SPITBOL 370 Reference Manual

14 March 1984

DISC
221 West Lake Street
Oak Park, Illinois 60302

This manual is based on the original SPITBOL 360 manual written by Robert B. K. Dewar. We thank the many people responsible for corrections and additions to the SPITBOL system and this manual. Thanks to their involvement, the current program and documentation have been greatly refined.

This manual assumes that the reader is familiar with the SNOBOL4 programming language. For those unfamiliar with SNOBOL4, appropriate texts are cited in "PART THREE--References" on page 97.

CONTENTS

PART ONE--THE SPITBOL LANGUAGE

SPITBOL 370 is an implementation of the SNOBOL4 computer language for use on the IBM 370, 43xx, 308x series computers running MVT, VS1, MVS, or CMS. SPITBOL is considerably smaller than the SIL implementation of SNOBOL4 and offers execution speeds up to ten times faster. For certain programs, notably those with in-line patterns, the gain in speed may be even greater.

Unlike SIL SNOBOL4, SPITBOL is a true compiler which generates executable machine code. The generated code may be listed in assembly form. Of course, the complexity of the SNOBOL4 language dictates that system subroutines be used for many common functions. SPITBOL can be run as an 'in-core' system like WATFIV, where jobs are executed as soon as they are compiled, and jobs may be batched together. Alternately, the compiler can generate an object module for later execution.

This manual is the documentation and user's guide for SPITBOL. It is assumed that the reader is familiar with the standard version of SIL SNOBOL4 as defined in Griswold et al "The SNOBOL4 Programming Language". Version 3 of SNOBOL4 is the reference version for comparison. There are several minor incompatibilities. In addition, there are several additions to the language in this implementation.

In general an attempt has been made to retain upward compatibility wherever possible. Most SNOBOL4 programs which operated correctly using SIL SNOBOL4 should operate correctly when compiled and executed using SPITBOL.

SPITBOL 370 is generally upward compatible with SPITBOL 360. However, both the compiler and OS interface use 370 instructions, which prevent SPITBOL 370 from running on 360 class machines. In addition, many new features have been added and a few functions changed.

FUNCTIONS

The following SPITBOL 370 functions are different from SPITBOL 360 functions.

LOAD	is now compatible with SIL SNOBOL4
SETEXIT	a new reserved label SCONTINUE is now available
SUBSTR	is NOT completely compatible with SPITBOL 360
TRIM	supports a second argument specifying the character to be trimmed

PARAMETERS

The support for imprecise interrupts on the 360/91 has been removed, and the I parameter now controls listing options. A new parameter, B, has been added for TSO attention handling. See "Running a SPITBOL Program" on page 69.

INTERFACE

The advanced operating system interface is now a standard part of the system. This allows for accessing PDS directories, PDS members, tape files, etc. See "Running a SPITBOL Program" on page 69 for more details. In addition, for a more complete understanding of the compatibility of SPITBOL 360 and SPITBOL 370, you should carefully review "Summary of Differences" on page 7.

MISCELLANY

Many bugs have been fixed. The number of 4K blocks has been increased from 32 to 64, allowing for much larger programs. There have been additions to the error codes.

This section contains a summary of the significant differences between SPITBOL and SIL SNOBOL4.

FEATURES IMPLEMENTED DIFFERENTLY

The following features are implemented in SPITBOL, but the usage is different from that in SIL SNOBOL4 and changes in existing programs may be required.

1. Recovery from execution errors (see "SETEXIT+ -- SET ERROR EXIT" on page 37).
2. I/O is somewhat different. the FORTRAN I/O routines are not used. However, a FORTRAN format processing routine has been included for compatibility.
3. The required JCL is different.

ADDITIONAL FEATURES

The following additional features (not in SIL SNOBOL4) are included in the SPITBOL system.

1. The datatype DREAL (double precision REAL).
2. The additional functions BREAKX, LEQ, LGE, LLE, LLT, LNE, LPAD, REVERSE, RTRIM, RPAD, SETEXIT, SUBSTR.
3. The additional trigonometric functions ARCCOS, ARCOSH, ARCSIN, ARCSING, ARCTAN, ARCTANH, COS, COSH, SIN, SINH, TAN, TANH.
4. The additional mathematical functions ABS, EXP, GAMMA, LOG, MAX, MIN, MOD, RANDOM.
5. The additional bit manipulation functions AND, BIT, BITSET, COMPL, OR, XOR.
6. The additional conversion functions FROMBIN, FROMDEC, FROMHEX, TOBIN, TODEC, TOHEX.
7. The additional descriptor conversion functions DCONV, ICONV, RCONV, SCONV.
8. The &FNCLEVEL keyword is writeable.
9. The additional keywords &E and &PI.
10. Additional flexibility in I/O. Support of all record formats recognized by QSAM. Format free variable record length I/O allowing simple input output of strings. Support for partitioned datasets and multi-file tape volumes.
11. Additional trace facilities for files and system events.
12. The symbolic DUMP optionally includes elements of arrays, tables and program-defined datatypes.
13. Both the pattern matching stack and the function call push down stack may expand to use all available dynamic memory if necessary.

14. A number of functions are provided to perform special I/O operations and other system activities.

OTHER INCOMPATABILITIES

1. The value of a modifiable keyword can be changed only by direct assignment using =, pattern assignment cannot be used to change a keyword value and the NAME operator cannot be applied to a keyword.

2. SPITBOL allows some datatype conversions not allowed in SIL SNOBOL4. For example, a REAL value may be used in pattern alternation and is converted to a STRING. In general, SPITBOL will convert objects to an appropriate datatype if at all possible.

3. The unary . (NAME) operator applied to a natural variable yields a NAME rather than a STRING. Since this NAME can be converted to a STRING when required, the difference is normally not noticed. the only points where the difference is apparent is in the use of the IDENT, DIFFER and DATATYPE functions and when used as a TABLE subscript.

4. SPITBOL normally operates in an optimized mode which generates a number of incompatabilities. This mode can be turned off if necessary, see description of the control statements "-OPTIMIZE -NOOPTIMIZE" on page 51.

5. SPITBOL permits leading and trailing blanks on numeric strings which are to be converted to STRING.

6. Several of the built-in functions are different. These are identified by an * next to their name. See "Functions" on page 19.

7. SPITBOL does not directly permit exponentiation of two real numbers; however, the EXP function is available.

8. The BACKSPACE function is not implemented.

9. Deferred expressions in pattern matching are not assumed to match one character in QUICKSCAN mode.

The following describes the various datatypes and conversions that are acceptable in SPITBOL.

DATATYPES IN SPITBOL

STRING Strings range in length from 0 (null STRING) to 32758 characters (subject to the setting of &MAXLNGTH). Any characters from the EBCDIC set can appear.

INTEGER INTEGERS are stored in 32 bit form allowing a range of -2**31 to +2**31-1. There is no negative zero.

REAL Stored as a 32 bit short form floating point number.

DREAL Stored using long form floating point. The low order byte is not available and is stored as zero, thus giving a 48 bit mantissa (15 decimal digits).

ARRAY ARRAYS may have up to 255 dimensions.

TABLE A table may have any number of elements, see description of -- Heading id 'srtab' unknown -- for further details. Any SPITBOL object may be used as the name of a table element, including the null string.

PATTERN PATTERN structures may range up to 32768 bytes which means there is essentially no limit on the complexity of a pattern.

NAME A NAME can be obtained from any variable. Note that in SPITBOL, The NAME operator (unary dot) applied to a natural variable yields a NAME, not a STRING as in SIL SNOBOL4.

EXPRESSION Any EXPRESSION may be deferred.

CODE A STRING representing a valid program can be converted to CODE at execution time. the resulting object, of type CODE, may be executed in the same manner as the original program.

CONVERSIONS IN SPITBOL

As far as possible, SPITBOL converts from one datatype to another as required. The following table shows which conversions are possible. A blank entry indicates that the conversion is never possible, X indicates that the conversion is always possible, and F indicates that conversion may be possible, depending on the value involved.

CONVERT TO

	S	I	R	D	A	T	P	N	E	C
S	X	F	F	F			X	X	F	F
I	X	X	X	X			X	X	X	
R	X	F	X	X			X	X	X	
D	X	F	X	X			X	X	X	
A					X	X				
T					F	X				
P							X			
N	X	F	F	F					F	F
E									X	
C										X

Where:

S is STRING
I is INTEGER
R is REAL
D is DREAL
A is ARRAY
T is TABLE
P is PATTERN
N is NAME
E is EXPRESSION
C is CODE

The following section gives detailed descriptions for each of the possible conversions.

STRING -->

1. STRING --> INTEGER

   Leading and trailing blanks are ignored. A leading sign is optional. The sign, if present, must immediately precede the digits. A null STRING or all blank STRING is converted to zero.

2. STRING --> REAL

   Leading and trailing blanks are ignored. A leading sign, if present, must immediately precede the number. The number itself may be written in standard (FORTRAN type) format with an optional exponent. The conversion is always accurate, the last bit is correctly rounded.

3. STRING --> DREAL

   The rules are the same as for STRING to REAL. Note that a STRING is considered to represent a DREAL if more than eight significant digits are given, or if a D is used for the exponent instead of an E. The conversion is always accurate, the last bit is correctly rounded.

4. STRING --> PATTERN

   A pattern is created which will match the STRING value.

5. STRING --> NAME

   The result is the NAME of the natural variable with a NAME of the given STRING. This is identical to the result of applying the unary dot operator to the variable in question. The null STRING cannot be converted to a NAME.

6. STRING --> EXPRESSION

   The STRING must represent a legal SPITBOL expression. The compiler is used to convert the STRING into its equivalent expression and the result can be used anywhere an expression is permitted.

7. STRING --> CODE

   The STRING must represent a legal SPITBOL program, complete with labels, and using semicolons to separate statements. The compiler is used to convert the STRING into executable CODE. The resulting CODE can be executed by transferring to it with a direct GOTO or by a normal transfer to a label within the CODE.

INTEGER -->

1. INTEGER --> STRING

   The result has no leading or trailing blanks. leading zeros are suppressed. A preceding minus sign is supplied for negative values. Zero is converted to '0'.

2. INTEGER --> REAL

   A REAL number is obtained by adding a zero fractional part. Note that significance is lost in converting integers whose absolute value exceeds 2**24-1.

3. INTEGER --> DREAL

   A DREAL is obtained by adding a zero fractional part. Significance is never lost in this conversion.

4. INTEGER --> PATTERN

   First convert to STRING and then treat as STRING to pattern.

5. INTEGER --> NAME

   First convert to STRING and then treat as STRING to NAME.

6. INTEGER --> EXPRESSION

   The result is a expression which when evaluated yields the integer as its value.

REAL -->

1. REAL --> STRING

   The REAL number is converted to its standard character representation. Fixed type format is used if possible, otherwise an exponent (using E) is supplied. Seven significant digits are generated, the last being correctly rounded for all cases. Trailing insignificant zeros are suppressed after rounding has taken place.

2. REAL --> INTEGER

   This conversion is only possible if the REAL is in the range permitted for integers. In this case, the result is obtained by truncating the fractional part.

3. REAL --> DREAL

   Additional low order zeros are added to extend the mantissa.

4. REAL --> PATTERN

   First convert to STRING and then treat as STRING to pattern.

5. REAL --> NAME

   First convert to STRING and then treat as STRING to NAME.

6. REAL --> EXPRESSION

   The result is an EXPRESSION which when evaluated yields the REAL as its value.

DREAL -->

1. DREAL --> STRING

   Like REAL to STRING except that 15 significant digits are given and a D is used for the exponent if one is required.

2. DREAL --> INTEGER

   This conversion is only possible if the DREAL is in the range permitted for integers. In this case, the result is obtained by truncating the fractional part.

3. DREAL --> REAL

   The low order digits of the mantissa are truncated to reduce the precision.

4. DREAL --> PATTERN

   First convert to STRING and then treat as STRING to PATTERN.

5. DREAL --> NAME

   First convert to STRING and then treat as STRING to NAME.

6. DREAL --> EXPRESSION

   The result is an EXPRESSION which when evaluated yields the DREAL as its value.

ARRAY -->

1. ARRAY --> TABLE

   The ARRAY must be two dimensional with a second dimension of two or an error occurs. For each entry (value of the first subscript), a TABLE entry using the (X,1) entry as NAME and the (X,2) entry as value is created. The TABLE built has the same number of hash headers (see the function "TABLE* -- CREATE TABLE STRUCTURE" on page 39) as the first dimension.

TABLES -->

1. TABLE --> ARRAY

   The TABLE must have at least one element which is non-null. The ARRAY generated is two dimensional. The first dimension is equal to the number of non-null entries in the TABLE. The second dimension is two. For each entry, the (X,1) element in the ARRAY is the NAME and the (X,2) element is the VALUE. The order of the elements in the array is the order in which elements were put in the table.

NAME -->

1. NAME --> STRING

   A NAME can be converted to a STRING only if it is the NAME of a natural variable. The resulting STRING is the character NAME of the variable.

2. NAME --> INTEGER,REAL,DREAL,PATTERN,EXPRESSION,CODE

   The NAME is first converted to a STRING (if possible) and then the conversion proceeds as described for STRING.

This section describes differences between the syntax in SPITBOL and SIL SNOBOL4. These differences are minor and should not affect existing programs.

1. Reference to elements of arrays which are themselves elements of arrays is possible without using the ITEM function. Thus the following are equivalent:

```
A<J><K> = B<J><K>
ITEM(A<J>,K) = ITEM(B<J>,K)
```

2. The full 80 columns of input may optionally be used. See "-IN72 -IN80" on page 51.

3. The only way to change the value of a keyword is by direct assignment. It is not permissible to use a keyword in any other context requiring a NAME.

4. The compiler permits REAL constants to be followed by a FORTRAN style exponent E+XXX or D+XXX, the latter signifies a double precision REAL (DREAL).

Pattern matching is essentially compatible with SNOBOL4, however some minor differences and extensions are described in this section.

The stack used for pattern matching can expand to fill all available dynamic memory if necessary. Thus the diagnostic issued for an infinite pattern recursion is simply the standard memory overflow message.

In QUICKSCAN mode, deferred expressions are not assumed to match one character. This is a definite incompatibility and some left recursive patterns may cause problems. However, experience seems to indicate that this heuristic has caused more problems than it has solved, so it has been abandoned.

In SPITBOL the values of &QUICKSCAN and &ANCHOR are obtained only at the start of the match. In SIL SNOBOL4, changing these values during a match can lead to unexpected results.

This section defines the functions built into the SPITBOL system. The functions are described in alphabetical order. In most cases, the arguments are preconverted to some particular datatype. This is indicated in the function header by the following notation.

FUNCTION(STRING, INTEGER, etc...)

If the corresponding argument cannot be converted to the indicated datatype, an error with major code 1 (illegal datatype) occurs. See "Error Messages and Handling" on page 53. In some cases, the range of arguments permitted is restricted. Arguments outside the permitted domain cause the generation of an error with major code 13 (incorrect value for function or operator). The usage 'ARGUMENT' implies that the argument can be of any datatype. 'NUMERIC' implies that any numeric datatype can occur (INTEGER, REAL, or DREAL).

In the following description, a single asterisk * following the name of the function indicates that the implementation of the function differs from that in SIL SNOBOL4, while a single plus sign + indicates that the function is not available in SIL SNOBOL4.

ABS+ -- COMPUTE ABSOLUTE VALUE

ABS(NUMERIC)

ABS computes the absolute value of its argument. The datatype of the returned value is the same as the argument.

AND+ -- COMPUTE LOGICAL AND

AND(STRING,STRING)

AND computes the logical "and" of its arguments by anding them together bit-by-bit. Both argument strings must be the same length.

ANY -- GENERATE PATTERN TO MATCH SELECTED CHARACTER

ANY(STRING) or ANY(EXPRESSION)

This function returns a pattern which will match a single character selected from the characters in the argument STRING. A null argument is not permitted.

If an expression argument is used, then the expression is evaluated during the pattern match and must give a non-null result.

APPLY* -- APPLY FUNCTION

APPLY(NAME,ARG,ARG,...)

The first argument is the name of a function to be applied to the (possibly null) list of arguments. Unlike SIL SNOBOL4, SPITBOL does not require the number of arguments to match. Extra arguments are ignored, and missing arguments are supplied as null strings.

ARBNO -- GENERATE PATTERN FOR ITERATED MATCH

ARBNO(PATTERN)

This function returns a pattern which will match an arbitrary number of occurrences of the pattern argument, including the null STRING (corresponding to zero occurrences).

ARCCOS+ -- COMPUTE ARCCOSINE

ARCCOS(DREAL)

ARCCOS computes the arccosine of its argument. The result is in radians and has type DREAL.

ARCCOSH+ -- COMPUTE HYPERBOLIC ARCCOSINE

ARCCOSH(DREAL)

ARCCOSH computes the hyperbolic arccosine of its argument. The result is DREAL.

ARCSIN+ -- COMPUTE ARCSINE

ARCSIN(DREAL)

ARCSIN computes the arcsine of its argument. The result is in radians and has type DREAL.

ARCSINH+ -- COMPUTE HYPERBOLIC ARCSINE

ARCSINH(DREAL)

ARCSINH computes the hyperbolic arcsine of its argument. The result is DREAL.

ARCTAN+ -- COMPUTE ARCTANGENT

ARCTAN(DREAL)

ARCTAN computes the arctangent of its argument. The result is in radians and has type DREAL.

ARCTANH+ -- COMPUTE HYPERBOLIC ARCTANGENT

ARCTANH(DREAL)

ARCTANH computes the hyperbolic arctangent of its argument. The result is DREAL.

ARG -- OBTAIN ARGUMENT NAME

ARG(NAME,INTEGER)

The first argument represents the name of the function. The integer is the formal argument number to this function. The returned result is the the selected argument STRING name. ARG fails if the integer is out of range (less than one, or greater than the number of arguments).

ARRAY -- CREATE ARRAY STRUCTURE

ARRAY(STRING, ARG)

The STRING represents the prototype of an ARRAY to be allocated. This is in the format 'LBD1:HBD1,LBD2:HBD2,..' the low bound (LBD) may be omitted for some or all of the dimensions, in which case a low bound of one is assumed. The second argument (of any datatype) is the initial value of all the elements in the array. If the second argument is omitted, the initial value of all elements will be the null STRING.

BIT+ -- EXTRACT BIT

BIT(STRING,INTEGER)

BIT extracts a bit from its argument. The second argument specifies the bit position--bit positions start with 0 as the leftmost bit in the string. The returned value is either an integer 0 or 1.

BITSET+ -- SET BIT

BITSET(STRING,INTEGER,INTEGER)

BITSET returns the result of setting a bit to 0 or 1 in the string specified by its first argument. The second argument specifies the bit position--bit positions start with 0 as the leftmost bit in the string. The third argument is the new value, 0 or 1, of the specified bit.

BREAK -- GENERATE SCANNING PATTERN

BREAK(STRING) or BREAK(EXPRESSION)

This function returns a pattern which will match any STRING up to but not including a character in the STRING argument. A null argument is not permitted.

If an expression argument is given, the resulting pattern causes the STRING to be evaluated during pattern matching. In this case, the evaluated result must be a non-null STRING.

BREAKX+ -- GENERATE EXTENDED SCANNING PATTERN

BREAKX(STRING) or BREAKX(EXPRESSION)

BREAKX returns a pattern whose initial match is the same as a corresponding BREAK pattern. However, BREAKX has implicit alternatives which are obtained by scanning past the first break character found and scanning to the next BREAK character.

BREAKX may be used to replace ARB in many situations where BREAK cannot be used easily. For example, the following replacement can be made:

```
ARB ('CAT' | 'DOG') ---> BREAKX('CD')('CAT' | 'DOG')
```

For an expression argument, the expression is evaluated during pattern matching and must yield a non-null STRING value. The evaluation of the expression is not repeated on rematch attempts by extension.

Note: BREAKX(S) = BREAK(S) ARBNO(LEN(1) BREAK(S))

CLEAR* -- CLEAR VARIABLE STORAGE

CLEAR(STRING, ARGUMENT)

This function causes the values of variables to be set to null. In the simple case, where both arguments are omitted, the action is the same as in SIL SNOBOL4. For example, all variables are cleared to contain null. Two extensions are available in SPITBOL. The first argument may be a STRING which is a list of variable names separated by commas. These represent the names of variables whose value is to be left unchanged. In addition, if a second non-null argument is supplied, then all variables containing pattern values are left unchanged. For example:

```
CLEAR('ABC,CDE,GGG',1)
```

Would cause the value of all variables to be cleared to null except for the variables ABC,CDE,GGG and all other variables containing pattern values.

CODE -- COMPILE CODE

CODE(STRING)

The effect of this function is to convert the argument to type CODE as described in the section on type conversion. The STRING must represent a valid SPITBOL program complete with labels and using a semicolon (;) to separate statements or the call to CODE fails.

COLLECT -- INITIATE STORAGE REGENERATION

COLLECT(INTEGER)

The COLLECT function forces a garbage collection which retrieves unused storage and returns it to the block of available storage. The integer argument represents a minimum number of bytes to be made available. If this amount of storage cannot be obtained, the COLLECT function fails. On successful return, the result is the number of bytes actually obtained.

Note that although the implementation of COLLECT is similar to that in SIL SNOBOL4, the values obtained will be quite different due to different internal data representations. Furthermore, the internal organization of SPITBOL is such that forcing garbage collections to occur before they are required always increases execution time.

COMPL+ -- COMPUTE LOGICAL COMPLEMENT

COMPL(STRING)

COMPL computes the logical "complement" of its argument by complementing each bit in its argument.

CONVERT* -- CONVERT DATATYPES

CONVERT(ARGUMENT,STRING)

The returned result is obtained by converting the first argument to the type indicated by the STRING name of the datatype given as the second argument. The section on type conversion describes the permitted conversions. Any conversions which are not permitted cause failure of the convert call.

CONVERT allows conversions of all objects to STRING as in SIL SNOBOL4.

An additional possibility for the second argument is 'NUMERIC', in which case, the argument is converted to INTEGER, REAL or DREAL according to its form.

COPY* -- COPY STRUCTURE

COPY(ARGUMENT)

The COPY function returns a distinct copy of the object which is its argument. This is only useful for arrays, tables, and program-defined datatypes. Note that, unlike SIL SNOBOL4, SPITBOL does permit the copying of tables.

COS+ -- COMPUTE COSINE

COS(DREAL)

COS computes the cosine of its argument. The argument is in radians and the result is DREAL.

COSH+ -- COMPUTE HYPERBOLIC COSINE

COSH(DREAL)

COSH computes the hyperbolic cosine of its argument. The result is DREAL.

DATA -- CREATE DATATYPE

DATA(STRING)

The argument to DATA is a prototype for a new datatype in the form of a function call with arguments. The function name is the name of the new datatype. The 'argument' names are names of functions which represent the fields of the new datatype.

Note: In SPITBOL, a significant increase in efficiency is obtained by avoiding the use of duplicate field names for different datatypes, although SPITBOL does allow such multiple use of field function names.

DATATYPE* -- OBTAIN DATATYPE

DATATYPE(ARGUMENT)

DATATYPE returns the formal identification of the datatype of its argument. In SPITBOL, the additional datatype name 'DREAL' is included in the list of possible returned results.

DATE -- OBTAIN DATE

DATE()

DATE returns an eight character STRING of the form MM/DD/YY representing the current date. See "SYSDATE" on page 91 for alternative date string formats.

DCONV+ -- DESCRIPTOR CONVERSION OF STRING TO DREAL

DCONV(STRING)

DCONV makes a copy of its argument and changes the datatype to DREAL. The argument string must have exactly eight characters. During conversion the rightmost character (low order eight bits) will be truncated.

```
DCONV( FROMHEX( '4110000000000000' ) ) = 1.
```

DEFINE -- DEFINE A FUNCTION

DEFINE(STRING) or DEFINE(STRING,NAME)

The DEFINE function is used to define program-defined functions.

DETACH -- DETACH I/O ASSOCIATION

DETACH(NAME)

NAME is the name of a variable which has previously been input or output associated. Use of the DETACH function does not affect the file involved.

DIFFER* -- TEST FOR ARGUMENTS DIFFERING

DIFFER(ARGUMENT,ARGUMENT)

DIFFER is a predicate function which fails if its two arguments are identical objects. Note that DIFFER(.ABC,'ABC') succeeds in SPITBOL since .ABC is a NAME. DIFFER and IDENT are the only functions in which the different implementation of the NAME operator (unary dot) may give rise to problems.

DUMP* -- DUMP STORAGE

DUMP(INTEGER)

The DUMP function causes a dump of current values. After the dump is complete, execution continues unaffected (the DUMP function returns the null STRING). If the argument to DUMP is one, then the dump includes values of all non-constant keywords and all non-null natural variables. If the argument to DUMP is two, then the dump includes values of all array and table elements, and of field values of all program-defined datatypes. The format of the latter dump is self explanatory and avoids printing any structure more than once. DUMP(3) causes a formatted memory dump to be printed.

A call to DUMP with a zero argument is ignored. This allows use of a switch value which can be turned on and off globally.

DUPL -- DUPLICATE STRING

DUPL(STRING, INTEGER)

DUPL returns a STRING obtained by duplicating the first (STRING) argument the number of times indicated by the second argument.

ENDFILE* -- CLOSE FILE

ENDFILE(STRING)

String is the name of a file (not the name of a variable associated with the file). The named file is closed, all associated storage is released and all variables associated with the file are automatically detached. Thus ENDFILE should be used only when no further use is to be made of the file. If the file is to be reread or rewritten, REWIND should be used rather than ENDFILE.

EQ -- TEST FOR EQUAL

EQ(NUMERIC,NUMERIC)

EQ is a predicate function which tests whether its two arguments are equal. DREAL arguments are permitted.

EVAL -- EVALUATE EXPRESSION

EVAL(EXPRESSION)

EVAL returns the result of evaluating its expression argument. Note that a STRING can be converted into an expression by compiling it into code. Thus EVAL in SPITBOL is compatible with SIL SNOBOL4 and handles strings in the same way.

EXP+ -- COMPUTE EXPONENTIAL

EXP(DREAL,DREAL)

EXP computes the exponential of its first argument raised to the power of its second argument. The result is DREAL.

Note: If the arguments are both integers, EXP will compute a more precise result in some cases.

FIELD -- GET FIELD NAME

FIELD(NAME, INTEGER)

FIELD returns the name of the selected field of the program-defined datatype whose name is the first argument. If the second argument is out of range (less than one, or greater than the number of fields), the FIELD function fails.

FROMBIN+ -- CONVERT BINARY STRING TO CHARACTER STRING

FROMBIN(STRING)

FROMBIN takes a string of binary digits and converts it to a string of EBCDIC characters. For example,

```
FROMBIN( '1100100011001001' ) = 'HI'
```

FROMDEC+ -- CONVERT PACKED DECIMAL STRING TO INTEGER

FROMDEC(STRING)

FROMDEC takes a string of packed decimal digits and converts it to an integer. For example,

```
FROMDEC( FROMHEX( '123F' ) ) = 123
```

FROMHEX+ -- CONVERT HEX STRING TO CHARACTER STRING

FROMHEX(STRING)

FROMHEX takes a string of hexadecimal digits and converts it to a string of EBCDIC characters. For example,

FROMHEX('C8C9') = 'HI'

GAMMA+ -- COMPUTE GAMMA

GAMMA(DREAL)

GAMMA computes the gamma function of its argument. The result is DREAL.

Note: GAMMA can be used to compute factorials, because of the identity

GAMMA(X) = (X-1)!

GE -- TEST FOR GREATER OR EQUAL

GE(NUMERIC,NUMERIC)

GE is a predicate function which tests if the first argument is greater than or equal to the second argument.

GT -- TEST FOR GREATER

GT(NUMERIC,NUMERIC)

GT is a predicate function which tests if the first argument is greater than the second argument.

ICONV+ -- DESCRIPTOR CONVERSION TO INTEGER

ICONV(REAL) or ICONV(STRING)

ICONV makes a copy of its argument and changes the datatype to integer. If the argument is REAL, the 32 bit real value is returned as an INTEGER

If the argument is a STRING, there are two possible cases which affect the conversion process: (1) the string is fewer than four characters or (2) the string is four or more characters in length. If the string is fewer than four characters, it is padded on the left with leading binary zeros to four bytes and returned as an INTEGER. If the string is four or more characters, the first four characters are copied and returned as an INTEGER. All other characters are truncated on the right. For example:

```
ICONV('A') = 193
ICONV('HI') = 51401
ICONV(1.0) = 1091567616
```

IDENT* -- TEST FOR IDENTICAL

IDENT(ARGUMENT,ARGUMENT)

IDENT is a predicate function which tests if its two arguments are identical. Note that in SPITBOL, IDENT (.ABC,'ABC') fails since .ABC is a NAME in SPITBOL. Otherwise IDENT is compatible.

INPUT* -- SET INPUT ASSOCIATION

INPUT(NAME,STRING,INTEGER)

The first argument is the name of a variable which is to be input associated. The second argument is the filename of the file to which the variable is to be associated. In OS, the name corresponds to the OS DDNAME of the file or to a pseudo-DDNAME that is specially interpreted by SPITBOL's operating system interface. For more details on the various forms and meanings of DDNAMES, see "Input/Output Facilities" on page 77. If the second argument is omitted, the filename 'SYSIN' (standard input file) is assumed. For compatibility with SIL SNOBOL4, the second argument may be a one or two digit integer, in which case, the DDNAME FTXXF001 is used. note however, that the filename 5 is interpreted as SYSIN if no FT05F001 DD statement is supplied. Also, there is no provision for multiple files in the FORTRAN sense. Dataset concatenation can be used instead.

The third argument is either zero, in which case it is ignored, or a positive non-zero integer, in which case input records longer than the given limit are truncated.

A restriction in SPITBOL is that only natural variables can be input associated. It is not possible to input associate ARRAY and TABLE elements.

INTEGER* -- TEST FOR INTEGRAL

INTEGER(NUMERIC)

INTEGER is a predicate function which tests whether its argument is integral. It fails if the argument cannot be converted to numeric, or if it has a non-integral value.

ITEM -- SELECT ARRAY OR TABLE ELEMENT

ITEM (ARRAY, INTEGER, INTEGER,...) or

ITEM (TABLE,ARGUMENT)

ITEM returns the selected ARRAY or TABLE element by NAME. Note that the use of ITEM is unnecessary in SPITBOL because of the extended syntax for array references. (see " Syntax" on page 15).

LE -- TEST FOR LESS THAN OR EQUAL

LE(NUMERIC,NUMERIC)

LE is a predicate function which tests whether the first argument is less than or equal to the second argument.

LEN -- GENERATE SPECIFIED LENGTH PATTERN

LEN(INTEGER) or LEN(EXPRESSION)

LEN generates a pattern which will match any sequence of characters of the length given by the argument which must be a non-negative integer greater than zero.

If the argument is an expression, it is evaluated during pattern matching and must yield a non-negative integer.

LEQ+ -- TEST FOR LEXICALLY EQUAL

LEQ(STRING,STRING)

LEQ is predicate function which tests whether its arguments are lexically equal. Note that LEQ differs from the IDENT function in that its arguments must be strings, thus LEQ(10,'10') succeeds as does LEQ(.ABC,'ABC').

LGE+ -- TEST FOR LEXICALLY GREATER OR EQUAL

LGE(STRING,STRING)

LGE is a predicate function which tests whether the first argument is lexically greater than or equal to the second argument.

LGT -- TEST FOR LEXICALLY GREATER

LGT(STRING,STRING)

LGT is a predicate function which tests whether its first STRING argument is lexically greater than the second STRING argument.

LLE+ -- TEST FOR LEXICALLY LESS OR EQUAL

LLE(STRING,STRING)

LLE is a predicate function which tests whether its first STRING argument is lexically less than or equal to the second argument.

LLT+ -- TEST FOR LEXICALLY LESS

LLT(STRING,STRING)

LLT is a predicate function which tests whether its first argument is lexically less than its second argument.

LNE+ -- TEST FOR LEXICALLY NOT EQUAL

LNE(STRING,STRING)

LNE is a predicate function which tests whether its arguments are lexically unequal. LNE differs from the DIFFER function in that its arguments must be strings.

LOAD* -- LOAD EXTERNAL FUNCTION

LOAD(STRING,STRING)

LOAD is used to load an external function. The form of the first argument is the same as in SIL SNOBOL4 except that the datatype DREAL may be used. In the case where the datatype is unspecified, the form of the descriptor passed is quite different from that in SIL SNOBOL4. The form of converted arguments is identical.

The second argument specifies the DDNAME of a load library containing the external function's load module. If the second argument is omitted, the standard OS job and step libraries are searched.

See "External Functions" on page 89 for details on external functions.

LOC -- GET NAME OF LOCAL

LOC(NAME,INTEGER)

The value returned is the name of the indicated local of the function named by the first argument. LOC fails if the second argument is out of range (less than one, or greater than the number of locals).

LOG+ -- COMPUTE LOGARITHM

LOG(DREAL,DREAL)

LOG computes the logarithm of its first argument to the base specified by its second argument. If the second argument is null or zero, the natural logarithm is computed (base &E).

Logarithms to base 2, base &E (natural logarithm), and base 10 are handled specially by LOG. Logarithms to other bases are computed using the identity:

LOG(A,B) = LOG (A) / LOG (B).

LPAD+ -- LEFT PAD STRING

LPAD(STRING,INTEGER,STRING)

LPAD returns the result obtained by padding out the first argument on the left to the length specified by the second argument, using the pad character supplied by the one character STRING third argument. If the third argument is null or omitted, a blank is used as the pad character. If the first argument is already long enough or too long, it is returned unchanged. LPAD is useful for constructing columnar output.

LT -- TEST FOR LESS THAN

LT(NUMERIC,NUMERIC)

LT is a predicate function which tests whether the first argument is less than the second argument.

MAX+ -- COMPUTE MAXIMUM VALUE

MAX(NUMERIC,...,NUMERIC)

MAX returns the maximum value of its arguments. A maximum of 63 arguments can be specified. The datatype used for comparisons and the returned value is derived from the datatype of the highest precision argument according to the hierarchy DREAL, REAL, INTEGER.

MIN+ -- COMPUTE MINIMUM VALUE

MIN(NUMERIC,...,NUMERIC)

MIN returns the minimum value of its arguments. A maximum of 63 arguments can be specified. The datatype used for comparisons and the returned value is derived from the datatype of the highest precision argument according to the hierarchy DREAL, REAL, INTEGER.

MOD+ -- COMPUTE REMAINDER FOR DREALS

MOD(DREAL,DREAL)

MOD computes the modulus--the first argument modulus the second argument. The result is DREAL.

NOTANY -- GENERATE CHARACTER SELECT PATTERN

NOTANY(STRING) or NOTANY(EXPRESSION)

NOTANY returns a pattern which will match any single character not in the STRING argument given. A null argument is not permitted.

If the argument is an expression, then the expression is evaluated at pattern match time and must yield a non-null STRING.

OPSYN* -- EQUATE FUNCTIONS

OPSYN(NAME,NAME,INTEGER)

The first argument is the name of a function defined to have the same definition as the function named in the second argument. OPSYN may be used to redefine operators using a third argument of 1 or 2 as in SIL SNOBOL4 with the following restrictions:

A. Only the first argument can be an operator name.

B. Only normally undefined operators can be redefined.

OR+ -- COMPUTE LOGICAL OR

OR(STRING,STRING)

OR computes the logical "or" of its arguments by or-ing them together bit-by-bit. Both argument strings must have the same length.

OUTPUT* -- SET OUTPUT ASSOCIATION

OUTPUT(NAME,STRING,STRING)

The first argument is the name of a variable to be ouput associated. The second argument is the name of the file to which the association is to be made. In OS, this name corresponds to the OS DDNAME of the file or to a pseudo-DDNAME that is interpreted by SPITBOL's operating system interface. For more details on the various forms and meanings of DDNAMEs see "Input/Output Facilities" on page 77. If the second argument is omitted, the filename 'SYSPRINT' (standard output print file) is assumed. For compatibility with SIL SNOBOL4, the filename may be a one or two digit integer, in which case the DDNAME FTXXF001 is used. Note however, that the filenames 6, 7 are interpreted as SYSPRINT and SYSPUNCH if the corresponding FTXXF001 DD statements are not supplied.

The third argument is the format. It may be entirely omitted. In this case, all parameters are taken from the dataset definition. Strings are transmitted directly. If a STRING exceeds the specified length (maximum record length for variable length records), then it is split into segments as required.

The second possibility for a format is a single character. This is used for print files. The character given is a control character which is appended to the start of each record. Thus the definition of the standard print file is:

```
OUTPUT(.OUTPUT,,' ')
```

A third possibility for the format is a FORTRAN format. This is supplied for compatibility with SIL SNOBOL4 and should not be used except where required since format processing is inherently time consuming.

A restriction on the output function in SPITBOL is that only natural variables may be associated. It is not possible to output associate array and table elements.

POS -- GENERATE POSITIONING PATTERN

POS(INTEGER) or POS(EXPRESSION)

POS returns a pattern which matches the null STRING after the indicated number of characters has been matched. The argument must be a non-negative integer.

If an expression argument is given it is evaluated during pattern matching and must yield a non-negative integer.

PROTOTYPE -- RETRIEVE PROTOTYPE

PROTOTYPE(ARRAY) or PROTOTYPE(TABLE)

PROTOTYPE returns the first argument used in the array or table function call which created the argument.

RANDOM+ -- COMPUTE RANDOM NUMBER

RANDOM(INTEGER,INTEGER,INTEGER)

RANDOM returns a random number generated by the linear congruential method. The first two arguments specify respectively the lower and upper bounds of the range from which the number is chosen. The third argument is the seed value, which can be changed at any time.

If the second argument is null, the upper bound is set to the maximum positive integer. If the third argument is null, the seed is defaulted to 31.

A call to RANDOM with no arguments, RANDOM (), will return a positive random integer.

RCONV+ -- DESCRIPTOR CONVERSION TO INTEGER

RCONV(INTEGER) or RCONV(STRING)

RCONV makes a copy of its argument and changes the datatype to REAL. If the argument is INTEGER, the 32 bit integer value is returned as REAL. If the argument is a STRING, it must have exactly four characters.

RCONV(0) = 0.

REMDR -- COMPUTE REMAINDER FOR INTEGERS

REMDR(INTEGER,INTEGER)

REMDR returns the remainder of dividing the first argument by the second, the remainder has the same sign as the first argument.

REPLACE -- TRANSLATE CHARACTERS

REPLACE(STRING,STRING,STRING)

REPLACE returns the result of applying the transformation represented by the second and third arguments to the first argument. REPLACE fails if the second and third arguments are unequal in length or null.

REVERSE+ -- REVERSE STRING

REVERSE(STRING)

REVERSE returns the result of reversing its STRING argument. Thus REVERSE('ABC') = 'CBA'.

REWIND -- REPOSITION FILE

REWIND(STRING)

String is the name of an external file (not the name of a variable associated with the file). The named file is repositioned so that the next read or write operation starts at the first record of the file. Existing associations to the file are unaffected.

RPAD+ -- RIGHT PAD STRING

RPAD(STRING,INTEGER,STRING)

RPAD is similar to LPAD except that the padding is done on the right.

RPOS -- GENERATE POSITIONING PATTERN

RPOS(INTEGER) or RPOS(EXPRESSION)

RPOS creates a pattern which will match null when the indicated number of characters remain to be matched. The integer argument must be non-negative.

If an expression argument is used, it is evaluated during the pattern match and must yield a non-negative integer.

RTAB -- GENERATE TABBING PATTERN

RTAB(INTEGER) or RTAB(EXPRESSION)

RTAB returns a pattern which matches from the current location to the point where the indicated number of characters remain to be matched. The argument must be a non-negative integer. If an expression is used, it is evaluated during pattern matching and must yield a non-negative integer.

RTRIM+ -- REVERSE TRIM

RTRIM(STRING,STRING)

RTRIM returns the result of trimming leading characters from its first argument. Argument 2 specifies the character to be trimmed. If argument 2 is omitted or null, a blank is assumed.

Note: RTRIM(S,C) = REVERSE(TRIM(REVERSE(S),C))

SCONV+ -- DESCRIPTOR CONVERSION TO STRING

SCONV(NUMERIC)

SCONV makes a copy of its argument and changes the datatype to a STRING. If the argument is an INTEGER or REAL, the resulting STRING is four characters long. If the argument is DREAL, the resulting string is seven characters long. If the argument is STRING, it is coerced to numeric before conversion.

```
TOHEX(SCONV(1)) = '00000001'
TOHEX(SCONV(4095)) = '00000FFF'
```

SETEXIT+ -- SET ERROR EXIT

SETEXIT(NAME) or SETEXIT()

The use of SETEXIT allows interception of any execution error. The argument to SETEXIT is a label to which control is passed if a subsequent error occurs, providing that the value of the keyword &ERRLIMIT is non-zero. The value of &ERRLIMIT is decremented when the error trap occurs. The SETEXIT call with a null argument causes cancellation of the intercept. A Subsequent error will terminate execution as usual with an error message.

The result returned by SETEXIT is the previous intercept setting (i.e., a label name or null if no intercept is set). This can be used to save and restore the SETEXIT conditions in a recursive environment.

The error intercept routine may inspect the error code stored in the keyword &ERRTYPE (see "Keywords" on page 43) and take one of the following actions:

1. Terminate execution by transferring to the special label ABORT. This causes error processing to resume as though no error intercept had been set.

2. Branching to the special label CONTINUE. This causes execution to resume by branching to the failure exit of the statement in error.

3. Branching to the special label SCONTINUE. This causes execution to resume at the point of error, by branching into the statement.

4. Continue execution elsewhere by branching to some section of the program. Note that if the error occurred inside a function, we are still 'down a level.'

The occurrence of an error cancels the error intercept. Thus the error intercept routine must reissue the SETEXIT if required.

Note: When a SETEXIT routine is entered due to exceeding &STLIMIT, the program is given about 5 additional statements in which &STLIMIT can be increased.

SIN+ -- COMPUTE SINE

SIN(DREAL)

SIN computes the sine of its argument. The argument is in radians and the result is DREAL.

SINH+ -- COMPUTE HYPERBOLIC SINE

SINH(DREAL)

SINH computes the hyperbolic sine of its argument. The result is DREAL.

SIZE -- GET STRING SIZE

SIZE(STRING)

SIZE returns an integer count of the length of its STRING argument.

SPAN -- GENERATE SCANNING PATTERN

SPAN(STRING) or SPAN(EXPRESSION)

SPAN creates a pattern which matches a non-null sequence of characters contained in the first argument. This argument must be a non-null STRING.

If an expression argument is used, it is evaluated during pattern matching and must yield a non-null STRING value.

STOPTR* -- STOP TRACE

STOPTR(NAME,STRING)

STOPTR terminates tracing for the name given by the first argument. The second argument designates the respect in which the trace is stopped as follows:

'VALUE' or 'V' or null (omitted)	value
'LABEL' or 'L'	label
'FUNCTION' or 'F'	function call & return
'CALL' or 'C'	function call
'RETURN' or 'R'	function return
'KEYWORD' or 'K'	keyword

SUBSTR+ -- EXTRACT SUBSTRING

SUBSTR(STRING,INTEGER,INTEGER)

SUBSTR extracts a substring from the first argument. The second argument specifies the position of the first character and the third argument specifies the number of characters.

The second argument may be positive or negative. A value of 1 specifies the first character, 2 specifies the second character, and so on. A value of -1 specifies the last character, -2 specifies the next-to-last character, and so on.

If the third argument is omitted or zero, the remainder of the string will be returned. If the third argument is larger (greater) than the number of characters remaining, all remaining characters are returned.

Note: SUBSTR works differently than in SPITBOL 360.

TAB -- GENERATE TABBING PATTERN

TAB(INTEGER) or TAB(EXPRESSION)

TAB creates a pattern which matches from the current position to the point where the indicated number of characters have been matched. The argument to TAB is a non-negative integer.

If an expression argument is used, it is evaluated during pattern matching and must yield a non-negative integer.

TABLE* -- CREATE TABLE STRUCTURE

TABLE(INTEGER)

The TABLE function creates an associative TABLE as in SIL SNOBOL4. However, in SPITBOL, the TABLE is implemented internally using a hashing algorithm. The integer argument to TABLE is the number of hash headers used. The average number of searches is about M/2N where M is the number of entries in the table, and N is the number of hash headers. Since the overhead for hash headers is small compared to the size of a TABLE element, a useful guide is to use an argument which is an estimate of the number of entries to be stored in the table. Since using even numbers of headers causes anomalies in the hashing algorithm, TABLE forces its argument odd by incrementing even arguments by one.

Note: This implementation of TABLE is compatible in that the call used in SIL SNOBOL4 will work, though possibly not with maximum efficiency.

TAN+ -- COMPUTE TANGENT

TAN(DREAL)

TAN computes the tangent of its argument. The argument is in radians and the result is DREAL.

TANH+ -- COMPUTE HYPERBOLIC TANGENT

TANH(DREAL)

TANH computes the hyperbolic tangent of its argument. The result is DREAL.

TIME -- GET EXECUTION TIME

TIME()

TIME returns the integer number of milliseconds of processor time since the start of execution. Note that the values obtained will be different (smaller) than those obtained with SIL SNOBOL4.

See "SYSDATE" on page 91 for information on how to obtain the time of day.

TOBIN+ -- CONVERT CHARACTER STRING TO BINARY STRING

TOBIN(DREAL)

TOBIN takes a string and converts it to a string of binary digits. For example,

```
TOBIN( 'HI' ) = '1100100011001001'
```

TODEC+ -- CONVERT INTEGER TO PACKED DECIMAL STRING

TODEC(INTEGER)

TODEC takes an integer and converts it to a string of packed decimal digits. For example,

```
TOHEX( TODEC( 123 ) ) = '000000000000123C'
```

TOHEX+ -- CONVERT CHARACTER STRING TO HEX STRING

TOHEX(STRING)

TOHEX takes a string of decimal digits and converts it to a string of hexadecimal digits. For example,

TOHEX('HI') = 'C8C9'

TRACE* -- INITIATE TRACE

TRACE(NAME,STRING,ARGUMENT,NAME)

The TRACE function initiates a trace of the item whose name is given by the first argument. The second argument specifies the sense of the TRACE as follows:

'VALUE' or 'V' or null (omitted)	value
'LABEL' or 'L'	label
'FUNCTION' or 'F'	function call & return
'CALL' or 'C'	function call
'RETURN' or 'R'	function return
'KEYWORD' or 'K'	keyword

Program defined trace functions are available and compatible with SIL SNOBOL4. However, the tracing of array or table elements is a feature of the SIL SNOBOL4 TRACE function which is not implemented:

Note: Keyword tracing is available for the keywords &STCOUNT, &FNCLEVEL, and &ERRTYPE.

TRIM* -- TRIM TRAILING CHARACTERS

TRIM(STRING,STRING)

TRIM returns the result of trimming trailing characters from argument 1. Argument 2 specifies the character to be trimmed. If argument 2 is omitted or null, a blank is assumed.

UNLOAD* -- UNLOAD FUNCTION

UNLOAD(STRING)

String is the name of an external function which is to be unloaded. The restriction in SIL SNOBOL4 concerning functions OPSYNed to loaded functions does not apply in SPITBOL. A function is not actually unloaded until all functions OPSYNed to it have been unloaded. SPITBOL also allows the names of ordinary functions to appear in calls to UNLOAD. In this case, the result is merely to undefine the function.

XOR+ -- COMPUTE LOGICAL EXCLUSIVE-OR

XOR(STRING,STRING)

XOR computes the logical "exclusive-or" of its arguments by exclusive-oring them together bit-by-bit. Both argument strings must have the same length.

The following is a list of the keywords implemented in SPITBOL. The notation (R) after the name indicates that the keyword is read only, that is, its value may not be modified by assignment.

Note: A restriction in SPITBOL is that the only way to change a keyword value is by a direct assignment. Keywords may not appear in any other context requiring a name (for example as the right argument of binary $).

&ABEND

Normally set to zero. If it is set to one when execution terminates, an abend dump is given. This is normally used only for system checkout.

&ABORT(R)

Contains the value of the pattern abort.

&ALPHABET(R)

Contains the 256 characters of the EBCDIC set in their natural collating sequence.

&ANCHOR

Set to zero for unanchored mode and one for anchored pattern matching mode.

&ARB(R)

Contains the value of the pattern ARB.

&BAL(R)

Contains the pattern BAL.

&CODE

The value in &CODE is used as a system return code if this job is the last in a batch. It is normally set to zero.

&DUMP

The standard value is zero. If the value is zero at the end of execution, then no symbolic dump is given. a value of one gives a dump including values of keywords and natural variables. If the value is two, the dump includes non-null array, table and program defined datatype elements as well. The dump format is self explanatory and deals with the case of branched structures including circular lists. If this value is 3 a hexadecimal dump of the compiler work areas will be generated.

&E(R)

&E contains the base for natural logarithms. Its datatype is DREAL.

&ERRTYPE

If an execution error is intercepted with the use of the SETEXIT function, then the error code is stored as an integer in &ERRTYPE. The value stored is 1000*majorcode+minorcode. Thus the error code 13.026 is stored as the integer 13026. &ERRTYPE may be assigned a value in which case an immediate error is signalled. This may be useful in signalling program detected errors. If such an error is intercepted, then either the standard error message appropriate to the major code assigned is printed, or a standard message user issued error message is printed if the major code is not in the standard range (1-14).

&ERRLIMIT

The maximum number of errors which can be trapped using the SETEXIT function. &ERRLIMIT is initially zero and is decremented each time a SETEXIT trap occurs. SETEXIT has no effect on normal error processing if &ERRLIMIT is zero.

&FAIL(R)

Contains the value of the pattern fail.

&FENCE(R)

Contains the value of the pattern FENCE.

&FNCLEVEL

Contains the current function nesting level.

&FNCLEVEL is writable and can be set to any integer between zero and its current value. The function returns triggered by decreasing its value do not trigger programmer defined trace functions or trace functions on &FNCLEVEL. Writing to &FNCLEVEL is useful in certain types of error recovery.

&FTRACE

The standard value is zero. If it is set to one, then all function calls and returns are traced.

&FULLSCAN

The standard value is zero (QUICKSCAN pattern matching mode). Value is set to one to obtain FULLSCAN mode.

&INPUT

Set to one for normal input (standard value). If set to zero, all input associations are ignored.

&LASTNO(R)

Contains the number of the last statement executed.

&MAXLNGTH

Contains the maximum permitted STRING length. This value may not exceed 32758.

&OUTPUT

Set to one for normal output (standard value). If set to zero, all output associations are ignored.

&PI(R)

&PI contains pi, the familiar constant. Its datatype is DREAL.

&REM(R)

Contains the pattern REM.

&RTNTYPE(R)

Contains 'RETURN', 'FRETURN' or 'NRETURN' depending on the type of function return most recently executed.

&STCOUNT(R)

The number of statements executed so far.

&STLIMIT(R)

The maximum number of statements allowed to be executed. The initial value is 5000. The maximum value allowed is 2**31-1.

&STNO(R)

Contains the number of the current statement.

&SUCCEED(R)

Contains the pattern succeed.

&TRACE

If the value is zero or negative, no TRACE output is generated. Each line of TRACE output decrements the value by one. The initial value is 0.

&TRIM

Set to zero for normal input mode (standard value). If the value is set to one, all input records are automatically trimmed (trailing blanks removed).

Control statements are identified by a minus sign in column one. They may occur anywhere in a source program and take effect when they are encountered. Most of these control statement types are special features of SPITBOL and are not implemented in SIL SNOBOL4.

LISTING CONTROL STATEMENTS

Listing control statements are used to alter the appearance of the listing. They have no other effect on the compilation or execution of the program. Listing control statements always occur individually.

-EJECT

The -EJECT control statement causes the compilation listing to skip to the top of the next page. The current title and sub-title (if any) are printed at the top of the page.

-SPACE

The -SPACE control statement causes spaces to be skipped on the current page. If -SPACE occurs with no operand, then one line is skipped. Alternately, an unsigned integer can be given (separated by at least one space from the -SPACE) which represents the number of lines to be skipped. If there is insufficient space on the current page, -SPACE acts like a -EJECT and the listing is spaced to the top of the next page.

-TITLE

The -TITLE statement is used to supply a title for the source program listing. The text of the title is taken from columns 8-72 of the -TITLE statement. The subtitle (if any), is cleared to blanks, and an eject to the next page occurs.

-STITL

The -STITL statement is used to supply a sub-title for the source program listing. An eject occurs to the top of the next page and the current title(if any) and the newly supplied sub-title are printed. The text for the sub-title is taken from columns 8-72 of the -STITL statement. Note that if both title and sub-title are to be changed, then the -TITLE statement should precede the -STITL statement.

OPTION CONTROL STATEMENTS

Option control statements allow selection of various compiler options. In each case, there are two modes. Two control statements allow switching from one mode to the other. The mode may be flipped back and forth within a single program. The full names are given for each control statement, however, only the first four characters are examined, and the names may thus be abbreviated to four characters. Several control options may be specified on the same control statement by separating the names with commas (no intervening spaces should occur). For example:

```
-CODE,LIST,PRINT
```

In each of the cases listed below, the default option is the the one represented by the first of the two control options listed.

-LIST -NOLIST

Normally, the source statements are listed (-LIST option). The -NOLIST option causes suppression of this printout. This may be useful for established programs known to work, or for terminal output. Note that line numbers are always listed on the left which is convenient for terminal output. If compilation errors are detected, the offending statements are printed regardless of the setting of the list mode.

-NOCODE -CODE

The -CODE option causes a printout of the generated code in assembly language type format. This listing may be useful in determining how SPITBOL handles the compilation of various types of statements. The -NOCODE control option resets the normal mode of no code listing. It is permissible to use these statements in combination to obtain listings for selected sections of the source program. The code listing occurs after the end of the source listing starting on a separate page so that the source listing is not affected.

-NOPRINT -PRINT

Normally, control statements are not printed (-NOPRINT). The -PRINT option causes control statements to be listed (provided that the -LIST option is in effect). This option may be useful if serialization is used for updating purposes.

-SINGLE -DOUBLE

The compilation listing is normally single spaced (-SINGLE). The -DOUBLE option causes double spacing to be used, with a blank line between each listed line.

-OPTIMIZE -NOOPTIMIZE

The SPITBOL compiler operates in an optimized mode where the following assumptions are made:

1. The values of BAL, ARB, FENCE, ABORT, REM, FAIL and SUCCEED are not modified during execution.

2. The standard system functions are not redefined.

3. Function calls in a statement do not result in modification of values of variables referenced elsewhere in the same statement.

The -NOOPTIMIZE control statement specifies that the compiler not make the above assumptions. This results in a higher level of compatibility with SIL SNOBOL4 at the expense of both space and speed. In some cases, the loss of speed may be as much as a factor of ten. The optimizing mode may be switched on and off so that only isolated statements are compiled in non-optimized mode.

Note: It is the references to redefined functions which cause the trouble, not the actual definition itself.

-IN72 -IN80

Normally, the compiler reads only columns 1-72 of the input images. Columns 73-80 may be used for serialization. The serialization will be listed on the source listing separated from the program text by a column of dots (this is to prevent accidentally punching past column 72). The -IN80 option causes all 80 columns of the input statements to be read. The -IN72 statement resets the normal option. -IN80 should be used from a terminal device, as it eliminates output on the right side of the page.

-NOSEQUENCE -SEQUENCE

This option is only relevant if -IN72 is in effect. The normal mode (-NOSEQUENCE) ignores any serialization occurring in columns 73-80. If the -SEQUENCE option is taken, then the SPITBOL compiler tests to see whether the serialization is in correct ascending sequence. If an out of sequence statement occurs, a message is printed, but no other action is taken (unless -NOERRORS is also specified at the time of the sequence error).

-ERRORS -NOERRORS

Normally execution is allowed even if compilation errors occur (-ERRORS). If a compilation error or a sequence error (-SEQUENCE on) occurs and the -NOERRORS option has been specified, then the execution of the program is suppressed.

-FAIL -NOFAIL

In SIL SNOBOL4, and in SPITBOL with the -FAIL mode set, a failure in a statement with no conditional GOTO is ignored and the program execution resumes with the next statement in sequence. This convention often results in errors going undetected, particularly in the case of array references with out of range subscripts and pattern matches which are expected to always succeed. The -NOFAIL option changes this convention. If a statement having no conditional GOTO is compiled under the -NOFAIL mode, and a failure occurs when the statement is executed, an execution error occurs and a suitable message is generated. The -NOFAIL operation is particularly useful for student jobs and other situations where many small programs are being debugged.

-EXECUTE -NOEXECUTE

Normally execution is initiated following compilation. The -NOEXECUTE option, if set at the end of compilation, inhibits execution. This is often useful in conjunction with the option to generate object modules.

COPY CONTROL STATEMENT

-COPY FILENAME

The -COPY control statement allows a section of coding to be copied into the source from an external file. The compiler proceeds as though the text in the file had been read instead of the -COPY statement. Filename is any filename wich would be legal as the second argument to the input function. In particular, OS allows member names to be supplied in parentheses after the DDNAME which allows sections of code (for example, function definitions), to be stored as members of a partitioned dataset. The text copied in may itself contain -COPY statements to a maximum nesting of eight levels.

There are two major divisions of error messages which include compilation type errors and execution type errors.

COMPILATION ERROR MESSAGES

When the compiler detects an error, a flag is placed under the point in the statement where the error was discovered and processing of the statement in error is discontinued. Compilation continues with the next statement. Execution is not suppressed unless the -NOERRORS option has been set (see "Control Statements" on page 49). If an attempt is made to execute a statement found erroneous by the compiler, an execution error occurs. Compiler error messages are surrounded by ****** so they are easy to find. The following section describes the various error messages.

******ATTENTION RECEIVED******

An attention was received during compilation.

******ERROR IN GO TO FIELD******

The GOTO field is incorrectly formed.

******ERROR IN NUMERIC ITEM******

A numeric item is illegally constructed.

******EXPRESSION IS TOO COMPLEX FOR THE COMPILER******

The expression being compiled overflows work areas in the SPITBOL. The expression must be broken into two or more statements.

******ILLEGAL CHARACTER******

The compiler detected a character which has no syntactic meaning in the SNOBOL4 language outside a STRING literal.

******ILLEGAL CONTINUATION******

The compiler detected an illegal continuation of a statement.

******ILLEGAL TRANSFER ADDRESS******

The operand on an end statement is not a simple variable. The operand is ignored and execution starts with the first statement.

******ILLEGAL USE OF , ******

A comma has been used in an illegal context. The only legal uses of comma are to separate array subscripts and function arguments. Note that this error can be caused by inserting a blank between the function name and the left parenthesis.

******ILLEGAL USE OF < ******

The character < (array left bracket) has been used in a context where an array left bracket cannot legally occur.

******ILLEGAL USE OF) ******

A right parenthesis is used in an illegal context.

******ILLEGAL USE OF > ******

An array right bracket has been used in an illegal context. This character can be used only to terminate a list or array subscripts.

******ILLEGAL USE OF = ******

An equal sign has been used in an illegal context. only one equal sign may occur in a statement.

*******INPUT RECORD TOO LONG*******

The compiler read an input record that was too long.

*******INVALID -COPY CARD*******

A -COPY statement has an incorrect filename (this could result from an error in system control statement setup), or -COPY has been nested more than eight levels. compilation proceeds after ignoring the erroneous statement.

*******LABEL HAS BEEN PREVIOUSLY DEFINED*******

The statement has a label which has already been used. Compilation of the statement is discontinued and the earlier definition of the label is retained.

******MISSING END CARD SUPPLIED*******

An end of file was read on the system input file (SYSIN) during compilation. The compiler supplies an end statement and initiates execution unless the -NOERRORS option is set.

******MISSING OPERAND******

This message is generated when the compiler expects an operand and none is found. For example:

```
A / / B, (C+)
```

******MISSING OPERATOR******

The compiler expected an operator and no operator was found. This occurs in situations like (X)A, where an operator is expected after the right parenthesis. This message is also given when the blanks surrounding a binary operator are omitted.

******NON-RECOVERABLE INPUT ERROR******

A non-recoverable input error has been signalled on the system input file (SYSIN). This is a fatal error which terminates compilation and prevents execution. Note that it also cancels any subsequent jobs when a batched run is being processed.

******PAGE LIMIT EXCEEDED******

The number of pages produced exceeds the value specified by the P parameter.

******PROGRAM TOO LONG FOR AVAILABLE STORAGE******

The storage required by the program exceeds available storage. Increase the region allocated and/or the H parameter in the compiler parameter field. Note that storage for execution time use has not yet been allocated. This must be taken into consideration in deciding how much additional memory to allocate. This is a fatal error which terminates compilation and prevents execution.

******TIME LIMIT EXCEEDED******

The time limit exceeded that specified by the T parameter.

******UNBALANCED () OR <> ******

This occurs if the parentheses or array brackets in a statement are not properly balanced.

******UNDEFINED TRANSFER ADDRESS******

The label used on an END statement is not defined. The operand is ignored, and execution starts with the first statement.

******UNMATCHED QUOTE******

A STRING literal has been started but not properly terminated. Note that STRING literals cannot be split over continuation statements.

EXECUTION ERROR MESSAGES

The execution package performs extensive error checking. When an error is detected, execution is terminated with an error message unless the error is intercepted by means of the SETEXIT function. The message is accompanied by an error code of the form AA.BBB, where AA is the major code and BBB is the minor code. The major code refers to the message given (see below). The minor code further identifies the exact error. The following is a list and explanation of the error messages together with their major codes.

Major = 1 Illegal DATATYPE

In a context where a definite datatype is required, a value of the wrong datatype is and the attempt to convert it to the correct datatype fails.

Major = 2 Unexpected failure

A statement having no conditional GOTO failed with the -NOFAIL option set. This usually corresponds to an error such as an unexpected out of range subscript.

Major = 3 Error in ARRAY reference

An array reference is incorrect. Either the object referenced is not an array or table, or the wrong number of subscripts is given.

Major = 4 Compiler detected error

An attempt was made to execute a statement found erroneous by the compiler. This message is also issued from statement number 'zero' if compiler errors were detected with the -NOERRORS option set.

Major = 5 Error in reference to keyword

An error was made in a keyword reference. Either the operand of & is incorrect, or the value stored is incorrect.

Major = 6 Memory overflow

Dynamic memory is exhausted. Note that this can occur as a result of runaway recursion in function references or pattern matching.

Major = 7 Evaluation of GOTO failed

If a complex expression is used in the GOTO field, it is not allowed to fail. Such a failure within a GOTO expression did occur.

Major = 8 Error in GOTO

The operand of a GOTO must be a natural variable which is a defined label. Some other value was given. This error message is also given on a return from level zero.

Major = 9 Call to undefined function or operator

A reference was made to an undefined function, or an undefined operator was used.

Major = 10 Error in arithmetic operation

This message covers a variety of arithmetic errors such as overflow, division by zero, etc.

Major = 11 Keyword or system limit exceeded

This message is issued when any of the following limits is exceeded -- time, page or statement system limits, &MAXLNGTH, &STLIMIT keyword limit.

Major = 12 Input/output or other system error

An error has been signalled by one of the operating system routines. Some examples are non-recoverable I/O error, LOAD on a non-existent function, etc.

Major = 13 Incorrect value for function or operator

An argument to a function or operand of an operator was of the right datatype, but outside the range of values permitted for some particular use. For example, the null STRING is an illegal argument for the BREAK function.

Major = 14 Value returned where NAME is required

In a context requiring a NAME (left side of =, GOTO expression, right argument of $ or .)

User issued error message

This message is given if &ERRTYPE is assigned a value greater than 14999, or less than 1000.

EXECUTION ERROR CODE LIST

This section gives the complete list of all execution error codes.

Major = 1 Illegal Datatype Errors

1.001 EVALUATED RESULT OF DEFERRED ARGUMENT TO POS IS NOT AN INTEGER
1.002 EVALUATED RESULT OF DEFERRED ARGUMENT TO RPOS IS NOT AN INTEGER
1.003 EVALUATED RESULT OF DEFERRED ARGUMENT TO RTAB IS NOT AN INTEGER
1.004 EVALUATED RESULT OF DEFERRED ARGUMENT TO TAB IS NOT AN INTEGER
1.005 EVALUATED RESULT OF DEFERRED ARGUMENT TO LEN IS NOT AN INTEGER
1.006 EVALUATED RESULT OF DEFERRED ARGUMENT TO ANY IS NOT A STRING
1.007 EVALUATED RESULT OF DEFERRED ARGUMENT TO NOTANY IS NOT A STRING
1.008 EVALUATED RESULT OF DEFERRED ARGUMENT TO SPAN IS NOT A STRING
1.009 EVALUATED RESULT OF DEFERRED ARGUMENT TO BREAKX IS NOT A STRING
1.010 EVALUATED RESULT OF DEFERRED ARGUMENT TO BREAK IS NOT A STRING

1.011 EVALUATED RESULT OF DEFERRED EXPRESSION USED IN A PATTERN MATCH IS NOT A STRING OR PATTERN
1.012 VALUE TO BE STORED IN A KEYWORD IS NOT AN INTEGER
1.013 REAL ARGUMENT TO LOADED FUNCTION IS NOT A REAL
1.014 INTEGER ARGUMENT TO LOADED FUNCTION IS NOT AN INTEGER
1.015 STRING ARGUMENT TO LOADED FUNCTION IS NOT A STRING
1.016 DREAL ARGUMENT TO LOADED FUNCTION IS NOT A DREAL
1.017 OPERAND OF UNARY $ IS NOT A NAME
1.018 REPLACING RIGHT HAND SIDE IN A PATTERN REPLACEMENT IS NOT A STRING
1.019 SUBJECT OF A PATTERN MATCH IS NOT A STRING
1.020 THE PATTERN IN A PATTERN MATCH IS NOT A PATTERN
1.021 SUBSCRIPT IN REFERENCE TO ONE DIMENSIONAL ARRAY IS NOT AN INTEGER
1.022 SUBSCRIPT IN REFERENCE TO A MULTI-DIMENSIONAL ARRAY IS NOT AN INTEGER
1.023 A FIELD FUNCTION WAS APPLIED TO AN INAPPROPRIATE PROGRAM DEFINED DATATYPE
1.024 THE LEFT OPERAND FOR ALTERNATION OR CONCATENATION IS NOT A STRING OR PATTERN
1.025 THE RIGHT OPERAND FOR ALTERNATION OR CONCATENATION IS NOT A STRING OR PATTERN
1.026 THE ARGUMENT TO A FIELD FUNCTION IS NOT A PROGRAM DEFINED DATATYPE
1.027 AN OPERAND OF BINARY + IS NON-NUMERIC
1.028 AN OPERAND OF BINARY - IS NON-NUMERIC
1.029 AN OPERAND OF BINARY * IS NON-NUMERIC
1.030 AN OPERAND BINARY / IS NON-NUMERIC
1.031 AN ARGUMENT TO NE,EQ,LE,GE,LT,GT IS NON-NUMERIC
1.032 AN OPERAND OF BINARY ** IS NON-NUMERIC
1.033 THE OPERAND OF UNARY + IS NON-NUMERIC
1.034 THE OPERAND OF UNARY - IS NON-NUMERIC
1.035 FIRST ARGUMENT TO LEQ,LNE,LGT,LLT,LGE OR LLE IS NOT A STRING
1.036 SECOND ARGUMENT TO LEQ,LNE,LGT,LLT,LGE OR LLE IS NOT A STRING
1.037 ARGUMENT TO SIZE IS NOT A STRING
1.038 LEFT OPERAND OF BINARY $ OR . IS NOT A PATTERN
1.039 ARGUMENT TO LEN IS NOT AN INTEGER OR EXPRESSION
1.040 ARGUMENT TO POS IS NOT AN INTEGER OR EXPRESSION
1.041 ARGUMENT TO TAB IS NOT AN INTEGER OR EXPRESSION
1.042 ARGUMENT TO RPOS IS NOT AN INTEGER OR EXPRESSION
1.043 ARGUMENT TO RTAB IS NOT AN INTEGER OR EXPRESSION
1.044 ARGUMENT TO SPAN IS NOT A STRING OR EXPRESSION
1.045 ARGUMENT TO BREAKX IS NOT A STRING OR EXPRESSION
1.046 ARGUMENT TO BREAK IS NOT A STRING OR EXPRESSION
1.047 ARGUMENT TO NOTANY IS NOT A STRING OR EXPRESSION
1.048 ARGUMENT TO ANY IS NOT A STRING OR EXPRESSION
1.049 ARGUMENT TO VALUE IS NOT A STRING, NAME OR CORRECT PROGRAMMER DEFINED DATATYPE
1.050 ARGUMENT TO ARBNO IS NOT A PATTERN
1.051 FIRST ARGUMENT TO APPLY IS NOT THE NAME OF A FUNCTION
1.052 FIRST ARGUMENT TO ARG IS NOT A NAME
1.053 SECOND ARGUMENT TO ARG IS NOT AN INTEGER
1.054 FIRST ARGUMENT TO ARRAY IS NOT A STRING
1.055 FIRST ARGUMENT TO CLEAR IS NOT A STRING
1.056 ARGUMENT TO CODE IS NOT A STRING
1.057 ARGUMENT TO COLLECT IS NOT AN INTEGER
1.058 SECOND ARGUMENT TO CONVERT IS NOT A STRING
1.059 ARGUMENT TO DATA IS NOT A STRING
1.060 FIRST ARGUMENT TO DEFINE IS NOT A STRING
1.061 SECOND ARGUMENT TO DEFINE IS NON-NULL AND IS NOT THE NAME OF A LABEL
1.062 ARGUMENT TO DETACH IS NOT THE NAME OF A NATURAL VARIABLE
1.063 SECOND ARGUMENT TO DUPL IS NOT AN INTEGER
1.064 FIRST ARGUMENT TO DUPL IS NOT A STRING
1.065 ARGUMENT TO ENDFILE IS NOT A STRING
1.066 ARGUMENT TO EVAL IS NOT AN EXPRESSION (OR A STRING, WHICH COULD BE CONVERTED INTO AN EXPRESSION)
1.067 FIRST ARGUMENT TO FIELD IS NOT A NAME
1.068 SECOND ARGUMENT TO FIELD IS NOT AN INTEGER

1.069 FIRST ARGUMENT TO INPUT IS NOT THE NAME OF A NATURAL VARIABLE
1.070 FILE NAME (SECOND ARGUMENT) TO INPUT IS NOT A STRING
1.071 FORMAT SPECIFICATION (THIRD ARGUMENT) TO INPUT IS NOT AN INTEGER
1.072 ARGUMENT TO LOAD IS NOT A STRING
1.073 FIRST ARGUMENT TO LOC IS NOT A NAME
1.074 SECOND ARGUMENT TO LOC IS NOT AN INTEGER
1.075 THIRD ARGUMENT TO LPAD IS NOT A STRING
1.076 SECOND ARGUMENT TO LPAD IS NOT AN INTEGER
1.077 FIRST ARGUMENT TO LPAD IS NOT A STRING
1.078 FIRST ARGUMENT TO OPSYN IS NOT THE NAME OF A NATURAL VARIABLE
1.079 SECOND ARGUMENT TO OPSYN IS NOT A FUNCTION NAME
1.080 FIRST ARGUMENT TO OUTPUT IS NOT THE NAME OF A NATURAL VARIABLE
1.081 FILE NAME (SECOND ARGUMENT) FOR OUTPUT FUNCTION IS NOT A STRING.
1.082 FORMAT SPECIFICATION (THIRD ARGUMENT) FOR OUTPUT FUNCTION IS NOT A STRING
1.083 ARGUMENT TO PROTOTYPE IS NOT AN ARRAY OR TABLE
1.084 SECOND ARGUMENT TO REMDR IS NOT AN INTEGER
1.085 FIRST ARGUMENT TO REMDR IS NOT AN INTEGER
1.086 THIRD ARGUMENT TO REPLACE IS NOT A STRING
1.087 SECOND ARGUMENT TO REPLACE IS NOT A STRING
1.088 FIRST ARGUMENT TO REPLACE IS NOT A STRING
1.089 ARGUMENT TO REVERSE IS NOT A STRING
1.090 ARGUMENT TO REWIND IS NOT A STRING
1.091 THIRD ARGUMENT TO RPAD IS NOT AN INTEGER
1.092 SECOND ARGUMENT TO RPAD IS NOT AN INTEGER
1.093 FIRST ARGUMENT TO RPAD IS NOT A STRING
1.094 ARGUMENT TO SETEXIT IS NOT A LABEL NAME
1.095 FIRST ARGUMENT TO SUBSTR IS NOT A STRING
1.096 SECOND ARGUMENT TO SUBSTR IS NOT AN INTEGER
1.097 THIRD ARGUMENT TO SUBSTR IS NOT AN INTEGER
1.098 ARGUMENT TO TABLE IS NOT AN INTEGER
1.099 ARGUMENT TO TRIM IS NOT A STRING
1.100 ARGUMENT TO UNLOAD IS NOT THE NAME OF A FUNCTION
1.101 SECOND ARGUMENT TO LOAD IS NOT A STRING
1.102 SECOND ARGUMENT TO TRIM IS NOT A STRING
1.103 ARGUMENT TO ABS IS NOT NUMERIC
1.104 FIRST ARGUMENT TO AND IS NOT A STRING
1.105 SECOND ARGUMENT TO AND IS NOT A STRING
1.106 ARGUMENT TO ARCCOS IS NOT A DREAL
1.107 ARGUMENT TO ARCSIN IS NOT A DREAL
1.108 ARGUMENT TO ARCTAN IS NOT A DREAL
1.109 ARGUMENT TO ARCCOSH IS NOT A DREAL
1.110 ARGUMENT TO ARCSINH IS NOT A DREAL
1.111 ARGUMENT TO ARCTANH IS NOT A DREAL
1.112 ARGUMENT TO COMPL IS NOT A STRING
1.113 ARGUMENT TO COS IS NOT A DREAL
1.114 ARGUMENT TO SIN IS NOT A DREAL
1.115 ARGUMENT TO TAN IS NOT A DREAL
1.116 ARGUMENT TO GAMMA IS NOT A DREAL
1.117 FIRST ARGUMENT TO MOD IS NOT A DREAL
1.118 SECOND ARGUMENT TO MOD IS NOT A DREAL
1.119 FIRST ARGUMENT TO EXP IS NOT A DREAL
1.120 SECOND ARGUMENT TO EXP IS NOT A DREAL
1.121 ARGUMENT TO FROMBIN IS NOT A STRING
1.122 ARGUMENT TO FROMHEX IS NOT A STRING
1.123 SECOND ARGUMENT TO BIT IS NOT AN INTEGER
1.124 FIRST ARGUMENT TO BIT IS NOT A STRING
1.125 ARGUMENT TO FROMDEC IS NOT A STRING
1.126 INVALID ARGUMENT PASSED TO DCONV
1.127 INVALID DATATYPE PASSED TO ICONV
1.128 FIRST ARGUMENT TO LOG IS NOT A DREAL
1.129 SECOND ARGUMENT TO LOG IS NOT A DREAL
1.130 FIRST ARGUMENT TO MAX IS NOT NUMERIC
1.131 An ARGUMENT TO MAX OTHER THAN THE FIRST WAS NOT A REAL WHEN THE FIRST ARGUMENT WAS A REAL

1.132 An ARGUMENT TO MAX OTHER THAN THE FIRST WAS NOT AN INTEGER WHEN THE FIRST ARGUMENT WAS AN INTEGER
1.133 An ARGUMENT TO MAX OTHER THAN THE FIRST WAS NOT A DREAL WHEN THE FIRST ARGUMENT WAS A DREAL
1.134 FIRST ARGUMENT TO MIN IS NOT NUMERIC
1.135 An ARGUMENT TO MIN OTHER THAN THE FIRST WAS NOT A REAL WHEN THE FIRST ARGUMENT WAS A REAL
1.136 An ARGUMENT TO MIN OTHER THAN THE FIRST WAS NOT AN INTEGER WHEN THE FIRST ARGUMENT WAS AN INTEGER
1.137 An ARGUMENT TO MIN OTHER THAN THE FIRST WAS NOT A DREAL WHEN THE FIRST ARGUMENT WAS A DREAL
1.138 SECOND ARGUMENT TO OR IS NOT A STRING
1.139 FIRST ARGUMENT TO OR IS NOT A STRING
1.140 THIRD ARGUMENT TO RANDOM IS NOT AN INTEGER
1.141 SECOND ARGUMENT TO RANDOM IS NOT AN INTEGER
1.142 FIRST ARGUMENT TO RANDOM IS NOT AN INTEGER
1.143 ILLEGAL ARGUMENT PASSED TO RCONV FUNCTION
1.144 FIRST ARGUMENT TO RTRIM IS NOT A STRING
1.145 SECOND ARGUMENT TO RTRIM IS NOT A STRING
1.146 ILLEGAL DATATYPE PASSED TO SCONV FUNCTION
1.147 ARGUMENT TO TOBIN IS NOT A STRING
1.148 ARGUMENT TO TOHEX IS NOT A STRING
1.149 ARGUMENT TO TODEC IS NOT AN INTEGER
1.150 SECOND ARGUMENT TO XOR IS NOT A STRING
1.151 FIRST ARGUMENT TO XOR IS NOT A STRING
1.152 ARGUMENT TO COSH IS NOT A DREAL
1.153 ARGUMENT TO SINH IS NOT A DREAL
1.154 ARGUMENT TO TANH IS NOT A DREAL
1.155 THIRD ARGUMENT TO BITSET IS NOT AN INTEGER
1.156 SECOND ARGUMENT TO BITSET IS NOT AN INTEGER
1.157 FIRST ARGUMENT TO BITSET IS NOT A STRING

Major = 2 Unexpected Failure Error

2.001 FAILURE OF A STATEMENT HAVING NO CONDITIONAL GOTO WITH -NOFAIL OPTION IN EFFECT

Major = 3 Array Reference Errors

3.001 ARRAY REFERENCE WITH ONE SUBSCRIPT REFERS TO AN OBJECT WHICH IS NEITHER A TABLE NOR AN ARRAY
3.002 MULTI-DIMENSIONAL ARRAY REFERENCE REFERS TO AN OBJECT WHICH IS NOT AN ARRAY
3.003 WRONG NUMBER OF SUBSCRIPTS IN AN ARRAY REFERENCE

Major = 4 Compiler Detected Error

4.001 ATTEMPTED EXECUTION OF A STATEMENT FOUND ERRONEOUS BY THE COMPILER.

Major = 5 Keyword Reference Errors

5.001 AN ATTEMPT WAS MADE TO REFERENCE THE KEYWORD ATTRIBUTE OF A NON-NATURAL VARIABLE
5.002 REFERENCE TO AN UNDEFINED KEYWORD
5.003 AN ATTEMPT WAS MADE TO CHANGE THE VALUE OF A KEYWORD ASSOCIATED WITH A NON-NATURAL VARIABLE
5.004 ATTEMPT TO CHANGE THE VALUE OF AN UNDEFINED KEYWORD
5.005 ATTEMPT TO CHANGE THE VALUE OF A PROTECTED KEYWORD

Major = 6 Memory Overflow Error

6.001 OVERFLOW IN MAIN DYNAMIC STORAGE AREA. THIS CAN OCCUR AS A RESULT OF RUNAWAY RECURSION IN PATTERN MATCHING OR FUNCTION REFERENCE AS WELL AS FROM GENERATION OF TOO MUCH DATA.

Major = 7 Evaluation of GOTO Error

7.001 THE EVALUATION OF A COMPLEX GOTO FAILED

Major = 8 GOTO Errors

8.001 RETURN FROM FUNCTION LEVEL ZERO
8.002 TRANSFER TO AN UNDEFINED LABEL
8.003 A TRANSFER TO THE LABEL CONTINUE OCCURRED, BUT NO PREVIOUS ERROR HAD BEEN INTERCEPTED
8.004 A TRANSFER TO THE LABEL ABORT OCCURRED, BUT NO PREVIOUS ERROR HAD BEEN INTERCEPTED
8.005 NAME USED AS A GOTO OPERAND IS NOT THE NAME OF A NATURAL VARIABLE
8.006 THE OPERAND OF A DIRECT GOTO IS NOT CODE.

Major = 9 Undefined Function Errors

9.001 REFERENCE TO AN UNDEFINED FUNCTION
9.002 USE OF THE UNDEFINED OPERATOR -- UNARY /
9.003 USE OF THE UNDEFINED OPERATOR -- BINARY &
9.004 USE OF THE UNDEFINED OPERATOR -- BINARY ¬
9.005 USE OF THE UNDEFINED OPERATOR -- BINARY @
9.006 USE OF THE UNDEFINED OPERATOR -- UNARY |
9.007 USE OF THE UNDEFINED OPERATOR -- UNARY #
9.008 USE OF THE UNDEFINED OPERATOR -- BINARY #
9.009 USE OF THE UNDEFINED OPERATOR -- BINARY ?
9.010 USE OF THE UNDEFINED OPERATOR -- UNARY %
9.011 USE OF THE UNDEFINED OPERATOR -- BINARY %
9.012 USE OF THE UNDEFINED OPERATOR - UNARY EXCLAMATION POINT.

Major = 10 Arithmetic Operation Errors

10.001 OVERFLOW IN + - / OR * OF TWO DREALS
10.002 OVERFLOW IN + - / OR * OF TWO REALS
10.003 REAL DIVISION BY ZERO
10.004 DREAL DIVISION BY ZERO
10.005 OVERFLOW IN REAL ** INTEGER OR DREAL ** INTEGER
10.006 INTEGER DIVISION BY ZERO
10.007 INTEGER ADDITION OVERFLOW
10.008 INTEGER SUBTRACTION OVERFLOW
10.009 INTEGER MULTIPLICATION OVERFLOW
10.010 NEGATIVE EXPONENT FOR INTEGER ** INTEGER
10.011 OVERFLOW IN INTEGER EXPONENTIATION
10.012 DREAL ** DREAL IS NOT PERMITTED
10.013 REAL ** REAL IS NOT PERMITTED
10.014 INTEGER OVERFLOW FOR UNARY MINUS (HAPPENS ONLY WITH LARGEST NEG NUM)
10.015 ATTEMPTED DIVISION BY ZERO IN REMDR FUNCTION

Major = 11 Keyword or System Limit Errors

11.001 PAGE LIMIT (P PARAMETER) EXCEEDED
11.002 CARD LIMIT (C PARAMETER) EXCEEDED
11.003 INPUT RECORD LONGER THAN &MAXLNGTH
11.004 ATTEMPT TO SET &MAXLNGTH TO A VALUE GREATER THAN THE MAXIMUM ALLOWED (32758)
11.005 &STLIMIT SET TO A VALUE LESS THAN THE NUMBER OF STATEMENTS ALREADY EXECUTED
11.006 STATEMENT LIMIT (&STLIMIT) EXCEEDED
11.007 ATTEMPT TO FORM A STRING LONGER THAN &MAXLNGTH BY CONCATENATION
11.008 A PATTERN STRUCTURE HAS EXCEEDED THE MAXIMUM PERMITTED SIZE (32K BYTES)
11.009 TIME LIMIT (T PARAMETER) EXCEEDED
11.010 ATTEMPT TO FORM A STRING LONGER THAN &MAXLNGTH IN CALL TO DUPL FUNCTION

11.011 ATTEMPT TO FORM A STRING LONGER THAN &MAXLNGTH IN CALL TO LPAD FUNCTION
11.012 ATTEMPT TO FORM A STRING LONGER THAN &MAXLNGTH IN CALL TO RPAD FUNCTION
11.013 ATTEMPT TO SET &FNCLEVEL TO NEGATIVE
11.014 ATTEMPT TO INCREASE THE VALUE OF &FNCLEVEL
11.015 EXTERNAL FUNCTION RETURNED A STRING EXCEEDING &MAXLNGTH
11.016 ATTEMPT TO FORM A STRING LONGER THAN &MAXLNGTH IN CALL TO TOBIN FUNCTION
11.017 ATTEMPT TO FORM A STRING EXCEEDING &MAXLNGTH IN CALL TO TOHEX FUNCTION

Major = 12 System Errors

12.xxx (SEE SECTION ON SYSTEM ERROR CODES)

Major = 13 Incorrect Value Errors

13.001 EVALUATED RESULT OF DEFERRED ARGUMENT TO POS IS NEGATIVE
13.002 EVALUATED RESULT OF DEFERRED ARGUMENT TO RPOS IS NEGATIVE
13.003 EVALUATED RESULT OF DEFERRED ARGUMENT TO RTAB IS NEGATIVE
13.004 EVALUATED RESULT OF DEFERRED ARGUMENT TO TAB IS NEGATIVE
13.005 EVALUATED RESULT OF DEFERRED ARGUMENT TO LEN IS NEGATIVE
13.006 EVALUATED RESULT OF DEFERRED ARGUMENT TO ANY IS NULL
13.007 EVALUATED RESULT OF DEFERRED ARGUMENT TO NOTANY IS NULL
13.008 EVALUATED RESULT OF DEFERRED ARGUMENT TO SPAN IS NULL
13.009 EVALUATED RESULT OF DEFERRED ARGUMENT TO BREAKX IS NULL
13.010 EVALUATED RESULT OF DEFERRED ARGUMENT TO BREAK IS NULL
13.011 OPERAND OF UNARY $ IS NULL
13.012 ARGUMENT FOR LEN IS NEGATIVE
13.013 ARGUMENT FOR POS IS NEGATIVE
13.014 ARGUMENT FOR TAB IS NEGATIVE
13.015 ARGUMENT FOR RPOS IS NEGATIVE
13.016 ARGUMENT FOR RTAB IS NEGATIVE
13.017 SPAN ARGUMENT IS NULL
13.018 ARGUMENT FOR BREAKX IS NULL
13.019 ARGUMENT FOR BREAK IS NULL
13.020 NOTANY ARGUMENT IS NULL
13.021 ANY ARGUMENT IS NULL
13.022 NULL FIRST ARGUMENT IN CALL TO THE ARRAY FUNCTION
13.023 AN ARRAY BOUND IN A CALL TO THE ARRAY FUNCTION IS NULL
13.024 AN ARRAY BOUND IN A CALL TO THE ARRAY FUNCTION IS NON-NUMERIC
13.025 IN THE FIRST ARGUMENT TO ARRAY, A SUBSCRIPT BOUND HAS TWO COLONS
13.026 AN ARRAY LOWER BOUND IN A CALL TO THE ARRAY FUNCTION IS NOT IN THE RANGE -32768 < LBD < +32768
13.027 AN ARRAY DIMENSION (HBD-LBD+1) IN A CALL TO THE ARRAY FUNCTION IS NOT IN THE RANGE 0 < DIM < 32768
13.028 NAME IN CLEAR FIRST ARGUMENT IS NULL
13.030 ARGUMENT TO DATA IS NULL
13.031 DATATYPE NAME IN ARGUMENT TO DATA IS NULL
13.032 MISSING LEFT PAREN IN DATA ARGUMENT
13.033 FIELD NAME IS NULL IN DATA ARGUMENT
13.034 DATA ARGUMENT DOES NOT END WITH)
13.035 TOO MANY FIELDS (MORE THAN 30), IN ARGUMENT TO DATA
13.036 FIRST ARGUMENT TO DEFINE IS NULL
13.037 FUNCTION NAME IN FIRST ARGUMENT TO DEFINE IS MISSING (NULL)
13.038 FIRST ARGUMENT TO DEFINE IS MISSING A LEFT PAREN
13.039 ARGUMENT NAME IN FIRST ARGUMENT TO DEFINE IS NULL
13.040 FIRST ARGUMENT TO DEFINE IS MISSING A)
13.041 NULL LOCAL NAME IN FIRST ARGUMENT TO DEFINE
13.042 ARGUMENT TO ENDFILE IS NULL
13.043 ARGUMENT TO LOAD IS NULL
13.044 FUNCTION NAME IN ARGUMENT TO LOAD IS NULL
13.045 MISSING (IN ARGUMENT TO LOAD
13.046 MISSING) IN ARGUMENT TO LOAD

13.047 TOO MANY ARGUMENTS (MORE THAN 64) IN FUNCTION TO BE LOADED
13.048 ARGUMENT TO REWIND IS NULL
13.049 ARGUMENT TO TABLE IS ZERO OR NEGATIVE
13.050 SECOND ARGUMENT TO TRIM IS LONGER THAN ONE CHARACTER
13.051 ARGUMENTS TO AND ARE NOT THE SAME LENGTHS
13.052 EVALUATION OF ARCCOS FAILED
13.053 EVALUATION OF ARCSIN FAILED
13.054 EVALUATION OF ARCTAN FAILED
13.055 EVALUATION OF ARCCOSH FAILED
13.056 EVAULATION OF ARCSINH FAILED
13.057 EVALUATION OF ARCTANH FAILED
13.058 EVALUATION OF COS FAILED
13.059 EVALUATION OF SIN FAILED
13.060 EVAULATION OF TAN FAILED
13.061 EVALUATION OF GAMMA FAILED
13.062 EVALUATION OF MOD FAILED
13.063 EVALUATION OF EXP FAILED
13.064 LENGTH OF ARGUMENT TO FROMBIN NOT A MULTIPLE OF EIGHT
13.065 ARGUMENT TO FROMBIN CONTAINS CHARACTERS OTHER THAN ZEROES AND ONES
13.066 LENGTH OF ARGUMENT TO FROMHEX IS NOT A MULTIPLE OF TWO
13.067 ARGUMENT TO FROMHEX CONTAINS ILLEGAL HEX CHARACTERS
13.068 SECOND ARGUMENT TO BIT IS NEGATIVE
13.069 FIRST ARGUMENT TO BIT IS NULL
13.070 SECOND ARGUMENT TO BIT IS NOT WITHIN FIRST ARGUMENT
13.071 ARGUMENT TO FROMDEC LONGER THAN 16 CHARACTERS
13.072 ARGUMENT TO FROMDEC IS NOT VALID PACKED DECIMAL
13.073 EVALUATION OF LOG FUNCTION FAILED
13.074 ARGUMENTs TO OR DO NOT HAVE SAME LENGTH
13.075 SECOND ARGUMENT TO RANDOM IS NEGATIVE
13.076 FIRST ARGUMENT TO RANDOM IS NOT POSITIVE
13.077 SCALING ERROR IN ARGUMENTS TO RANDOM
13.078 SECOND ARGUMENT TO RTRIM IS LONGER THAN ONE CHARACTER
13.079 ARGUMENTs TO XOR DO NOT HAVE THE SAME LENGTHS
13.080 EVALUATION OF COSH FUNCTION FAILED
13.081 EVALATION OF SINH FUNCTION FAILED
13.082 EVALUATION OF TANH FUNCTION FAILED
13.083 THIRD ARGUMENT TO BITSET NOT ZERO OR ONE
13.084 SECOND ARGUMENT TO BITSET IS NEGATIVE
13.085 SECOND ARGUMENT TO BITSET IS NOT WITHIN FIRST ARGUMENT
13.086 FIRST ARGUMENT TO BITSET IS NULL

Major = 14 Value Returned Errors

14.001 A FUNCTION CALLED BY NAME RETURNED A VALUE
14.002 AN EXPRESSION OTHER THAN A FUNCTION CALL RETURNED A VALUE WHERE A NAME WAS REQUIRED

SYSTEM ERROR CODES FOR OS

This section gives the list of all system errors that can be signalled by the OS interface.

12.001 INVALID FILE NAME
12.002 MISSING DD CARD
12.003 MODULE NAME FOR LOAD OR UNLOAD EXCEEDS 8 CHARACTERS
12.004 UNCORRECTABLE INPUT ERROR
12.005 UNCORRECTABLE OUTPUT ERROR
12.006 ATTEMPT TO READ PAST END OF FILE
12.007 UNCORRECTABLE INPUT ERROR DURING LOAD
12.008 MODULE NOT FOUND IN LIBRARY
12.009 MODULE TO BE UNLOADED IS NOT LOADED

12.010 ATTEMPT TO REWIND SYSTEM FILE
12.011 ATTEMPT TO READ OUTPUT FILE (REWIND FILE FIRST)
12.012 ATTEMPT TO WRITE INPUT FILE (REWIND FILE FIRST)
12.013 DUPLICATION FACTOR OR TAB LOCATION IN FORMAT SPECIFICATION IS ZERO
12.014 ILLEGAL CHARACTER IN FORMAT SPECIFICATION
12.015 TOO MANY PARENTHESES IN FORMAT SPECIFICATION
12.016 TOO MANY RIGHT PARENTHESES IN FORMAT SPECIFICATION
12.017 MISSING NUMBER AFTER T FORMAT
12.018 LENGTH IN H FORMAT SPECIFICATION EXCEEDS FORMAT SPECIFICATION LENGTH
12.019 OUTPUT FORMAT MISSING INITIAL LEFT PARENTHESIS
12.020 OUTPUT FORMAT MISSING FINAL RIGHT PARENTHESIS
12.021 SYSIN RECORD EXCEEDS 80 BYTES
12.022 ERROR IN OPENING FILE FOR OUTPUT
12.023 ERROR IN OPENING FILE FOR INPUT
12.024 ATTEMPT TO PROCESS TWO MEMBERS OF THE SAME PDS AT ONCE
12.025 ATTEMPT TO PROCESS TWO FILES ON THE SAME TAPE AT ONCE
12.026 FORMAT DOES NOT CONTAIN ANY A-TYPE ITEMS
12.027 UNBALANCED QUOTES IN FORMAT SPECIFICATION LITERAL
12.028 ATTEMPT TO WRITE TO READ-ONLY FILE (SUCH AS PDS DIRECTORY OR READ-ONLY PSEUDO FILE)
12.029 ATTEMPT TO READ FROM WRITE-ONLY FILE (SUCH AS WRITE-ONLY PSEUDO FILE)
12.030 ATTEMPT TO TRACE SYSPRINT FILE
12.031 INSUFFICIENT MEMORY TO COMPLETE INTERFACE FUNCTION (RAISE R PARAMETER)
12.032 ERROR IN CLOSING FILE
12.033 ATTEMPT TO OPEN FILE WHICH IS ALREADY OPEN
12.034 INVALID STRING PASSED TO SYSOPEN
12.035 ERROR IN OPENING FILE FOR UPDATE
12.036 TWO WRITES IN A ROW TO UPDATE FILE
12.037 ATTEMPT TO USE FORTRAN FORMAT WITH UPDATE OR LENGTH OF UPDATING RECORD NOT THE SAME AS OF RECORD TO BE UPDATED
12.038 INPUT I/O ERROR ON UPDATE FILE
12.039 OUTPUT I/O ERROR ON UPDATE FILE
12.040 ATTEMPT TO REFERENCE INPUT ASSOCIATED VARIABLE AFTER FILE HAS BEEN ENDFILED BUT BEFORE VARIABLE HAS BEEN DETACHED
12.041 ATTEMPT TO READ A NON-EXISTANT PDS MEMBER
12.042 INVALID WYLBUR EDIT FORMAT
12.043 ATTEMPT TO GENERATE WYLBUR LINE NUMBER OVER 99999.999
12.044 ERROR OPENING A FILE USING WYLBUR EDIT FORMAT
12.045 INVALID DDNAME PASSED AS A SECOND ARGUMENT TO THE LOAD FUNCTION
12.046 ATTEMPT TO WRITE A RECORD LARGER THAN LRECL TO A FILE IN WHICH SPANNED RECORDS ARE PROHIBITED
12.047 ERROR ATTEMPTING TO OPEN A FILE FOR EXTEND
12.098 RECORD OUT OF RANGE IN *DA1 FILE
12.099 I/O ERROR ON *DA1 FILE
12.100 ATTENTION RECEIVED

The internal organization of SPITBOL is quite different from that in SIL SNOBOL4. Consequently the relative speed of various operations differs. This section attempts to give some idea of what is going on inside so that the SPITBOL programmer can achieve maximum efficiency.

SPACE CONSIDERATIONS

The SPAN, BREAK and BREAKX functions use translate and test tables. For the case of one character arguments, the tables are built into the system and require no additional space. For arguments longer than one character, tables must be built for each call. Each such table requires 260 bytes of storage. If the argument is deferred, no storage is required, but the execution of the pattern is much slower.

ANY and NOTANY allocate 16 byte tables (actually one bit position in a shared 256 byte table)

The space required for each element of an array is 8 bytes in addition to storage required for a STRING or other structure. All numeric items require no additional space beyond the 8 byte item.

The space required for each non-null element of a table is 24 bytes in addition to space for a STRING or other structure. A table hash header is 4 bytes. Thus the number of headers can be made reasonably large without using much additional space.

Program defined datatypes require 8(F + 1) bytes where F is the number of fields. They are thus quite compact and can be used freely.

The memory required for dynamically compiled code (code function) is not reclaimed efficiently in this version. Improvements will be attempted in future versions.

Each variable block requires 32 bytes. This space is a constant requirement whether or not the variable name has a single use or multiple uses (label,function,variable etc.). This space is never reclaimed once it has been allocated. Thus it is inefficient to use variables to build a table with the $ operator. Instead, use the TABLE datatype.

The COLLECT function is used to obtain more information on memory utilization for various structures.

SPEED CONSIDERATIONS

To a greater extent than is the case with SIL SNOBOL4, there is a loss of efficiency in encoding complex structures as strings. Use arrays, tables and program defined datatypes where possible. The latter are particularly efficient in SPITBOL.

A POS pattern may be used freely at the start of a pattern since SPITBOL optimizes this occurrence to prevent useless movements of the ANCHOR point. This optimization (which is completely transparent) occurs in both QUICKSCAN and FULLSCAN modes.

Time for datatype conversions will be relatively more noticeable in SPITBOL. Where efficiency is important, avoid unnecessary conversions.

The $ pattern assignment is, if anything, faster than the pattern assignment and may be used more freely.

SPITBOL precomputes all constant expressions before execution. When the optimize mode is on (normal case), most patterns can be precomputed, thus no efficiency is lost by writing patterns in line rather than predefining them. Use of the unary * operator to defer computation is still useful in certain cases. For example, consider these in-line pattern matches:

```
X POS(0) ARB N 'X'
X POS(0) ARB *N 'X'
```

The second form is more efficient, since the compiler can precompute the entire pattern.

Break, BREAKX and SPAN are very fast, except that deferred arguments having more than one character are quite slow. ARBNO is quite slow.

ARB is slow and should be avoided where possible.

The actual matching process is much faster in FULLSCAN mode than in QUICKSCAN mode since the heuristics require time consuming tests. If a match does not back up much, FULLSCAN may well be faster.

The process of obtaining the value of &LASTNO or &STNO is reasonably fast.

The SETEXIT error intercepts are fast and may be used for program control as well as debugging.

If a variable is traced or I/O associated, references to the variable are substantially slowed down even if the trace and I/O associations are later removed.

The unary $ (indirect) operator applied to a STRING argument works differently in SPITBOL and corresponds to a hash search of existing variables. The process of applying $ to a NAME (including the name of a natural variable) is much faster, which is why SPITBOL returns a NAME instead of a STRING when the unary dot (NAME) operator is used with a natural variable. Thus it is better to use names where possible, for example in passing labels indirectly.

The REPLACE function is optimized when the second argument is &ALPHABET. In this case, the third argument can be used as a translate table directly, and there is no need to construct a table dynamically. The REPLACE function itself can be used to construct the necessary third argument. Thus the call:

```
A = REPLACE(X,Y,Z)
```

May be replaced by the two calls:

```
   TBL = REPLACE(&ALPHABET,Y,Z)
   A   = REPLACE(X,&ALPHABET,TBL)
```

The first of these calls is slow and need only appear once. The second call is fast and could be executed repeatedly for various values of X.

PART TWO--HOW TO RUN A SPITBOL PROGRAM

This section discusses the components required for successfully running a SPITBOL program. By carefully reading this section, you should be able to run a SPITBOL program on your system.

STANDARD BATCH SPITBOL

To use SPITBOL as a standard batch program use the following Job Control Language (JCL) statements.

```
//xxxx JOB (yyyy,...)
//  EXEC   PGM=SPITBOL,PARM='...'
//STEPLIB  DD  DSN=SYS1.SPITBOL.LOAD,DISP=SHR
//SYSPUNCH DD  SYSOUT=B
//SYSPRINT DD  SYSOUT=A
//SYSIN    DD  *
- SPITBOL program goes here-
END
- data (if any) goes here-
/*
//
```

Note: The load library containing SPITBOL is installation dependent; check with your systems support group.

INTERACTIVE SPITBOL

SPITBOL may also be executed in the foreground using TSO by using the following TSO commands.

```
ALLOC F(SYSIN)    DA(program-filename)
ALLOC F(SYSPUNCH) DA(*)
ALLOC F(SYSPRINT) DA(*)
CALL  'SYS1.SPITBOL.LOAD(SPITBOL)'  '...parms...'
```

When an input file, such as SYSIN, is allocated to a terminal, EOF is indicated by entering '/*'.

SYSTEM FILES

There are required and optional datasets needed for the operation of SPITBOL under OS.

REQUIRED DATASETS

The following datasets are required for proper execution of SPITBOL under OS.

SYSIN

This file contains the source images for the source program. The LRECL on this dataset must not exceed 80 (84 for V format records). In addition, any records following the END line of the program can be accessed by the standard input variable 'INPUT'.

Multiple programs can be batched together by using './*' statements as delimiters. The text of the delimiter string is an installation option; check with your systems support group.

SYSPRINT

This data set is used for printed output including listing of the source program, error messages and trace output. In addition, the standard output variable 'OUTPUT' is associated with the file SYSPRINT. SYSPRINT may be defined with any convenient RECFM and LRECL, except that LRECL=1 is not permitted.

SYSPUNCH

The output variable 'PUNCH' is associated to the file SYSPUNCH by default. Normally this file is used for punched output and is defined accordingly with LRECL=80 (LRECL =84 for V type records).

OPTIONAL DATASETS

The following datasets are optional for execution of SPITBOL under OS.

SYSOBJ

If a DD statement is supplied for this file, then SPITBOL will generate an object module for each program compiled. (IDR data is generated on the END record of an object module.) See "Linking and Execution of Object Modules" on page 75 for information on how to link edit and run compiled programs. If no object module is required, then the DD statement for this file should be omitted.

DEFAULT DCB PARAMETERS

SPITBOL supplies the following DCB parameters if they are omitted:

FILE	DEFAULT DCB
SYSIN	RECFM=FB,LRECL=80,BLKSIZE=400
SYSPRINT	RECFM=VBA,LRECL=137,BLKSIZE=600
SYSPUNCH	RECFM=VB,LRECL=84,BLKSIZE=172
SYSOBJ	RECFM=FB,LRECL=80,BLKSIZE=400
ASP CTC FILES	RECFM=FB,LRECL=133,BLKSIZE=1995
OTHER FILES	RECFM=VBS,LRECL=2004,BLKSIZE=400

If the RECFM is supplied, then LRECL and BLKSIZE must also be supplied. LRECL and BLKSIZE may be overridden separately. Additional DCB parameters such as OPTCD, BUFNO may be supplied on the DD statement as required. Note that certain systems have special requirements for SYSPRINT DCB's.

PARAMETERS TO THE SPITBOL COMPILER

This section discusses the parameters to the SPITBOL compiler.

The PARM parameter on the EXEC statement may be used to set parameters for the run. The following table lists all parameters, their meanings, and default values. Note that the default values can be altered during installation; check with your systems support group.

NAME=DEFAULT	MEANING
B=5	Number of TSO attentions to be handled by SPITBOL. Each time an attention is received, the B value is decremented by one. When this value reaches 0, attention handling is disabled.

C=100000 Maximum number of punched cards.

D=10 Maximum number of dumps. This is decremented by N each time DUMP(N) is executed or at end- of-job if a dump is requested. When it reaches zero, no dumps are given. If DUMP=3 and D=0, a SYSUDUMP will be given if a //SYSUDUMP DD statement is provided.

E=0 Parameter for future use.

F=0 Maximum number of file trace lines. This parm is used in conjunction with the new external function SYSTRACE which is explained in the section entitled External Functions.

H=1000K Maximum size for allocated data area to be used by SPITBOL. This can be set lower to keep SPITBOL from immediately grabbing all available core, and is useful in Multi-Tasking situations. Note that the effect can also be obtained by the use of the R parameter. The size of the allocated data area (ADA) in general is equal to MINIMUM(H,REGION SIZE)-R.

I=0 Value controlling compiler listings. The low order 3 bits of this value are interpreted as follows:

1 (bit 31) -- suppress header pages
2 (bit 30) -- suppress compilation statistics
4 (bit 29) -- suppress execution statistics

For example, I=7 will suppress header pages and all statistics.

M=0 Parameter for future use.

N=58 Number of lines per page.

P=100000 Maximum number of pages.

R=18K Amount of core to be reserved to the system. Both OS and SPITBOL's OS interface GETMAIN space out of the reserved area for things like buffers, work areas, control blocks and room for loaded modules (i.e., external functions). 18K is sufficient for MVT; higher values (50K) are necessary for MVS. Raise the value when processing many files, using files with very large BLKSIZEs, using large external functions, or using external functions which GETMAIN large amounts of core. Signals to raise the R parameter include S80A abends and error code 12.31 (insufficient space to complete interface function.)

S=0 Maximum number of system trace lines. If S is greater than zero, system trace messages are written on SYSPRINT and S is decreased by one until it reaches zero, at which point tracing stops. Trace messages are written any time a function is loaded or unloaded, any time a file is opened, and any time a file is closed (The close message includes a count of the number of records read or written.)

T=55 Time limit in seconds. Note that this value should always be slightly lower than the time estimate on the OS job statement. This difference in time values allows SPITBOL to regain control should a program exceed its stated time limit. SPITBOL can then clean up and produce a dump and/or other diagnostic information allowing determination of

what has happened. For example, a program goes into an infinite loop.

In a batched run, the parameters T,P,C are applied separately to each program in the batch. This prevents any single program from overrunning system limits and aborting the run.

Any or all of these seven parameters may be altered in the PARM field. This value may be expresses either as an integer, or in units of 1024 by using a K following an integer. For example:

```
PARM='R=20000,H=16K,T=30'
```

The above parm specification would reserve 20000 bytes for system use, 16384 bytes for minimum dynamic memory, and allow up to 30 seconds of CPU time. The parameters may occur in any order. For parameters not specified, the appropriate default from the above table is used.

EXECUTION PARAMETERS FOR THE SPITBOL PROGRAM

All characters following the first slash in the PARM field of the EXEC statement are defined as the user parameter string. This string can be obtained by an executing program by calling the external function UPARM.

Since UPARM will typically be called only once, the entire process of loading it, calling it, and unloading it can be done in a single statement. (Remember, both LOAD and UNLOAD return the null string.)

```
PARMS = LOAD('UPARM()STRING') UPARM() UNLOAD('UPARM')
```

For more information, see the UPARM entry in the section see "External Functions" on page 89.

ALTERNATE DDNAMES

This information is only useful when invoking SPITBOL from another program.

Upon entry to SPITBOL from the calling program, register one can be zero or point to a variable length parameter list in standard OS format. (eg high-order bit in the last address is turned on to indicate end of the list). The first entry in the list is the address of the parameter list. The second entry, if supplied, points to an alternate DDNAME list. In normal invocation via JCL, the first address wil be set to point to the text supplied for the PARM= keyword of the EXEC statement preceded by a half word length count by the system.

The alternate DDNAME list, if supplied, is a halfword which defines the length of the list, followed by the alternate DDNAME. The first 96 bytes define alternates for the following DDNAMES:

SYSLMOD
-not used-
SYSLIN
SYSLIB
SYSIN
SYSPRINT
SYSPUNCH
SYSUT1
SYSUT2
SYSUT3
SYSUT4
SYSTERM

Entries which indicate no alternate should be coded as 8X'00'.(zeros) after the first 96 bytes any number of additional 16-byte entries consisting of 8-byte DDNAMES followed by 8-byte alternates. This permits an alternate DDNAME to be supplied for any DDNAME.

USER ABEND CODES

The following user abend codes can be issued.

CODE	MEANING
100	Internal error abort. Note: This abend is also generated when an external function program checks.
200	Insufficient dynamic memory, increase region
300	Permanent output error on SYSPRINT
400	Missing DD statement for system file
500	Error on opening system file
800	Internal configuration mismatch in OS interface and transient module OSINIT.
1001	Stack overflow in OS interface.
1001	Stack underflow in OS interface.

SYSTEM ABEND CODES

The following system abend codes can be issued.

CODE	MEANING
80A	Insufficient memory reserved for system use. Increase value of R parameter.

LINKING AND EXECUTION OF OBJECT MODULES

If a DD statement for SYSOBJ is supplied to SPITBOL, an object module for the compiled program will be written to that file. (The program will still be executed as well!) This facility provides a way to avoid recompilation of frequently run programs and a mechanism for distribution of SPITBOL programs. However, in general, compilation is so fast that it is simpler to just recompile for every run.

After creation of the object module, it must be link edited with the SPITBOL library routines to produce an executable load module. These library routines are contained in load module SPITPROG of the SPITBOL load library. SPITPROG is similar in structure to SPITBOL, but excludes routines needed only for compilation. The sample job below shows how to create a load module for a previously compiled SPITBOL program. (The OS Loader can also be used.)

```
//xxxx JOB (yyyy,...)
// EXEC LKED,PARM='LET,LIST,NCAL,TERM,XREF'
//SYSLMOD  DD  DSN=your-load-library,DISP=SHR
//SPITLIB  DD  DSN=SYS1.SPITBOL.LOAD,DISP=SHR
//OBJECT   DD  DSN=your-object-module,DISP=SHR
//SYSIN    DD  *
 INCLUDE SPITLIB(SPITPROG)
 INCLUDE OBJECT
 ENTRY OSINT
 NAME progname(R)
/*
//
```

Note: Only a single program's object module can be linked with the library routines. Thus, it is not possible to use the linkage editor (or loader) to combine multiple SPITBOL programs. Communication between a SPITBOL program and any other program must be through the interface defined for external functions.

When the resulting load module is executed, DD statements and and PARM field values must be supplied as for a regular compile and execute run. However, since the program has already been compiled, the dynamic memory obtained is used only for objects created during the program's execution. The SYSIN DD statement should point to a file containing records to be accessed via standard input; i.e., variable "INPUT". The following job shows how to execute a load module.

```
//xxxx JOB (yyyy,...)
// EXEC PGM=progname,PARM='...'
//STEPLIB  DD  DSN=your-load-library,DISP=SHR
//         DD  DSN=SYS1.SPITBOL.LOAD,DISP=SHR
//SYSPRINT DD  SYSOUT=A
//SYSPUNCH DD  SYSOUT=B
//SYSIN    DD  *
- data (if any) goes here-
/*
//
```

The sample job above has the SPITBOL load library available during execution of the load module. However, this can be inconvenient to many users and is not necessary in all cases. To determine whether or not the SPITBOL load library is required during execution of a load module, use the following checklist:

1. Does your site have non-resident initialization? (I.e., is module OSINIT loaded to initialize SPITBOL?).

2. Does your program use LOAD to access external functions?

3. Does your program use BDAM, ISAM, or VTOC support?

If the answer to any of the questions above is yes, then the SPITBOL load library is required during the execution of that load module.

Another more empirical approach is run the program with system tracing enabled; entries in the system trace log will list any modules loaded, whether explicitly by the program or implicitly by the OS interface. See the S parameter in "Parameters to the SPITBOL Compiler" on page 71.

This section discusses the facilities available for input/output. Information in this section augments the descriptions for the INPUT and OUTPUT functions.

RECORD FORMAT SUPPORT

For variable length record formats, the ouput STRING is written as a single record of appropriate length if possible. If the length of the STRING exceeds the LRECL, then the STRING is split into several records as required. An over-size record on input causes an error. The null STRING is written and read as a one byte record consisting of the character X'00' (hexadecimal zero). Spanned records are implemented. However, the input LRECL should not be too much larger than required, since SPITBOL must temporarily find a buffer of length LRECL on input.

For the fixed length record formats, the output STRING is split into several logical records if its length exceeds the specified LRECL. The last, or only, record written is padded with blanks. This means that on input, extra blanks may be read. A null value is written as a blank record.

For undefined records, the STRING is written as a block if possible, or split up if necessary. The null STRING is handled as for variable length records.

It should be clear that variable length record formats are preferable for the input and output of SNOBOL4 strings, the F formats are implemented primarily for compatibility with other OS processors.

Note: SPITBOL ignores the A (ASA control characters) specification. If control characters are to be generated, the output association should specify an appropriate format. The standard association for SYSPRINT specifies blank control characters.

INPUT OUTPUT ASSOCIATION--DDNAMES

The standard mechanism for accessing a file is to provide the DDNAME as the second argument to an INPUT or OUTPUT function call. For example:

INPUT(.IN,'MYFILE') input-associates the variable IN to the dataset defined by the DD statement with DDNAME MYFILE.

OUTPUT(.OUT,'PUTOUT') output-associates the variable OUT to the dataset defined by the DD statement with the DDNAME PUTOUT.

Remember that a file is opened at the time of the first read or write, NOT when INPUT or OUTPUT is called.

PDS MEMBER SUPPORT

To input or output associate a PDS member, specify DDNAME(MEMBER) as the second argument to INPUT or OUTPUT. The member name can, of course, be specified at run-time. One restriction applies to accessing PDS members: only one member of a PDS can be open for writing at a time. Thus, ENDFILE must be used to close out a member before opening another member for writing. There is no restriction on the number of members open for reading.

For example, given this DD statement

```
//MYPDS DD DSN=ANY.OLD.PDS,DISP=SHR
```

Member FOOBAR can be read by

```
INPUT(.IN,'MYPDS(FOOBAR)')
```

When processing is completed for member FOOBAR, the file can be closed by

```
ENDFILE('MYPDS(FOOBAR)')
```

The member name can be specified dynamically at execution time as in

```
MEMBER = 'FOOBAR'
INPUT(.READ,'MYPDS(' MEMBER ')')
...
ENDFILE('MYPDS(' MEMBER ')')
```

Issuing an ENDFILE for a PDS member after completion of its processing is considered good practice.

PDS DIRECTORY SUPPORT

SPITBOL supports reading directory entries of a PDS; writing is not supported. (Other PDS directory functions are provided by the external function "SYSDIR" on page 92) The format of the second argument to the INPUT function is similar to that of specifying a PDS member, except that no member is supplied; only the parentheses. For example,

```
INPUT (.DIRENTRY,'MYPDS()')
```

specifies that each access of variable DIRENTRY causes the next directory entry of MYPDS to be read.

PDS directory entries are of variable length between 12 and 74 bytes and contain the 8-byte member name, followed by a 4-byte TTRC address, followed by any user data which may be present in the directory entry. The last record of the directory is 16 bytes long and contains eight bytes of X'FF' followed by two four-byte numeric strings. Eight bytes of X'FF" are easily generated by DUPL(SUBSTR(&ALPHABET,256,1),8). The two numeric strings give the total number of directory blocks in the directory and the number which are unused respectively. The next reference to the variable which is input associated to the directory will fail, just like any other input-associated variable when end-of-file is reached.

For example, to read the directory of the PDS in the example above:

```
      LASTENTRY = DUPL(SUBSTR(&ALPHABET,256,1),8)
      INPUT(.DIRENTRY,'MYPDS()')
LOOP  ENTRY  = DIRENTRY                                  :F(EXIT)
      MEMBER =  SUBSTR(ENTRY,1,8)
      LEQ(MEMBER,LASTENTRY)                              :S(TOTAL)
      OUTPUT = ENTRY                                     :(LOOP)
TOTAL OUTPUT = 'OUT OF ' SUBSTR(ENTRY,9,4)
+              ' BLOCKS, ' SUBSTR(ENTRY,13,4) ' ARE FREE'
```

MULTIPLE FILE TAPE SUPPORT

SPITBOL supports the specification of tape file numbers at run time. File numbers are specified in the same manner as PDS members, appearing within parentheses following the DDNAME. The DD statement for the tape must specify no labels, i.e.

```
LABEL=(,NL)   or   LABEL=(,BLP)
```

To process a standard label (SL) tape, specify the tape as NL in the JCL and then request the file numbered 3N-1. File 3N-1 on a NL tape is equivalent to file N on a labeled tape. Note that tape labels can be read from or written to a SL tape by simply ignoring the 3N-1 rule. (After writing such a tape, it could be read as a SL tape.) Only one file on a given tape can be open at a time. For example, having the following DD statement:

```
//MYTAPE DD UNIT=9TRACK,LABEL=(,NL)
```

The third and fourth files will be written in the following manner.

```
        OUTPUT(.TAPEOUT,'MYTAPE(3)')
*
*       ... WRITE TAPE FILE USING VARIABLE TAPEOUT ...
*
        ENDFILE('MYTAPE(3)')
*
        OUTPUT(.TAPEOUT2,'MYTAPE(4)')
*
*       ... NOW WRITE FOURTH FILE ON TAPE USING TAPEOUT2 ...
*
```

DIRECT ACCESS FILE (BDAM) SUPPORT

In SPITBOL 370 support has been added to access files of fixed-length records randomly as well as sequentially; that is to say, specific blocks from the file can be read or written at will.

Random access of fixed-length files is accomplished through the use of the *DA1 pseudo-file and its associated external functions. The block to be read or written is identified by its block number, an integer ranging from zero for the first block sequentially up to the number of the last block in the file.

Before using a random-access file the file must be formatted. There are two ways to format a file. The first is to write the file as a standard SPITBOL sequential file with RECFM=F and the LRECL equal to the BLKSIZE. The second way, which is most suitable for files that will then be loaded in random order, is to use the new external function SYSFMT. This function is loaded via:

```
LOAD('SYSFMT(STRING)INTEGER')
```

and is invoked as

```
SYSFMT('*DA1(ddname)')
```

where ddname is the OS JCL ddname of the file to be formatted. The entire file will be filled with blocks of binary zeroes. The value returned by SYSFMT is the number of blocks in the file it has just formatted. Note that if a file is simply to be initialized with zero blocks, SYSFMT is generally faster that writing out the zero blocks using a standard SPITBOL sequential file. Note that if SYSFMT is used, LRECL and BLKSIZE must be specified on the DD card defining the ddname to be formatted.

Once the file is formatted, it can be read or written by making standard I/O associations to the pseudo-file *DA1(ddname). The block number to be read or written is specified through the SYSRECNO function that is described below. If the intent is to both read and write the file, it should be opened for UPDATE using the SYSOPEN function (e.g. SYSOPEN('*DA1(ddname)','U')) after which variables can be input and output associated to it in the usual fashion. See "SYSOPEN" on page 93.

Note that opening a *DA1 pseudo-file for update does not mean the same thing as opening a normal sequential file for update in place. For a *DA1 pseudo-file, reads and writes can occur and be intermixed in any order; there is no requirement that each write be proceeded by a read as in sequential update-in-place.

Note also that when writing blocks SPITBOL will provide blank padding up to the required LRECL (or BLKSIZE, which is the same in the case of *DA1 pseudo-files) but that records exceeding the LRECL cannot be written to a *DA1 pseudo-file (this will cause a 12.046 error code to be generated).

Associated with each *DA1 pseudo-file is a record number, which defines the next record to be read or written. This record number is initialized to zero when the file is first opened and can be reset at will using the SYSRECNO function, which is loaded by:

```
LOAD('SYSRECNO(STRING,INTEGER)STRING')
```

and called as SYSRECNO('*DA1(ddname)',blocknumber) where blocknumber gives the number of the next block to be read or written. SYSRECNO will al-

ways return the null string, and will fail if the specified file is not open or otherwise invalid. Thus, to open and read the third record of the *DA1 pseudo-file with ddname MYDATA the following program fragment could be used:

```
LOAD('SYSRECNO(STRING,INTEGER)STRING')
INPUT(.R,'*DA1(MYDATA)')
SYSOPEN('*DA1(MYDATA)','I')
RECORD3 = SYSRECNO('*DA1(MYDATA)',3) R
```

Note that the SYSOPEN must be done before the first time that SYSRECNO is called.

An attempt to read a block that does not exist (eg. is out of range) from a *DA1 pseudo-file will result in a 12.98 error code (which can be trapped through use of the SETEXIT function). An I/O error in reading a record from a *DA1 pseudo-file results in a 12.99 error.

When working with random-access pseudo-files it is occasionally useful to determine the number of records in the file. This can be done using the SYSFSIZE function. (SYSFSIZE also works on normal RECFM=F sequential files, where for an input file it gives the number of records in the file, and for an output file it gives the number of records that the file will hold). SYSFSIZE is loaded by LOAD('SYSFSIZE(STRING)INTEGER') and when called it is passed a file name. For example, a *DA1 pseudo-file is represented by *DA1(ddname). For more details see -- Heading id 'sysfsiz' unknown --.

SYSFSIZE requires that its argument file be open. SYSFSIZE will fail if the file is not open or if the file name is invalid. Note that a file is opened at the time of the first read or write operation, not at the time of input or output association. SYSOPEN can be used to force an open before the first read or write is actually done to the file. Thus, in order to find out how many records are in a direct-access file, a sequence like

```
NRECS = SYSOPEN('*DA1(ddname)') SYSFSIZE('*DA1(ddname)')
```

is appropriate.

ISAM SUPPORT

SPITBOL supports sequential reading and writing of ISAM files. ISAM file access is indicated by the form of the DDNAME to the INPUT/OUTPUT function

*ISAM(<DDNAME>)

where <DDNAME> is the DDNAME of a DD statement defining the ISAM file to be processed. For example, to input associate a variable called ISAMIN to a ISAM file defined by a DD statement with the DDNAME ISAMFILE, use

```
INPUT(.ISAMIN,'*ISAM(ISAMFILE)')
```

QSAM UPDATE-IN-PLACE SUPPORT

To process a sequential file or PDS member using update-in-place, the file must be input associated, output associated, and opened by SYSOPEN. (Refer to "SYSOPEN" on page 93 for details.) The reason for having both input and output associations is that the input association handles the reading of a record, while the output association handles the writing (updating) of a record. Note that the third argument, specifying a FORTRAN format to the INPUT and OUTPUT functions, must not be used for files being updated-in-place.

Update-in-place record processing proceeds as follows. Records are read using the input associated variable. If no changes are needed to the current record, the next record is read. However, if modification to the current record is necessary, an updated record is constructed and then assigned to the output-associated variable. This assignment causes the record to be updated-in-place. Note that a write can only follow a read.

The following restrictions apply to the modification permissible on a record which is to be written back to a file open for update:

1. For RECFM=U or V files the record to be rewritten must have the same length as the record read.

2. For RECFM=F files the record to be re-written must have length less than or equal to the file's LRECL. (this is really the same as the RECFM=U or V case because SPITBOL pads any record written to a RECFM=F file with blanks to a length equal to the file's LRECL anyway.)

If these restrictions are not satisfied, a system error code of 12.037 (invalid length) will be generated. This error code will also occur on an attempt to use variables input or output associated to the file with the third argument to the input or output function specified.

The handling of null records for RECFM=V or U is somewhat inconsistent with the way SPITBOL handles null records for input-only or output-only files. For a update file, a null record is returned as 1 byte of binary zeroes, rather than as a null STRING. The following program will reverse every other record in a file called 'UPTEST'.

```
        INPUT(.IN,'UPTEST')
        OUTPUT(.OUT,'UPTEST')
        LOAD('SYSOPEN(STRING,STRING)STRING')
        SYSOPEN('UPTEST','U')
        UNLOAD('SYSOPEN')
LOOP    REC = IN                                :F(END)
        SW  = IDENT(SW) 1                       :S(LOOP)
        OUT = REVERSE(REC)
        SW  =                                   :(LOOP)
END
```

PSEUDO FILE SUPPORT

A DDNAME with bit 1 of byte 0 off designates a pseudo DDNAME. This bit was chosen because it is on for all alphabetic, national and numeric characters. Pseudo files allow a SPITBOL user to perform basic I/O operations on other than sequential QSAM files with a DD statement supplied. Examples include support of an access method other than QSAM or us of "IN-CORE" I/O. The type of processing desired is determined by interpreting the pseudo file DD name as a series of special fields.

There are 3 sources for a pseudo DDNAME:

1. Supplied in the alternate DDNAME list.

2. Supplied by tables within OSINT.

3. A user-supplied DDNAME passed to an input or output association.

A pseudo DDNAME has one of the following formats:

1. FLAGS(1)

   NAME1(3)
   DWORDS(1)
   NAME2(3)

2. FLAGS(1)

   NAME1(3)
   DWORDS(1)
   RECFM(1)
   LRECL(2)

The two forms are distinguished by the flag bits in the first byte. These bits, and their meanings, are:

X'80' If on, an alternate DDNAME is used only if the old DDNAME is not in the task I/O table. This allows pseudo- DDNAMES which will be used only if no DD statement exists for the main DDNAME.

X'40' If on, this is a pseudo DDNAME.

X'20' If on, the NAME1 and the NAME2 (if applicable) fields are considered as core addresses. If off, they are concatenated to "OSINT" and treated as module names.

X'10' If on, this indicates that this is a format 4 pseudo DDNAME.

X'08' If on, this indicates that this is an input only file.

X'04' If on, this indicates that this is an outout only file.

DWORDS gives the number of additional double words to be appended to the QSAM DCB for work areas, etc.

For a format1 DDNAME, NAME1 determines the module name or core address of a routine which will be called to do the open for the file. NAME2 determines the module name or core address of a routine which will do the close for the file. Linkage conventions are:

Register 1 Points to an OS open or close list containing 1 DCB address.

Register 13 Points to a save area.

Register 14 Contains the return address.

Register 15 Contains the entry point address.

The open routine can modify the DCB as desired and should set the address of a GET or PUT routine into the DCB. The DCBOFLGS bit X'10' must be set on to indicate a successful open.

For a format 2 pseudo DDNAME, NAME1 determines the module name or core address of a GET or PUT routine to be placed in the DCB. The specified LRECL and RECFM are also set in the DCB. For update files being defined as format 2 pseudo DDNAMES the first call to the user GET routine should result in the DCB as well as any other desired processing.

The linkage conventions to all types of GET and PUT routines are as follows:

```
Register 0 Points to the record or a buffer
Register 1 Points to the DCB and user doublewords
Register 13 Points to a save area
Register 14 Is the return point
Register 15 Is the entry point
```

For output:

```
RECFM=U LRECL is in DCB
RECFM=V LRECL is in RDW
RECFM=F The record is blank padded to length LRECL
```

For input:

```
RECFM=U LRECL in DCB must be set
RECFM=V An RDW must be built
RECFM=F The record must be padded to length LRECL
```

The GET routine can signal end of file by loading DCBEODAD into R14 and branching to it. The GET or PUT routine can signal an I/O error by loading DCBSYNAD into R14 and branching to it.

The routine can issue any 12.Xxx error by loading the minor code into register zero and then branching to the contents of DCBSYNAD minus 4.

Note: Before attempting to write pseudo-file support routines, the implementor should review the existing pseudo-file support routines provided with SPITBOL.

VTOC SUPPORT

A system pseudo-file has been implemented to allow the SPITBOL user to read disk VTOCs quickly and efficiently. This is based on a VTOC-reading routine which employs EXCP-level coding to read the VTOC with 1 EXCP per track. To use this facility:

Include a DD statement of the form:

```
//<DDNAME>  DD  UNIT=....,VOL=SER=.....,DISP=OLD
```

To read the VTOC of a volume, input-associate a variable to the VTOC as follows:

```
INPUT(.<VARNAME>,'*VTOC(<DDNAME>)')
```

References to the variable <VARNAME> will result in DSCBs from the VTOC (sequentially, starting from the first) in the format:

Key	44 bytes
Data	96 bytes
MBBCCHHR	8 bytes

Format-0 DSCBs are not returned, but are bypassed by the pseudo-file read module. A reference to the variable will fail at end-of-file; an I/O error is treated just as an I/O error on any other input file. The R parameter should be increased by about 14K for the first VTOC to be read and about 9K for each additional VTOC being read simultaneously (for VTOCs on 3330 disks; other device types are supported, the rule being in general to allow about 5K+ (148*# DSCBs/TRK for the device) for the first VTOC and about (148*# DSCBs/TRK) for additional VTOCs being processed simultaneously.) If system tracing is enabled, when the pseudo-file is closed the read count gives the number of calls to the variable input-associated with the VTOC (eg the number of DSCBs returned to the calling SPITBOL program) and the GET count gives the number of EXCP's issued by the VTOC reading routines themselves. (Note: if a VTOC is completely processed, the GET count should be equal to the number of tracks in the VTOC, since the VTOC reading routines read the VTOC a track at a time).

Note: This function has not been fully tested on MVS; it is not likely to work properly with indexed VTOCs.

CONSOLE SUPPORT

Doing an OUTPUT(.CONSOUT,'CONSOLEO') will, if there is no DD statement for DDNAME CONSOLEO, output-associate the variable CONSOUT to the OS operator's console (via WTO).

Console input association is done by associating to the DDNAME CONSOLEI without providing a DD statement for that dataset. Each reference to the input associated variable will return the text of the next OS MODIFY command, i.e. the <TEXT> of the OS operator command:

```
F <JOBNAME>,<TEXT>
```

directed to the job. End of file is indicated to the program by the STOP command, i.e.:

```
   P <JOBNAME>
```

Each reference to the input-associated variable causes the program to wait for a STOP or MODIFY command to be entered at the console. This wait can be avoided by using the CONSOLEA file as follows:

Doing an INPUT(.<VARIABLE>,'CONSOLEA') with no DD statement for CONSOLEA will cause any reference to the variable <VARIABLE> to:

```
Fail if no STOP or MODIFY command is available
Return the value 'F' if a MODIFY is available
Return the value 'P' if a STOP is available
```

WYLBUR EDIT FORMAT FILE SUPPORT

Support has been added to SPITBOL to read and write WYLBUR EDIT format files. This requires no special effort on the part of the user program as long as valid WYLBUR EDIT format files are provided as input or output. However, as part of this support, the following features have been added.

Note: WYLBUR EDIT file support is an installation option; check with your systems support group.

Extensions to SYSOPEN

WYLBUR EDIT files can be opened either explicitly or implicitly. Inserting a 'W' at the front of a valid option string specifies an explicit open of a WYLBUR file. For example, SYSOPEN('MYFILE','WI'). Additionally, any disk file with the following DCB attributes will be implicitly opened as a WYLBUR EDIT file:

RECFM=U

LRECL=BLKSIZE

BLKSIZE between 3156 and 7000

Inserting an 'N' before a valid option string overrides the implicit opening of a WYLBUR EDIT file. This means that the file will NOT be opened as a WYLBUR EDIT file, even if its DCB characteristics match those above.

External Functions for Use with WYLBUR EDIT Format Files

External functions are provided to support WYLBUR EDIT format files. SYSDELTA changes the DELTA for line numbers of lines written to a WYLBUR file. SYSLINEI returns the line number of the last line read from a WYLBUR file. SYSLINEO sets the line number for the next line to be written to a WYLBUR file. For additional information, see the section "External Functions" on page 89.

TSO FACILITIES

The following facilities are available to a SPITBOL program executing under TSO.

Attention Handling

SPITBOL 370 under TSO provides attention handling. In order to provide this support, the following features have been added.

B Parameter

The B parameter specifies the number of attentions SPITBOL will handle. Each time an attention is received, the B parameter is decremented by one, and, when the count becomes zero, attention handling is turned off. See "Execution Parameters for the SPITBOL Program" on page 73.

Types of Attention Handling

SPITBOL 370 provides two types of attention handling: IMMEDIATE and DEFERRED. In IMMEDIATE mode, each time an attention is received, a execution error 12.100 is generated. This error may be trapped by means of the SETEXIT function, but, if not trapped, causes the SPITBOL program to terminate. In DEFERRED mode, the value of the B parameter is still decremented by one each time an attention is received, but the user program is not notified of any attentions until it specifically requests whether any have been received. This allows the user program to continue executing until it gets to a point at which it is ready to field attentions.

The default for attention handling is IMMEDIATE mode.

External Functions for Attention Handling

Two new external functions have been provided. Refer to "Execution Parameters for the SPITBOL Program" on page 73 for aditional background information.

SYSATNST LOAD('SYSATNST(STRING)STRING')

SYSATNST('x') sets the attention handling state depending on the value of 'x'. An 'I' sets the state to IMMEDIATE mode and a 'D' sets the state to DEFERRED mode. The function returns the null string.

SYSATNCK LOAD('SYSATNCK(STRING)STRING')

SYSATNCK() is called to check if any attentions have been received while the program has been running in DEFERRED mode. If any attentions were received, the function succeeds and

returns the null string. If no attentions were received, the function fails. The function also fails if called while in IMMEDIATE mode.

TSO Terminal I/O External Functions

The following new external functions have been implemented:

TCONV — LOAD('TCONV(STRING)STRING')

TCONV(STRING) is equivalent to a TPUT(STRING) followed by a TGET(). Thus, TCONV issues a prompt before reading a line from the terminal.

TCONV0 — TCONV0 is equivalent to a TPUT(STRING) followed by a TGET0(). Thus, TCONV0 issues a prompt before reading a line from the terminal. The function will fail if a null string is entered by the user.

TGET — LOAD('TGET(STRING)STRING')

TGET() reads a string from the terminal without issuing a prompt. The string read from the terminal is returned exactly as entered, and is not converted in any way. TGET will automatically retry most TSO terminal I/O error conditions.

TGET0 — TGET0 is similar to TGET except the function will fail if a NULL string is entered by the user.

TPUT — LOAD('TPUT(STRING)STRING')

TPUT(STRING) writes its argument string to the terminal exactly as given. Note that if a carriage return is desired at the end of the line, it must be appended to the argument string. The function normally returns a NULL string and will fail under certain error conditions such as an I/O error.

Note: These functions are dependent on the TCAM translate tables. They function as transparently as possible, but if problems occur consult your systems support group.

This section discusses the calling conventions for writing external functions, as well as use of the external functions provided with the SPITBOL 370 distribution.

CONVENTIONS FOR EXTERNAL FUNCTIONS

This section gives detailed information necessary for writing external functions in FORTRAN and Assembly Language for use in SPITBOL. To a great extent, compatibility with SIL SNOBOL4 is maintained but there are some differences.

An external function exists as a load module in one of the standard job libraries (system link library list, JOBLIB or STEPLIB, or a private load library). Most usually, STEPLIB is pointed to a priva te library containing the module. Concatenation may be used to introduce more than one library if required. Note carefully that the module name is the same as the entry name and the function name.

The function is introduced by means of the LOAD system function which is compatible with that supplied in SIL SNOBOL4 with minor exceptions:

```
LOAD('FNAME(ARG 1,ARG 2,...,ARG N)RESULT', 'LIBRARY')
```

FNAME Is the function (and module) name. It must not be longer than 8 characters (there is no truncation of longer names as in SIL SNOBOL4)

ARGs Are the argument datatypes which may be STRING, INTEGER, REAL, DREAL. Any other entry (including null) means that no conversion takes place. If one of these four special entries is used the corresponding argument is converted to the indicated datatype and passed in special external form.

RESULT Is the result type specified in a similar manner. It may be omitted if no conversion is required.

LIBRARY Is the DDNAME of the private load library. If null, the standard job libraries are searched.

The call to the function obeys standard OS conventions:

Register 1 Points to the parameter list

Register 13 Points to save area

Register 14 Return address

Register 15 Function entry address

Note: Register 8 points to the SPITBOL data area for use by functions which interact in an intimate way with the SPITBOL system.

In accordance with OS standards, any registers used must be saved and restored. (SIL SNOBOL4 allows destruction of registers).

Parameters are given as addresses of double word quantities as follows:

STRING	Word 1 Starting address
	Word 2 Length in bytes
INTEGER	Word 1 Integer value
	Word 2 Unused
REAL	Word 1 Real value
	Word 2 Unused
DREAL	Words 1, 2 Long form REAL value
UNCONVERTED	Words 1, 2 Standard SPITBOL specifier

It should be noted that the form of unconverted arguments is totally incompatible with SIL SNOBOL4. Strings are not aligned on any given boundary. Note that the converted forms of numeric data are suitable for use in a call to a FORTRAN function.

The result is returned as follows:

INTEGER	Register 0 has integer value
REAL, DREAL	Floating point Register 0 has REAL or long REAL value
STRING	Register 0 points to two word block with Word 1 = Address, Word 2 = length in bytes.
UNCONVERTED	Register 0 points to eight byte SPITBOL value specifier

String results are always copied to free core on return and may therefore be built inside the function.

There are three ways to return from an external function: success, failure, or error. For a successful return use

```
   BR     R14                    SUCCESS RETURN
```

For a failure return use

```
   B      4(,R14)                FAILURE RETURN
```

For an error return use

```
   LR     R0,=AL2(mm,nn)         MAJOR CODE = MM, MINOR CODE = nn
   B      8(,R14)                TAKE ERROR RETURN
```

Where MAJOR and MINOR are the error codes to be generated.

Special external data types may be introduced as special byte specifiers passed unconverted with the first byte set to one of the following values:

X'20', X'22', X'24', X'26', X'28', X'2A', X'2C', X'2E'

Such external data only has significance to external functions which recognize it.

The SPIE active is one which generates a dump if an interrupt occurs which is unknown to SPITBOL. SPITBOL does not use a stae. There is a STIMER outstanding.

The program mask is all zeros on entry and must be all zeros on exit

The call UNLOAD(.FNAME) has the effect in SPITBOL of undefining the function FNAME for any kind of function. SPITBOL automatically removes an external function from storage if all functions referring to it become undefined (or redefined). Thus in the normal case, UNLOAD is compatible with SIL SNOBOL4, but it also works in complex uses of OPSYN and function redefinition, avoiding bugs which are currently present in SIL SNOBOL4.

AVAILABLE EXTERNAL FUNCTIONS

The following external function are supplied on the standard SPITBOL distribution tape.

SYSATNCK

```
LOAD('SYSATNCK(STRING)STRING')
```

SYSATNCK() is called to check if any attentions have been received while the program has been running in DEFERRED mode. If any attentions were received, the function succeeds and returns the null string. If no attentions were received, the function fails. The function also fails if called while in IMMEDIATE mode.

SYSATNST

```
LOAD('SYSATNST(STRING)STRING')
```

SYSATNST('x') sets the attention handling state depending on the value of 'x'. An 'I' sets the state to IMMEDIATE mode and a 'D' sets the state to DEFERRED mode. The function returns the null string.

SYSDATE

```
LOAD('SYSDATE(STRING)STRING')
```

The SYSDATE function returns a string containing the Julian date, the time of day, and the Gregorian date. The format of the date string is

```
mm/dd/yy hh.mm.ss ddd.yy
```

Note: The format of the string returned by SYSDATE differs from the previous implementation.

SYSDELTA

```
LOAD('SYSDELTA(STRING,INTEGER)STRING')
```

SYSDELTA(DDNAME,DELTA) changes the DELTA for line numbers written to a WYLBUR EDIT file. The first argument specifies the DDNAME of the WYLBUR EDIT file. The second argument specifies the new DELTA value. Note that DELTA is an integer value equal to the WYLBUR line number multiplied by 1000.

The function fails if the specified file is not output associated or if DELTA is greater than 9999999. The function returns the null string.

SYSDIR

```
LOAD('SYSDIR(STRING,STRING,STRING,STRING)STRING')
```

SYSDIR provides general PDS directory manipulation facilities. The operands to SYSDIR are as follows:

```
SYSDIR(ddname,membername,opcode,data)
```

The first three arguments must always be supplied. The inclusion of the fourth argument depends on the opcode. The supported opcodes are:

D Delete member. The data argument is ignored.
A Add alias. The alias to be added is specified by the data argument.
C Replace user data for member. The new user data is specified by the data argument.
R Rename member. The new member name is specified by the data argument.

SYSFEOV

```
LOAD('SYSFEOV(STRING)STRING')
```

SYSFEOV(DDNAME) forces end of volume on the current volume of a multiple volume file. The file's DDNAME must be provided as the string argument. An error will occur if called when the current volume is the last volume of a file. The function returns the null string.

SYSFSIZE

```
LOAD('SYSFSIZE(STRING)INTEGER')
```

SYSFSIZE(DDNAME) returns the number of records in the dataset. There are a number of restrictions on the use of this function; see the section "Pseudo File Support" on page 83 for details.

SYSLINEI

LOAD('SYSLINEI(STRING)INTEGER')

SYSLINEI(DDNAME) returns the integer line number of the last line read from the specified WYLBUR EDIT file. The first argument specified the DDNAME of the WYLBUR EDIT file. Note that the value returned is is an integer equal to the last line number read multiplied by 1000.

SYSLINEO

LOAD('SYSLINEO(STRING,INTEGER)STRING')

SYSLINEO(DDNAME,LINENUM) sets the line number for the next line to be written to a WYLBUR EDIT file. The first argument specifies the DDNAME of the WYLBUR EDIT file. The second argument specifies the line number. Note that the line number is an integer value equal to the desired WYLBUR line number multiplied by 1000.

The function fails if the file is not output associated or if the line number is greater than 99999999. The function returns the null string.

SYSOPEN

LOAD('SYSOPEN(STRING,STRING)STRING')

SYSOPEN provides additional I/O facilties beyond those provided by the standard OS interface. Normally, a file is opened at the time of the first read or write. SYSOPEN, however, opens a file when called. In addition, SYSOPEN provides facilties for opening files for update, specifying dataset names, specifying entire JFCBs, and specifying WYLBUR files.

SYSOPEN(DDNAME,OPTION) opens a file. The file to be opened is either specified in the JCL or in the OPTION argument. The option argument has the following forms:

1. 1 character string: 'I' for input, 'O' for output, or 'U' for update. The file specified by the DDNAME will be immediately opened in the appropriate mode.

2. 45 character string: This is an extension of the previous case, except that the extra 44 characters represent a dataset name to be substituted into the JFCB before the open is done. Thus, the file opened is named by the extra 44 characters.

   The DD statement referenced by the DDNAME must be present in the JCL and the dataset name specified in the option must be located on the volume defined by the DD statement.

3. 177 character string: This is an extension of the previous case, except that the extra 176 characters represent an entire JFCB to be used for the open.

   Build the JFCB carefully!

SYSOPEN may be called either before or after the call to INPUT or OUTPUT. However, if INPUT or OUTPUT is called before SYSOPEN, be sure the file has not been opened implicitly by reading or writing a record. SYSOPEN returns the null string.

WYLBUR EDIT files can be opened either explicitly or implicitly. Inserting a 'W' at the front of a valid option string specifies an explicit open of a WYLBUR file. For example, SYSOPEN('MYFILE','WI'). Additionally, any disk file with the following DCB attributes will be implicitly opened as a WYLBUR EDIT file:

RECFM=U

LRECL=BLKSIZE

BLKSIZE between 3156 and 7000

Inserting an 'N' before a valid option string overrides the implicit opening of a WYLBUR EDIT file. This means that the file will NOT be opened as a WYLBUR EDIT file, even if its DCB characteristics match those above.

SYSOS

LOAD('SYSOS(STRING)STRING')

SYSOS() returns the name of the operating system under which the program is running, and indication of whether or not the program is running under TSO.

SYSPARM

LOAD('SYSPARM(STRING)STRING')

SYSPARM() returns the entire parameter field. See also UPARM().

SYSRELSE

LOAD('SYSRELSE(STRING)STRING')

SYSRELSE(DDNAME) causes the current block of an input file to be released. The next read from that file will get the first record from the next block. The function returns the NULL string.

SYSTRACE

LOAD('SYSTRACE(STRING,INTEGER,INTEGER)STRING')

SYSTRACE is called to enable file tracing for a file other than SYSPRINT. When a file is being traced, any record read from it or written to it will be listed on SYSPRINT. The F parameter limits the number of records written to SYSPRINT by file tracing.

SYSTRACE(<FILENAME>,<COUNT>,<SKIP>)

where <FILENAME> is the name for the file to be traced, <COUNT> is the maximum number of records to be traced from that file, and <SKIP> is the number of records to be read or written before tracing is to start. Note that <FILENAME> can be any filename valid as the second argument to the input or output functions.

SYSTRACE always returns a null STRING.

SYSTRUNC

```
LOAD('SYSTRUNC(STRING)STRING')
```

SYSTRUNC(DDNAME) forces the current QSAM buffer to be written out immediately, whether it is full or not. SYSTRUNC only works for OUTPUT associated files. The function returns the NULL string.

SYSUSER

```
LOAD('SYSUSER(STRING)STRING')
```

SYSUSER() returns the jobname under which the program is running. If the program is running under TSO, the TSO userid is returned instead.

TCONV

```
LOAD('TCONV(STRING)STRING')
```

TCONV(STRING) is equivalent to a TPUT(STRING) followed by a TGET(). Thus, TCONV issues a prompt before reading a line from the terminal.

TCONV0

```
LOAD('TCONV0(STRING)STRING')
```

TCONV0 is equivalent to a TPUT(STRING) followed by a TGET0(). Thus, TCONV0 issues a prompt before reading a line from the terminal. The function will fail if a null string is entered by the user.

TGET

```
LOAD('TGET(STRING)STRING')
```

TGET() reads a string from the terminal without issuing a prompt. The string read from the terminal is returned exactly as entered, and is not converted in any way. TGET will automatically retry most TSO terminal I/O error conditions.

TGETO

```
LOAD('TGETO(STRING)STRING')
```

TGETO is similar to TGET except the function will fail if a NULL string is entered by the user.

TPUT

```
LOAD('TPUT(STRING)STRING')
```

TPUT(STRING) writes its argument string to the terminal exactly as given. Note that if a carriage return is desired at the end of the line, it must be appended to the argument string. The function normally returns a NULL string and will fail under certain error conditions such as an I/O error.

UPARM

```
LOAD('UPARM(STRING)STRING')
```

UPARM() returns all characters following the first slash in the parameter field. The compiler does not process the user parameters, but keeps them for later reference by the executing user program.

If no slash appears in the parameter field, or if nothing follows the slash, UPARM returns the null string.

Since UPARM will typically be called only once, the entire process of loading it, calling it, and unloading it can be done in a single statement. (Remember, both LOAD and UNLOAD return the null string.)

```
PARMS = LOAD('UPARM()STRING') UPARM() UNLOAD('UPARM')
```

*****PUBLISHED BOOKS*****

1. Dewar, Robert B.K. SPITBOL Version 2.0, SNOBOL4 Project Document S4D23, Chicago, Illinois: Illinois Institute of Technology, 1971.

   Original documentation for SPITBOL/360 and the basis for this manual. Out of print.

2. Gimpel, James F. Algorithms in SNOBOL4, New York: John Wiley, 1976.

   Describes many useful applications of SNOBOL4, presents program source, and discusses differences among the various SNOBOL implementations.

3. Griswold, R.E., J.F. Poage, and I.P. Polonsky. The SNOBOL4 Programming Language, 2nd edition. Englewood Cliffs, N.J.: Prentice Hall, 1971.

   Definitive description of SNOBOL4. Essential reference.

4. Griswold, R.E. and M.T. Griswold. A SNOBOL4 Primer, Englewood Cliffs, N.J.: Prentice Hall, 1973.

   A general introduction to programming in SNOBOL4. Useful for persons not experienced in programming.

5. Griswold, Ralph E. String and List Processing in SNOBOL4, Englewood Cliffs, NJ: Prentice Hall, 1975.

   Describes SNOBOL4 techniques for string and list processing followed by a discussion of various applications. Book is out of print but can be obtained for $8.00 postpaid from the SNOBOL Project at the address given below.

*****SNOBOL4 PROJECT PUBLICATIONS*****

The SNOBOL4 Project is still active at The University of Arizona and certain documents can be obtained at the following address:

SNOBOL PROJECT
Department of Computer Science
The University of Arizona
Tucson, AZ 85721

1. SNOBOL4 Information Bulletin, published irregularly.

2. Bibliography of Documents Related to SNOBOL Programming Languages, TR78-18a.

3. Bibliography of Numbered SNOBOL4 Documents, Publication S4D43.