Macro SPITBOL Quick Reference Guide

Last Revision: February 12, 2017

SPITBOL Command Line

```
spitbol   [-?] [-#=file-and-options]
          [-T=term-file] [-a] [-b]
          [-c] [-d#] [-e] [-f] [-g#]
          [-h] [-i#] [-k] [-l] [-m#]
          [-n] [-o=out-file] [-p]
          [-r] [-s#] [-t#]
          [-u "string"] [-x] [-y]
          [-z] [in-file ...] [args]
```

Option default values are:

```
-d64m -g60 -i128k -m4m -s128k
-t120
```

Protected Keywords

&ABORT
&ALPHABET
&ARB
&BAL
&FAIL
&FENCE
&FILE
&FNCLEVEL
&LASTFILE
&LASTLINE
&LASTNO
&LCASE
&LINE
&REM
&RTNTYPE
&STCOUNT
&STNO
&SUCCEED
&UCASE

Unprotected Keywords

&ABEND
&ANCHOR
&CASE
&CODE
&COMPARE
&DUMP
&ERRLIMIT
&ERRTEXT
&ERRTYPE
&FTRACE
&FULLSCAN
&INPUT
&MAXLNGTH
&OUTPUT
&PROFILE
&STLIMIT
&TRACE
&TRIM

Special Names

ABORT
CONTINUE
END
FRETURN
INPUT
NRETURN
OUTPUT
RETURN
SCONTINUE
TERMINAL

Built-in Data Types

Array
Code
Expression
Integer
Name
Pattern
Program-defined
Real
String
Table

Data Type Conversion

Convert From	*Convert To*								
	S	**I**	**R**	**A**	**T**	**P**	**N**	**E**	**C**
String	A	U	U			A	A	U	U
Integer	A	A	A			A	A	A	
Real	A	U	A			A	A	A	
Array				A	U				
Table				A	A				
Pattern						A			
Name	U	U	U			U	A	U	U
Expression								A	
Code									A

Space means conversion is impossible
A means conversion is always possible
U means conversion is usually possible (depending on the value involved)

Variable Name Rules

1. Each variable name must begin with an upper-case or lower-case letter.
2. The remaining characters may be any combination of letters, numbers, periods (.), and underscores (_).
3. Variable names can't be longer than the maximum program line length.

Multiple Statements on a Line

1. The semicolon (;) may be used to place several statements on a single line.
2. Placing labels in the middle of a line is difficult to read and should be avoided.
3. This feature is most frequently used when preparing one or more SPITBOL statements for compilation via the CODE() built-in function.

Statement Labels

1. Statement labels must start in column one of each statement.
2. If multiple statements are on a single line (separated by semicolons), the label must be immediately after the preceding semicolon.
3. A statement label is needed when a statement is the target of any statement's Goto field.
4. The END label must be the last line in a SPITBOL program.

SPITBOL Comment Statements

A comment is specified by placing an asterisk (*) in column one. All text through the end of the line is considered to be a comment. IT will be included in the listing file but will otherwise be ignored by SPITBOL.

Statement Continuation

A SPITBOL statement may be divided across two or move lines by placing a plus (+) or period (.) in column one of each successive line.

Goto Field

Each statement has the concept of success or failure. This is used to manage the program flow.

Unconditional Goto

```
:(TargetLabel)
:($('STR' VAR))
```

Conditional Goto

```
:S(Success.Lbl)
:F(Failure.Lbl)
:S(Success.Lbl)F(Failure.Lbl)
:F(Failure.Lbl)S(Success.Lbl)
```

Direct Goto

```
:<VAR>
:S<VAR>F(COMPILE_ERROR)
```

Control Statements

Control statements provide instructions to the SPITBOL compiler. They begin with a minus (-) in column one. Only the first four letters of each Control Statement are recognized (remaining letters are ignored). Unrecognized Control Statements are silently ignored.

General Controls	*Listing Controls*
-ERRORS	-EJECT
-NOERRORS	-LINE *n "filename"*
-CASE *#*	-LIST
-COPY *"filename"*	-NOLIST
-INCLUDE *"filename"*	-PRINT
-FAIL	-NOPRINT
-NOFAIL	-SINGLE
-EXECUTE	-DOUBLE
-NOEXECUTE	-SPACE *n*
-IN*##*	-STITL *text*
-OPTIMIZE	-TITLE *text*
-NOOPTIMIZE	

Defaults:

-CASE 1	-EXECUTE
-PRINT	-IN1024
-NOLIST	-NOERRORS
-FAIL	-OPTIMIZE

General Statement Format

```
LABEL SUBJECT ? PATTERN = REPLACEMENT :GOTO
```

Evaluate expression:

```
SUBJECT
```

Assignment:

```
SUBJECT = REPLACEMENT
```

Pattern Match:

```
SUBJECT ? PATTERN
```

Pattern Match with Replacement:

```
SUBJECT ? PATTERN = REPLACEMENT
```

Unary Operators

Operator	Definition
@	Assign cursor position to its operand
~	Negates failure or success of its operand
?	Interrogation – returns null if operand succeeds
&	Keyword
+	Indicates positive numeric operand
-	Negates numeric operand
*	Defers evaluation of expression
$	Indirection
.	Returns a name

Unused unary operators are !, %, /, #, =, and |

Binary Operators

Operator	Associates	priority	Definition
=	Right	0	Assignment
?	Left	1	Pattern match
|	Right	3	Pattern alternation
Space	Right	4	Concatenation or match
+	Left	6	Addition
-	Left	6	Subtraction
/	Left	8	Division
*	Left	9	Multiplication
^ or ! or **	Right	11	Exponentiation
$	Left	12	Immediate assignment
.	left	12	Conditional assignment

The following binary operators are undefined:

Operator	Associates	priority	Definition
&	Left	2	Unused
@	Right	5	Unused
#	Left	7	Unused
%	Left	10	Unused
~	Right	13	Unused

Primitive Built-in Patterns

ABORT — Causes immediate failure of entire pattern match.

ARB — Matches zero or more characters. It matches the shortest string possible that allows the remaining portion of the pattern to match.

BAL — Matches any non-null string which is balanced with respect to parentheses. A string without parentheses is considered balanced. It matches the shortest string possible.

FAIL — Causes failure of this portion of the pattern match, causing the scanner to backtrack and try alternatives.

FENCE — Matches the null string and succeeds when the scanner is moving left to right in the pattern, but fails if the scanner must back up through it seeking alternatives, if any.

REM — Matches zero or more characters from the current cursor position to the end of the subject string.

SUCCEED — Matches the null string and always succeeds.

Pattern Matching Algorithm

1. Set the subject string cursor to zero.
2. Point to the first pattern element.
3. If the current pattern has an alternative, push the alternative and current cursor position onto the stack.
4. Apply the current pattern to the subject string at the current cursor position. If it succeeds, advance the cursor past the characters matched and go to step 5. Go to step 6 if it fails.
5. (Pattern element matched) If the current pattern has a subsequent, point to it and go to step 3. If there is not subsequent, the entire pattern match has succeeded.
6. (Pattern element did not match) Pop the stack to get a previous alternative and old cursor position. IF there is one, make it the current pattern and cursor and go to step 3. If the stack is empty and keyword &ANCHOR is nonzero, the entire pattern match has failed. If &Anchor is zero, advance the starting cursor position by one and go to step 2 if more subject characters remain.

Built-in Functions : Arrays and Tables

ARRAY(s, arg)
ITEM(array, i1, i2, …)
ITEM(table, arg)
PROTOTYPE(array)
RSORT(array, i)
RSORT(table, i)
SORT(array, i)
SORT(table, i)
TABLE(I, x, arg)

Built-in Functions : Compiliation

EVAL(s)
CODE(s)

Built-in Functions : Function Control

APPLY(name,arg1, …)
ARG(name, i)
DEFINE(s, name)
LOAD(s1, s2)
LOCAL(name, i)
OPSYN(s1, s2, i)
UNLOAD(name)

Built-in Functions : Input/Output

BACKSPACE(name)
DETACH(name)
EJECT(channel)
ENDFILE(channel)
INPUT(name, chan, s)
OUTPUT(name, chan, s)
REWIND(channel)
SET(channel, I, i)

Built-in Functions : Memory

CLEAR(s)
COLLECT(i)
DUMP(i)

Built-in Functions : Miscellaneous

CHAR(i)
CONVERT(arg, s)
DATATYPE(arg)
DATE()
EVAL(expression)
SIZE(s)
TIME()

Built-in Functions : Numeric

ATAN(n)
CHOP(r)
COS(n)
EXP(n)
LN(n)
REMDR(n1, n2)
SIN(n)
SQRT(n)
TAN(n)

Built-in Functions : Numeric Comparison

EQ(n1, n2)
GE(n1, n2)
GT(n1, n2)
INTEGER(arg)
LE(n1, n2)
LT(n1, n2)
NE(n1, n2)

Built-in Functions : Object Comparison

DIFFER(arg1, arg2)
IDENT(arg1, arg2)

Built-in Functions : Object Creation

COPY(arg)
DUPL(pattern, i)

Built-in Functions : Pattern Match

ANY(s)
ARBNO(pattern)
BREAK(s)
BREAKX(s)
FENCE(pattern)
LEN(i)
NOTANY(s)
POS(i)
RPOS(i)
RTAB(i)
SPAN(s)
TAB(i)

Built-in Functions : Program Control

EXIT(i)
EXIT(s)
HOST(I, arg1, …)
SETEXIT(name)
STOPTR(name, type)
TRACE(name, s)

Built-in Functions : Program-Defined Data Type

DATA(s)
FIELD(s, i)
DATATYPE(arg)

Built-in Functions : String Comparison

LEQ(s1, s2)
LGE(s1, s2)
LGT(s1, s2)
LLE(s1, s2)
LLT(s1, s2)
LNE(s1, s2)

Built-in Functions : String Synthesis

DUPL(s, i)
LPAD(s1, I, s2)
REPLACE(s1, s2, s3)
REVERSE(s)
RPAD(s1, I, s2)
SUBSTR(s, i1, i2)
TRIM(s)

Initial Values for Unprotected Keywords

&ABEND = 0‡
&ANCHOR = 0
&CASE = 1
&CODE = 0
&COMPARE = 0‡
&DUMP = 0
&ERRLIMIT = 0
&ERRTEXT = “”
&ERRTYPE = 0
&FTRACE = 0
&FULLSCAN = 1‡
&INPUT = 1
&MAXLNGTH = ‡
&OUTPUT = 1
&PROFILE = 0
&STLIMIT = 2147483647
&TRACE = 0
&TRIM = 0

‡ &ABEND and &COMPARE are present for compatibility with other SNOBOL4 and SPITBOL implementations.

‡ &FULLSCAN is always 1. Assigning a zero value will cause an error condition to occur.

‡ &MAXLNGTH default value is implementation dependent (typical values are 65536, 4194304, or 8388608). For the Linux version, it is 4194304. For the macOS version, it is 8388608.