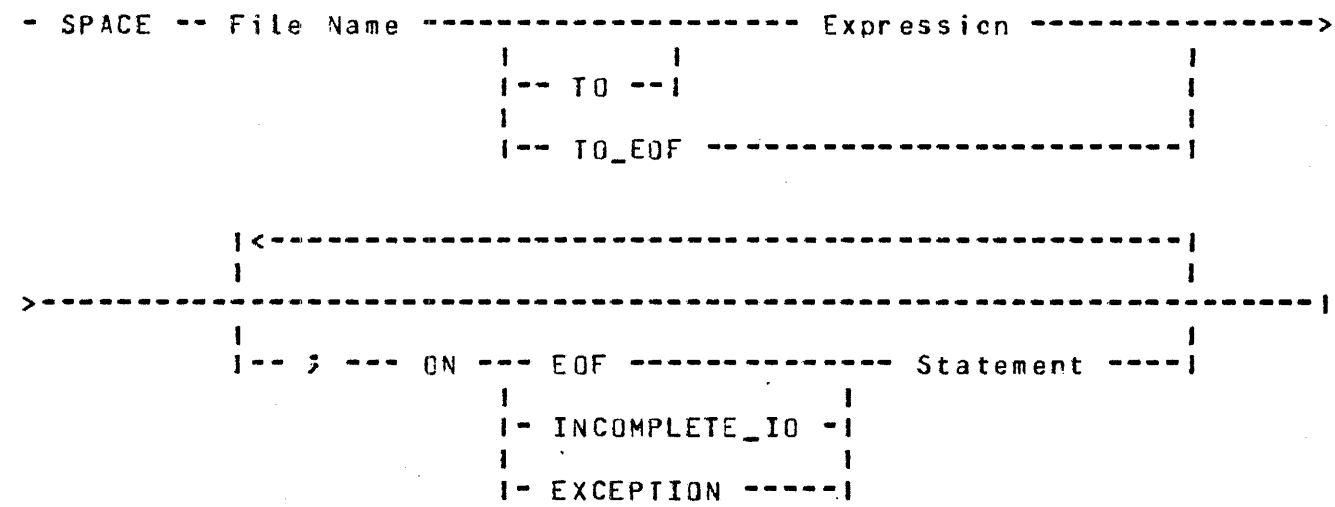


The Write statement allows a record to be written to a file, and to specify printer spacing if it is a printer file.

Example:

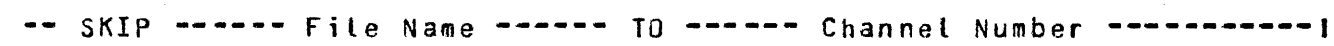
```
WRITE LINE DOUBLE ("HEADING LINE");

WRITE LINE;
```

Space



The Space statement is used to skip over records in a sequential file. If TO Expression is used it must be a positive number, and will position the file at that record. Expression alone specifies the number of records to space from the current record (it may be negative to go backwards). TO_EOF will space the file to its current end.

Skip

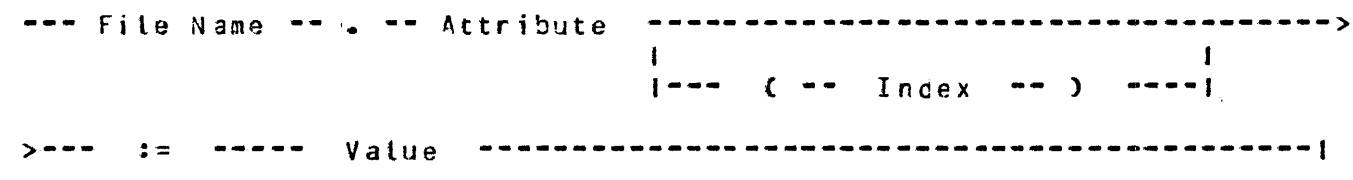


The SKIP statement will cause the line printer to skip to the specified channel number on its carriage tape. (The Channel Number can range from 1 to 12).

Seek

The Seek statement is used with random disk files to read a record into the MCP's buffer in preparation for a Read of that record. A Seek statement performed immediately before a Read Statement is less efficient than just reading the record.

Put File Attribute



The Put File Attribute statement allows an attribute of a file to be changed dynamically at run time.

Value can be either a predefined mnemonic for the attribute (as specified in the File Attributes section), or an Expression.

The following attributes are allowed. Attributes marked by an asterisk must be indexed.

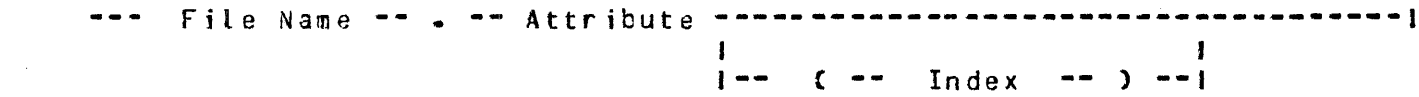
ACCESSMODE	FILENAME	OPTIONAL
AREAADDRESS *	FILESECTION	OTHERUSE
AREALENGTH	FLEXIBLE	PAGESIZE
AREAS	FOOTING	PARITY
AUDITED	FRAMESIZE	PROTECTION
BACKUPKIND	HOSTNAME	SAVEFACTOR
BACKUPPERMITTED	INTNAME	SECURITYTYPE
BLOCKSIZE	KIND	SECURITYUSE
BLOCKSTRUCTURE	LABEL	SERIALNO
BUFFERS	LASTRECORD	TITLE
COMPRESSION *	LOWERMARGIN	TRANSLATE
DATACOMPRESSION	MAXRECSIZE	UPDATE
DENSITY	MAXSUBFILES	UPDATEFILE
DEPENDENTSPECS	MINRECSIZE	UPPERMARGIN
DIRECTION	MYNAME *	VOLUMEINDEX
EXTEND	MYUSE	YOURHOSTNAME *
EXTMODE	NEWFILE	YOURNAME *
FAMILYINDEX	OPENNOREWIND	YOURUSERCODE *
FAMILYNAME	OPENWITHPRINT	
FILEKIND	OPENWITHPUNCH	

Note that the MCP expects a twenty character string with no slash for FILENAME, instead of a name of the form #fid/fid.

Example:

```
CARDS.TITLE := "SDL2_SRC/EXPRESSION ON SDL2";
```

Get File Attribute



The Get File Attribute statement will return the value of the attribute. The value returned will be either numeric or character, as specified in the 'MCP COMMUNICATES AND STRUCTURES' product specification.

The following attributes are allowed. Attributes marked with an asterisk must be indexed.

ACCESSMODE
AREAADDRESS *
AREAALLOCATED *
AREALENGTH
AREAS
ATTERR
ATTYPE
ATTVALUE
AUDITED
AVAILABLE
BACKUPFILENAME
BACKUPKIND
BACKUPPERMITTED
BLOCK
BLOCKSIZE
BLOCKSTRUCTURE
BUFFERS
CENSUS *
CHANGEDSUBFILE *
COMPRESSION *
CREATIONDATE
CREATIONTIME
CURRENTBLOCK
DATACOMPRESSION *
DENSITY
DEPENDENTSPECS
DIRECTION
DISPOSITION *
EXTEND
EXTMODE
FAMILYINDEX
FAMILYNAME
FILEKIND
FILENAME
FILESECTION
FILESTATE *
FLEXIBLE
FOOTING
FRAMESIZE
HOSTNAME
INTNAME
KIND
LABEL
LASTRECORD
LASTSUBFILE
LINEFORMAT
LINENUM
LOWERMARGIN
MAXCENSUS *
MAXRECSIZE
MAXSUBFILES *
MINRECSIZE
MYHOSTNAME *
MYNAME *
MYUSE
NEWFILE
NEXTRECORD
OPEN
OPENNOREWIND
OPENWITHPRINT
OPENWITHPUNCH
OPTIONAL
OTHERUSE
PAGESIZE
PARITY
PROTECTION
RECORD
SAVEFACTOR
SECURITYTYPE
SECURITYUSE
SERIALNO
STATE
TITLE
TRANSLATE
TRANSLATING
UPDATEFILE
UPPERMARGIN
USEDATE
VOLUMEINDEX
YOURHOSTNAME *
YOURNAME *
YOURUSECODE *

Example:

```
WRITE LINE ( "SOURCE FILE = " CAT SOURCE.TITLE );
```

SEPARATE COMPILATION

When a program becomes large it may be desirable to break the program up into several separate units, each of which may be independently compiled. These separately compilec code files can later be bound together, and calls across unit boundaries resolved.

A simple method, both for the user and compiler, is available in SDL2. A GLOBAL unit containing all global data declarations and code must be compiled (without syntax errors) prior to compilation of any of the separate units. The symbol table for the declarations (excluding procedures) is saved in the partial code file. When a separate unit is compiled the symbol table for the global unit is loaded into the compiler, making accessible the global declarations of the program.

The Procedure Declaration has been extended to allow EXTERNAL procedures to be declared. The calls on these, and the parameter checking, will be resolved by the binder.

Separately compiled units may not declare any global variables, but may declare global (to that unit only) Defines, Sets and Records.

There is no intention to hide data between units, only to make possible a faster compilation turn around. All global data is available to any unit and all procedures are available from any unit providing the procedure has been declared EXTERNAL in the using unit.

Each unit is one page in the segment dictionary, so there is a limit of 64 units to a program. To implement this facility the following control options are used:

CREATE_GLOBAL

This is used to specify to the compiler that the Global Unit of a separately compiled program is being compiled. The resulting 'code file' is not executable, but will have the name as defined in the compile statement.

USE_GLOBAL [file_name]

This is used to specify to the compiler that an independent unit of a separately compiled program is being compiled, and to load the Global symbol table from the file defined by file_name.

Separately compiled units which do not access any global data may also be created. This is done by using the following control option:

INDEPENDENT

An independent unit can be bound into any program.

The separately compiled units are bound together with the binder. The Binder will check that each unit was compiled with the same version of the global unit, and that the declarations of External procedures match.

COMPILER CONTROL

Compiler control statements must be preceeded by a single or double dollar sign ($ or $$), beginning in column 1 of the source image. If a double '$$' is used, then the control statement is considered to be permanent, and will be included in the new source file (if one is requested). If a single '$' is used, then the control statement is considered to be temporary, and will not be included in the new source file.

[NO] AUTO_SEGMENT (Default = TRUE)

The compiler will automatically break the program into segments. The segment size is determined by the 'AUTO_SEGMENT_SIZE' control card.

Segment breaks will occur at the end of a procedure when the code size exceeds the size specified in 'AUTO_SEGMENT_SIZE'.

AUTO_SEGMENT_SIZE (Default = 10000)

Specifies the segment sizes in Bits for automatic segmentation.

Example:

$ AUTO_SEGMENT_SIZE 15000

[NO] CLEAR_LOCALS (Default = FALSE)

When TRUE the compiler will generate code to clear local data space to zeros, and local descriptors to NULL.

[NO] CODE (Default = FALSE)

When TRUE the compiler will list the code generated.

CREATE_GLOBAL (Default = FALSE)

Specifies to the compiler that the global unit of a separately compiled program is being compiled.

DEBUG_DUMP

Compiler Debugging

[NO] DEBUG_RECOVERY

Compiler Debugging

[NO] DEBUG_SCANNER

Compiler Debugging

[NO] DEBUG_SYMBOL_TABLE

Compiler Debugging

DUMP_SYMBOL_TABLE

Compiler Debugging

DYNAMIC_MEMORY (Default = 0)

Specifies the amount of dynamic memory in Bits.

ERRORLIMIT (Default = 100)

Specifies the maximum number of errors which will be allowed. The compile will terminate when this limit is exceeded.

[NO] ERROPLIST (Default = FALSE)

When TRUE, the compiler will print the error messages in a separate error listing file (ERRORS).

[NO] INCLNEW (Default = FALSE)

When TRUE, the source images read in from an INCLUDE file will be copied to the new source file (NEWSOURCE). The INCLNEW option takes effect only when the NEW option is specified.

INCLUDE <library file name>

Specifies that a library source is to be read in and compiled. The remainder of the source image contains

the file name. A library file may not contain an INCLUDE statement.

INDEPENDENT

Specifies that a separately compiled unit is being compiled that does not access any global data. The independent unit created can be bound into any program.

INTRINSIC

Specifies that an intrinsic is being compiled.

[NO] LIST (Default = TRUE)

Specifies if a listing is to be produced.

[NO] LISTAMPERSAND (Default = TRUE)

If TRUE, conditional compilation statements beginning with an '&' will be included in the listing. If FALSE statements beginning with an '&' will not be listed. NO LIST overrides LISTAMPERSAND.

[NO] LISTDOLLAR (Default = TRUE)

If TRUE, all temporary compiler control statements (those beginning with a single '$') will be included in the listing. If FALSE, the temporary compiler control statements will not be listed. Permanent compiler control statements (those beginning with a double '$$') are always printed, regardless of the setting of the LISTDOLLAR option. NO LIST overrides LISTDOLLAR.

LISTINCL (Default = TRUE)

If TRUE, source images from an included library file will be listed. NO LIST overrides LISTINCL.

LISTOMITTED (Default = TRUE)

If TRUE, source that is not compiled because of conditional compilation will still be listed. NO LIST overrides LISTOMITTED.

MERGE [source file name] (Default = FALSE)

Specifies that the CARD file be merged with the SOURCE file. The file name of the source file may be specified on the rest of of the source image. If it is not specified, then the external name of the SOURCE file will be used.

NEW [new source file name] (Default = FALSE)

Specifies that a new source file be created. The file name of the new source file may be specified on the rest of the source image. If it is not specified, then the external name of the NEWSOURCE file will be used.

PAGE

Causes a page eject if listing.

PROCEDURE_STACK_SIZE (Default = 20)

Specifies the maximum number of nested procedure calls in this program. Memory for the control stack and the program pointer stack will be allocated accordingly.

SEGMENT

Causes a segment break to be generated. This option is allowed only when it immediately precedes a PROCEDURE declaration.

SEGMENT_PAGE

Causes a page and segment break to be generated. This option is allowed only when it immediately precedes a PROCEDURE declaration.

[NO] SEQCHECK (Default = FALSE)

When TRUE, the compiler will verify that the sequence number of the current source image contains only numeric characters, and is greater than the sequence number of the previous source image. The sequence number may be all blanks only if the previous sequence number was also all blanks. An invalid sequence number will generate a warning, not a syntax error.

STATIC_MEMORY

Specifies the amount of static memory to be allocated.

[NO] SUMMARY (Default = FALSE)

When TRUE, the compiler will produce a summary of the compilation process on the output listing. This summary is always produced if the LIST option is set, so SUMMARY takes effect only when NO LIST is specified.

[NO] SYMBOLIC_DUMP (Default = TRUE)

When set, symbolic information will be included in the code file for use by the Analyzer. When reset, this information will not be included in the code file. The code file will be smaller when this option is reset, but the dumps will not be informative.

USE_GLOBAL [global file name]

Specifies to the compiler that a separate unit is being compiled. The file name of the global code unit may be specified on the rest of the source image. If no name is specified, then the external name of the GLOBAL file will be used.

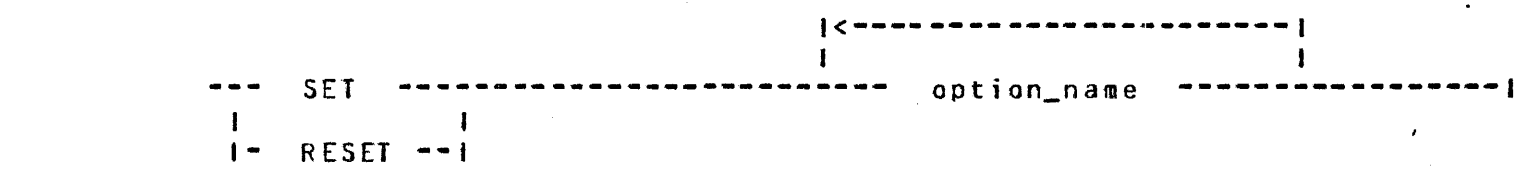
VOID <sequence number>

The VOID option may be followed by an optional Terminating Sequence Number. All subsequent records in the SOURCE file with sequence fields less than or equal to the Terminating Sequence Number will be deleted. If the Terminating Sequence Number is omitted, only the record with its sequence field equal to the sequence field of the VOID record will be deleted. The VOID option will not delete records in the CARD file.

CONDITIONAL COMPILATION

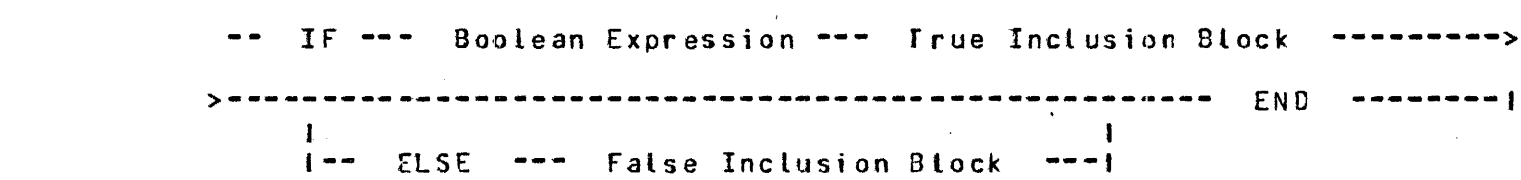
The conditional compiation facility allows the programmer to selectively compile blocks of code without the necessity of physically adding or removing records. Conditional compilation control records must begin in column 1 with an "&".

SET AND RESET STATEMENT

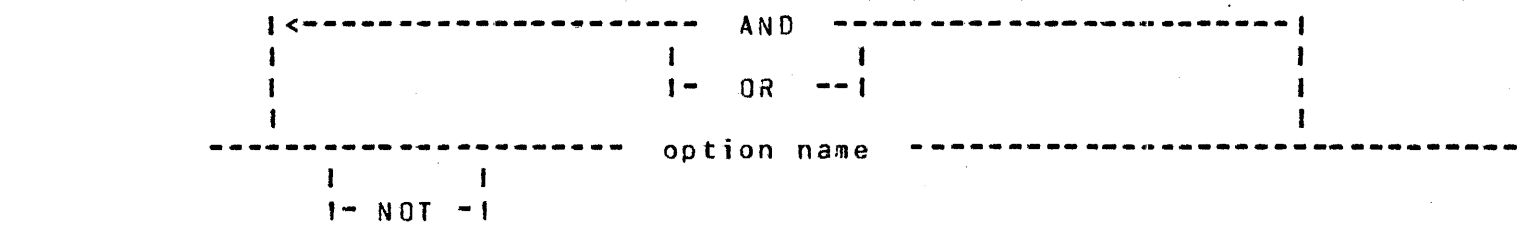


The SET and RESET statements will both declare, if not already declared, a conditional compilation option name, and SET it to true, or RESET it to false. A reference to a conditional compilation option name that has not been SET or RESET will be initialized as if it had been RESET.

IF STATEMENT



Boolean Expression



The conditional compilation IF statement is used to bracket blocks of code to be compiled conditionally on the result of the Boolean Expression.

Example:

```
&RESET  DEBUG_ALL
&SET    DEBUG_A
A := B;
&IF DEBUG_ALL OR DEBUG_A
   WRITE LINE ("A CHANGED TO" CAT DECIMAL(A,8));
&ELSE
   WRITE LINE PAGE;
&END
```

SDL2 CROSS REFERENCE

The SDL2 Cross Reference produces a symbolic cross reference from a source level SDL2 language file. The input is assumed to be compilable by the SDL2 Compiler. The output is a printer backup file.

Each entry in the cross reference contains a short description of the type of that entry. The line number where the variable was declared is also included. When doing a cross reference on a separately compiled unit, all global variables are referred to as external variables. Each entry also includes a list of the line numbers where the variable was assigned, and a separate list of where the variable was referenced.

The user may vary the width of the output file by file equating the MAXRECSIZE of the LINE file. The minimum line width is 60 characters and the maximum line width is 132 characters. This option may be used to create 80 character files, which will fit on a terminal screen.

When the cross reference program is executed, it will ask for various option settings. The user should use an AX command to set each of the options.

The user may request an alphabetical or a structured listing. The alphabetical listing places all variables in alphabetical order, except for record fields. The field identifiers are listed under their respective record identifiers. The fields are indented in the order in which they were declared. The structured listing indents all local data under each procedure. The default is to produce a structured listing.

The user may also choose to have sequence numbers printed, rather than line numbers. The default is to print line numbers.

Example:

```
HASH_INDEX IS VARIABLE OF TYPE FIXED DECLARED ON LINE 30
 ASSIGNMENTS : 50, 65, 80
 REFERENCES  : 62, 75, 84

NAME_COMPARE IS A TYPED PROCEDURE, RETURNING BOOLEAN, DECLARED ON
LINE 342
 REFERENCES  : 571, 589, 633

     FIND_ID IS A VARIABLE OF TYPE FIXED DECLARED ON LINE 331
      REFERENCES : 336, 337, 338

     FIND_TOTAL_VALUE IS A PROCEDURE DECLARED ON LINE 340
```

```
      REFERENCES  : 380

          SUB_TOTAL IS A VARIABLE OF TYPE BIT DECLARED ON LINE 342
           ASSIGNMENTS : 345,
           REFERENCES  : 347, 349

      TEMP_NAME_LENGTH IS A VARIABLE OF TYPE FIXED DECLARED OF LINE 332
       ASSIGNMENTS : 334
       REFERENCES  : 336, 339

XREF_TABLE IS A RECORD DECLARED ON LINE 99
 ASSIGNMENTS : 958, 1052
 REFERENCES  : 2164

      NAME_OFFSET IS A FIELD OF XREF_ENTRY, DECLARED ON LINE 27
       ASSIGNMENTS : 477, 512, 964
       REFERENCES  : 337, 338, 433, 434

      NAME_LENGTH IS A FIELD OF XREF_ENTRY, DECLARED ON LINE 28
       ASSIGNMENTS : 471, 507, 960
       REFERENCES  : 336, 426, 435, 440, 446
```

COMPILER EXECUTION

The primary input to the compiler is from the file CARD.

FILES

CARD	primary source input file (default device type is DISK)
SOURCE	used if 'MERGE' option is true
NEWSOURCE	new source file, if 'NEW' option is true
CODE	the generated code file
GLOBAL	code file of GLOBAL unit, if 'USE_GLOBAL' is specified
LINE	output listing
ERRORS	error message listing, if 'ERRORLIST' option is true
LIBRARY	file used to copy in library source if 'INCLUDE' is specified
FPBS	internal file for saving file parameter blocks

See 'MCP OPERATORS MANUAL' for details of MCP syntax to compile programs.

Example:

```
? CO SLD2/GLOBAL SDL2 LIBRARY;
? FI CARD NAME SDL2/SOURCE/GLOBAL;
```

BINDER EXECUTION

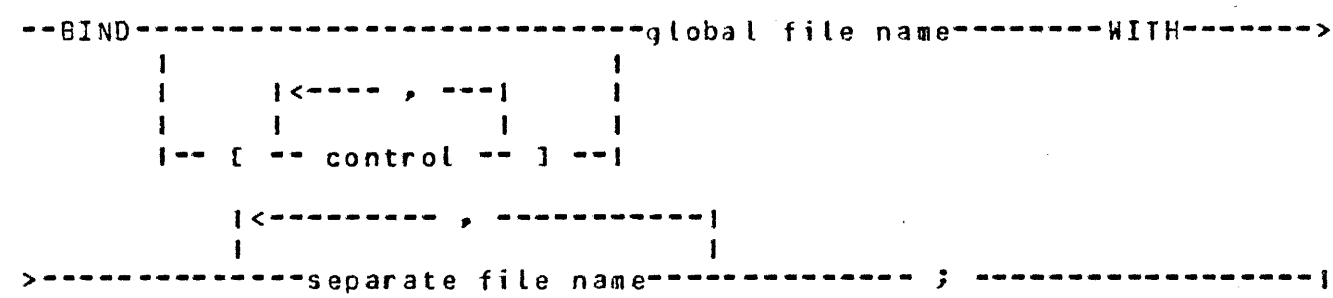
FILES

CARD	source input to the binder (default device type is DISK)
CODE	the bound code file
LINE	output listing
OBJ	the separately compiled units

See 'MCP OPERATORS MANUAL' for details of MCP syntax to compile programs.

DESCRIPTION

Syntax for the binder is:



The following is a list of legal control statements and their default values:

[NO] SYMBOLIC_DUMP	(true)
[NO] LIST	(true)

The NO SYMBOLIC_DUMP control statement removes the symbol table from the final code file. The NO LIST statement deletes the listing produced by the binder.

Example:

```
? CO SDL2 SDL2/BINDER LIBRARY;
? FI CARD CARD.READER;
? DATA CARD
BIND SDL2/GLOBAL WITH
     SDL2/DECLARE, SDL2/STATEMENT, SDL2/EXPRESSION;
? END
```

CROSS REFERENCE EXECUTION

FILES

CARD	source input (default device type is DISK)
LINE	output listing

Example:

```
?EX SDL2/XREF
?FI CARD NAME SDL2/SOURCE/GLOBAL
```

```
XREF:   SEQUENCE OR LINE NUMBERS (S/L) DEFAULT = L

     <job#>AX
```

```
XREF:   ALPHABETICAL OR STRUCTURED LISTING (A/S) DEFAULT = S

     <job#>AX
```

```
XREF:   CROSS REFERENCE OR <file name> HAS BEGUN
```

ANALYZER EXECUTION

FILES

D	dumpfile to be analyzed
LINE	output listing

Example:

```
?EX SDL2/ANALYZER;
?FI D NAME DUMPFILE/123;
```

FURTHER ANALYSIS

The SDL2/ANALYZER does not provide a hex dump, file information and buffers, or an RSN dump, since these can be obtained by running DUMP/ANALYZER with SWITCH 1=1.

Example:

```
?PM 123 SAVE;SW1 = 1;
```

Executing DUMP/ANALYZER without specifying SAVE will remove the the dumpfile. Execution of SDL2/ANALYZER will not remove the dumpfile.

APPENDIX A -- SDL TO SDL2 CONVERSION

This appendix provides information useful when converting SDL programs to SDL2. The first section describes SDL constructs which must be changed and/or avoided when converting to SDL2. The second section describes the standard routines which are available in SDL but not in SDL2. The third section describes how to convert SDL-style file attributes to the CSG-Standard file attributes used by SDL2.

DIFFERENCES BETWEEN SDL AND SDL2

The following differences between SDL and SDL2 can be resolved by direct text substitution.

1) Cosoatial record declarations must use '|' instead of "," to delimit fields.

2) The relational operators listed on the left, below, are no longer allowed. They must be replaced with the operators on the right.

/=	<>
NEQ	<>
EQL	=
GTR	>
LSS	<
GEQ	>=
LEQ	<=

3) SWAP must be replaced by READ_LOCK.

4) DUMP_FOR_ANALYSIS must be replaced by DUMP.

5) SPO_INPUT_PRESENT must be replaced by OOT_INPUT_PRESENT.

6) The vertical bar "|" cannot be used for assignments. It must be replaced by ":=".

7) SORT_SWAP must be replaced by ":=:".

8) DYNAMIC is no longer required within dynamic variable declarations, and it must be replaced by blanks.

9) Valid SDL identifier names may cause reserved-word conflicts with SDL2. New reserved words include:

CHARACTER_SET	MEMBER
CONSTANT	POINTER
CONTAINS	REPEAT
DOWNTO	SET
EXTERNAL	TYPE
FOR	UNION
IN	WHILE
INTERSECT	WITH

The next set of differences cannot be resolved by direct textual substitution. However, all occurences of these constructs in an SDL program can be easily converted to the equivalent SDL2 construct.

10) There is no UNDO (*) construct. Do-loops using this construct must be converted to named do-loops, and an undo of the outermost loop substituted for each instance of UNDO (*).

11) All formal declarations must be specified in the same order as the variables appear in the procedure heading.

12) All close options must be separated by commas.

13) While the CHAR_TABLE function is still implemented in SDL2, it no longer produces a bit table in the same order as SDL. Thus, programs which relied on the bit positions of a CHAR_TABLE string must be rewritten. For example, the following SDL code checks whether CH is a digit:

```
SUBBIT (CHAR_TABLE("0123456789"), CH, 1)
```

It must be replaced by the following SDL2 code:

```
CH IN CHAR_TABLE ("0123456789")
```

14) The GROW construct is no longer necessary, since paged arrays will automatically grow in size. The number of elements in the array may be tested by the function ARRAY_BOUND.

15) The LOCATION function, which returned a 33-bit value, must be replaced by CODE_ADDRESS, which returns a 32-bit value.

16) SDL2 treats comments and end-of-lines as separators. This means tokens cannot be split across two source images, and comments cannot be embedded within identifiers or literals. For example, the following SDL construct:

```
A := @4236/*
  */7DF0@;
```

must be replaced by the following SDL2 construct:

```
A := @4236@ CAT
     @7DF0@;
```

17) OVERLAY was a procedure in SDL, but it is a function in SDL2.

The next set of differences requires more thought than those above.

18) No descriptor manipulation is allowed. All descriptor-related constructs in SDL must be replaced by the use of reference variables, and the constructs DATA_LENGTH, DATA_ADDRESS, DATA_TYPE, REFER_ADDRESS, REFER_LENGTH, REFER_TYPE, REFER_BOUND and REFER_PAGE_SHIFT.

19) Self-relative variables will not be initialized to zero in SDL2, as they were in SDL. Instead, they must be explicitly set to zero if this is what the programmer intended. This could lead to subtle programming bugs, if any usages of non-initialized variables are not detected during the conversion process. However, if the $ CLEAR_LOCALS dollar-card option is specified, then the compiler will generate code to set all local data space to zero and all local descriptors to null.

20) File declarations must use CSG-Standard file attributes. The CHANGE, READ_FPB and WRITE_FPB statements have been replaced by direct references to standard file attributes. Refer to the FILE ATTRIBUTES section of this appendix for more information.

21) The OPEN statement does not allow any options, eg. OPEN INPUT. All open options must be set before the open request, using file attribute assignments.

22) The result of the logical operations OR, EXOR and AND is always of type bit. In SDL, the result was of the same type as the longest operand, and sc could sometimes be of type character.

23) All unstructured records in SDL had a type of bit. In SDL2, the record will be of type bit only if at least one of its subfields is not of type character. If all ot the subfields are character fields, then the record will be of type character.

24) The extended arithmetic operators X_ADD, X_SUB, X_MUL, X_DIV and X_MOD always returned bit strings of a

predictable length in SDL. In SDL2, the length of the result can vary, depending on the number of bits required to maintain precision.

The following list describes those features which will be the most difficult to change when converting from SDL to SDL2.

25) No indexing is allowed. All indexing must be replaced by the use of record or reference variables.

26) No structured declarations are allowed, and consequently no USE statements. All such declarations must be converted to records. These changes may be simplified by the use of structured records and the WITH statement.

27) SEARCH_LINKED_LIST and SEARCH_SERIAL_LIST use records instead of templates, and are formatted slightly differently than in SDL.

28) REMAPS is not allowed. It must be replaced either by cospatial record fields, or the use of reference variables.

The following differences between the two languages should also be noted.

29) The CREATE_MASTER and RECOMPILE capabilities do not exist in SDL2. They have been replaced by the compilation of separate modules, which are later bound together.

30) Segmentation is less flexible in SDL2. Segmentation may only be done at procedure boundaries, and only by requesting that a new segment and/or page be generated. Note, however, that SDL2 provides automatic segmentation, which was not available in SDL. User-specified segmentation is done through the use of the dollar cards $ SEGMENT and $ SEGMENT_PAGE. SDL Segment statements will generate an advisory message, not a syntax error. Eventually, however, specifying SEGMENT outside of a dollar-card option will be considered a syntax error. Note that SEGMENT cannot be defined as blanks, since this will work incorrectly when used in an If or Case statement.

31) There have been some changes to the dollar-card options. Any usage of dollar-card options in SDL should be checked against the allowable options in SDL2. In particular, the monitoring, profiling, and timing capabilities of SDL are not implemented in SDL2. Also, permanent dollar options must be preceded by '$$' instead of '&'. The use of an ampersand will result in an advisory message, not a syntax error. Eventually,

however, it will be considered a syntax error.

32) In conformance with the CSG Compiler Control Images Standard, the internal names of some files have been changed. The primary input file is CARD instead of CARDS, and the error message file is ERRORS instead of ERROR.LINE.

STANDARD FUNCTIONS AND PROCEDURES

UNIMPLEMENTED STANDARD ROUTINES

The following standard SDL functions and procedures are not available in SDL2.

ACCESS_FILE_INFORMATION
CHANGE_STACK_SIZES
CLEAR
CONSOLE_SWITCHES
CONTROL_STACK_BITS
CONTROL_STACK_TOP
DEBLANK
DELIMITED_TOKEN
DISPLAY_BASE
ENTER_COROUTINE
EVALUATION_STACK_TOP
EXECUTE
EXIT_COROUTINE
GROW
HASH_CODE
INTERROGATE_INTERRUPT_STATUS
INITIALIZE_VECTOR
LAST_LIO_STATUS
M_MEM_SIZE
MAKE_DESCRIPTOR
MAKE_READ_ONLY
MAKE_READ_WRITE
MONITOR
NAME_STACK_TOP
NEXT_ITEM
NEXT_TOKEN
PARITY_ADDRESS
PREVIOUS_ITEM
READ_CASSETTE
READ_FILE_HEADER
READ_FPB
READ_OVERLAY
REINSTATE
RESTORE
S_MEM_SIZE
SAVE
SAVE_STATE
SEARCH_DIRECTORY
SORT
SORT_MERGE
SORT_SEARCH
SORT_STEP_DOWN
SORT_UNBLOCK
THREAD_VECTOR
TRACE
VALUE_DESCRIPTOR
WRITE_FILE_HEADER

CONVERSION OF STANDARD ROUTINES

Search Linked List

SDL syntax:

```
SEARCH_LINKED_LIST  (START_RECORD,
                     COMPARE_FIELD,
                     COMPARE_VALUE,
                     RELATION,
                     LINK_FIELD)
```

SDL2 syntax:

```
SEARCH_LINKED_LIST  (START_RECORD,
                     COMPARE_FIELD RELATION COMPARE_VALUE,
                     LINK_FIELD)
```

In SDL, START_RECORD could be either an expression or an address generator. In SDL2, it must be an address generator with a RECORD type. COMPARE_FIELD and LINK_FIELD must be fields within that record type.

SDL returned the base-relative address of the structure containing COMPARE_VLAUE; if no match was founc it returned @FFFFFF@. SDL2 returns the structure containing COMPARE_VALUE, or an address generator with a data address of @FFFFFF@.

Example:

```
SDL:    RS_ADDR := SLL(START_ADDR, RS_JOB_NUMBER [0],
                       JN, =, RS_Q_LINK [0]);
        IF RS_ADDR = @FFFFFF@
           THEN NOT_FOUND;

SDL2:   DECLARE FIRST REFERENCE RS_NUCLEUS,
                RSN   REFERENCE RS_NUCLEUS;
        REFER_ADDRESS (FIRST, START_ADDR);
        REFER_ADDRESS (RSN, SLL(FIRST,
                                RS_JOB_NUMBER = JN,
                                RS_Q_LINK));
        IF DATA_ADDRESS (RSN) = @FFFFFF@
          THEN NOT_FOUND;
```

Search Serial List

SDL syntax:

```
SEARCH_SERIAL_LIST (COMPARE_VALUE,
                    RELATION,
                    COMPARE_FIELD,
                    START_RECORD,
                    TABLE_LENGTH,
                    RESULT_VARIABLE)
```

SDL2 syntax:

```
SERACH_SERIAL_LIST (START_RECORD,
                    COMPARE_FIELD RELATION COMPARE_VALUE,
                    TABLE_ITEMS)
```

START_RECORD must be an address generator with a RECORD type, and COMPARE_FIELD must be a field within that record type. Or, START_RECORD can be any address generator, and "COMPARE_FIELD" must be omitted; in that case START_RECORD will be used as the comparison field.

In SDL, TABLE_LENGTH specified the number of bits to be searched. In SDL2, TABLE_ITEMS must be used instead of TABLE_LENGTH. TABLE_ITEMS specifies the number of sequential entries that are to be searched.

SDL returned a 1 if the search succeeded, and a 0 if it failed, with RESULT_VARIABLE being set to the base relative address of the entry containing COMPARE_VALUE. SDL2 does not use a parameter to return the result of the search. Instead, the function returns the element number of the matching entry (the first element is 0). If no match is found, then TABLE_ITEMS will be returned.

Example:

```
SDL :  IF SSL(JN, =, UNIT_JOB_NUMBER [0],
          IOAT_REC [HINT.IOAT_POINTER],
          HINTS.IOAT_END - HINTS.IOAT_POINTER,
          IOAT_ADDR) = 0
       THEN NOT_FOUND;

SDL2: DECLARE IOAT REFERENCE IOAT_REC;
      REFER_ADDRESS (IOAT, HINTS.IOAT_POINTER);
      ENTRY := SSL (IOAT, UNIT_JOB_NUMBER=JN, IOAT_ENTRIES);
      IF ENTRY = IOAT_ENTRIES
        THEN NOT_FOUND;
        ELSE IOAT_ADDR := HINTS.IOAT_POINTER +
```

```
ENTRY * IOAT_SIZE;
```

Clear

SDL syntax:

```
CLEAR ARRAY;
```

SDL2 syntax:

```
PROCEDURE CLEAR (A);
   FORMAL A(*) BIT;
   DECLARE MAXINDEX  FIXED,
           INDEX     FIXED;
   MAXINDEX := ARRAY_BOUND (A) - 1;
   FOR INDEX := 0 TO MAXINDEX
      A(INDEX) := 0;
   RETURN;
END CLEAR;
.
.
.
.
CLEAR(ARRAY);
```

FILE ATTRIBUTES

SDL2 uses CSG-Standard file attributes in File Declaration statements, and in Get/Put Attribute statements. The CSG-Standard attributes implemented by the B1000 MCP are documented in the 'MCP COMMUNICATES AND STRUCTURES' product specification.

This section lists the attributes allowed in SDL, along with the equivalent standard attributes available in SDL2. It does not list all of the standard attributes allowed by SDL2. These are documented in the Input/Output section.

Attributes Used In File Declaration Statements

SDL	SDL2
ALL_AREAS_AT_OPEN	not implemented
AREA_BY_CYLINDER	not implemented
AREAS = areas/blks_per_area	AREAS = AREALENGTH =
BUFFERS =	BUFFERS =
DEVICE =	KIND = (in alphabetical order)
CARD	DATARECORDER
CARD_PUNCH forms backup	DISK
CARD_READER	ODT
CASSETTE	PAPERPUNCH
DATA_RECORDER_80	PAPERREADER
DISK access	PORT
DISK_FILE access	PRINTER
DISK_PACK access	PUNCH
DISK_PACK_CENTURY access	PUNCH80
DISK_PACK_CAELUS access	PUNCH96
PAPER_TAPE_PUNCH forms backup	QUEUE
PAPER_TAPE_READER	READER
PORT	READERSORTER
PRINTER forms backup	READER80
PUNCH forms backup	READER96
PUNCH_PRINTER forms backup	REMOTE
QUEUE (max msgs)	TAPE
QUEUE (max msgs) FAMILY (size)	TAPECASSETTE
READER_PUNCH_PRINTER forms backup	TAPEPE
READER_SORTER	TAPE7
READER_96	TAPE9
REMOTE (max msgs) remote_options	

```
    SORTER_READER
    SPO
    TAPE
    TAPE_NRZ
    TAPE_PE
    TAPE_7
    TAPE_9

access = null                          ACCESSMODE =
         SERIAL                             SERIAL
         RANDOM                             RANDOM
         DELAYED_RANDOM                     not implemented
         SERIAL_WITH_OVERWRITE              not implemented

forms = null | FORMS                   not implemented

backup = null                          BACKUPKIND =
         BACKUP                             DISK
         BACKUP DISK                        TAPE
         BACKUP TAPE
         NO BACKUP                     BACKUPPERMITTED =
         OR BACKUP                          DONTCARE
         OR BACKUP DISK                     DONTBACKUP
         OR BACKUP TAPE                     MUSTBACKUP

queue max msgs                         not implemented

queue family size                      not implemented

remote options = WITH HEADERS          not implemented
                 FAMILY

END_OF_PAGE_ACTION                     not implemented

EU_INCREMENTED =                       \
                                        > FAMILYINDEX =
EU_SPECIAL =                           /

EXCEPTION_MASK =                       not implemented

FILE_TYPE =                            FILEKIND =
    DATA                                    DATA
    INTERPRETER                             INTERPRETER
    CODE                                    CODE
    INTRINSIC                               INTRINSIC
    PSR_DECK                                PSEUDO_CARD

HOST_NAME =                            HOSTNAME =

INVALID_CHARACTERS =                   not implemented

LABEL = mfid/fid                       TITLE=
```

LABEL_TYPE =	
ANSI	LABEL = EBCDICLABEL or ASCIILABEL
BURROUGHS	not implemened
UNLABELED	LABEL = OMITTED or OMITTEDEOF
LOCK	not implemented
MODE =	
ODD	PARITY = ODD
EVEN	PARITY = EVEN
EBCDIC	EXTMODE = EBCDIC
ASCII	EXTMODE = ASCII
BCL	EXTMODE = BCL
BINARY	EXTMODE = BINARY
MULTI_PACK	not implemented
NUMBER_OF_STATIONS =	not implemented
OPEN_OPTION =	
INPUT	MYUSE = IN
INPUT/OUTPUT	MYUSE = IO
INTERPRET	OPENNOREWIND = TRUE
LOCK	OTHERUSE = IN
LOCK_OUT	OTHERUSE = SECURED
NEW	NEWFILE = TRUE
NO_REWIND	OPENNOREWIND = TRUE
ON_BEHALF_OF	not implemented
OUTPUT	MYUSE = OUT
PUNCH	OPENWITHPUNCH = TRUE
PRINT	OPENWITHPRINT = TRUE
REVERSE	DIRECTION = REVERSE
STACKERS	not implemented
OPTIONAL	OPTIONAL = TRUE
PACK_ID =	TITLE =
PROTECTION =	SECURITYTYPE =
PROTECTION_IO =	SECURITYUSE =
RECORDS = size/recs_per_blk	MAXRECSIZE = BLOCKSIZE =
REEL =	VOLUMEINDEX =
REMOTE_KEY	not implemented
SAVE =	SAVEFACTOR =
SECURITYTYPE =	SECURITYTYPE =

SECURITYUSE =	SECURITYUSE =
SERIAL =	SERIALNO =
TRANSLATE = fileid	TRANSLATE = FORCESOFT fileid not implemented
USE_INPUT_BLOCKING	DEPENDENTSPECS = TRUE
USER_NAMED_BACKUP	not implemented
VARIABLE	BLOCKSTRUCTURE = VARIABLE
WORK_FILE	not implemented

Attributes Used In Change (Get/Put) Statements

All attributes allowed in an SDL2 File Declaration statement are allowed in a Get/Put attribute statement. In addition, the following attributes are allowed in a Get/Put Attribute statement, but are not allowed in a File Declaration statement.

SDL	SDL2
AREA_BY_CYLINDER =	not implemented
BLOCKS_PER_AREA =	AREALENGTH =
EU_DRIVE =	FAMILYINDEX =
EU_INCREMENT =	FAMILYINDEX =
FILE_ID =	FILENAME =
MULTI_FILE_ID =	FILENAME =
NUMBER_OF_AREAS =	AREAS =
OPEN_ON_BEHALF_OF =	not implemented
QUEUE_FAMILY_SIZE =	not implemented
QUEUE_MAX_MESSAGES =	not implemented
REMOTE_HEADERS =	not implemented
RECORDS_PER_BLOCK =	BLOCKSIZE =
RECORD_SIZE =	MAXRECSIZE =

DOLLAR OPTIONS

SDL2 allows different compiler options than SDL. Most of the changes were made to conform to the CSG Compiler Control Images standard.

SDL allowed permanent compiler options to be preceded by an '&'. SDL2 uses '$$' instead. The use of an '&' will generate an advisory message, not a syntax error.

This section lists the options allowed in SDL, along with the equivalent options available in SDL2. It does not list all of the compiler options allowed by SDL2. These are documented in the Compiler Control section.

SDL	SDL2
ADVISORY	not implemented
AMPERSAND	LISTAMPERSAND
CHECK	SEQCHECK
CODE	CODE
CONTROL	LISTDOLLAR
CONVERTDOTS	not implemented
CREATE_MASTER	not implemented
CSSIZE	PROCEDURE_STACK_SIZE
DEBUG	various
DETAIL	not implemented
DOUBLE	not implemented
DYNAMICSIZE	DYNAMIC_MEMORY
ERROR_FILE	ERRORLIST
ESSIZE	STATIC_MEMORY
EXPAND_DEFINES	not implemented
FREEZE	not implemented
FORMAL_CHECK	not implemented

INTERPRETER	not implemented
INTRINSIC	INTRINSIC
LIBRARY_PACK	not implemented
LIST	LIST
LISTALL	LISTOMITTED and LIST
LOCKI	not implemented
MERGE	MERGE
MONITOR	not implemented
NEST_PROCEDURE_TIMES	not implemented
NEW	NEW
NO_DUPLICATES	not implemented
NO_SOURCE	not implemented
NSSIZE	STATIC_MEMORY
PAGE	PAGE
PASS_END	not implemented
PPSSIZE	PROCEDURE_STACK_SIZE
RECOMPILE	not implemented
RECOMPILE_TIMES	not implemented
SEQ	not implemented
SINGLE (SGL)	not implemented
SIZE	not implemented
SUPPRESS	not implemented
TIME_BLOCKS	not implemented
TIME_PROCEDURES	not implemented
TIME_MCP	not implemented
UNDERSCORES_IN_FILE_NAME	not implemented
USEDOTS	not implemented

VOID	VOID
VSSIZE	STATIC_MEMORY
WORKING_SET_BYTES	not implemented
XMAP	not implemented
XREF	not implemented
XREF_ONLY	not implemented

INDEX