NotSoBASIC is based upon Sinclair QL SuperBASIC.

This document describes the differences between the language definition of SuperBASIC as described in the Sinclair QL User Manual and NotSoBASIC.

Language levels.

Because this language has the purpose of being both a beginner's language which allows for growth and a fully fledged data processing language the system has two modes of operation, "Beginner" and "Standard". The only difference between them is that in Beginner mode the language enforces such things as sequential line numbers, which can act as "training wheels" for those starting off, and will disable a small number of potentially problematic functionality. In other words, the language will look very much like 1980s BASICs. It is a complete sub-set of the Standard mode.

Standard mode removes the need for sequential numeric labels and enables all features.

By default when started in interactive mode the language will be in Beginner mode. This will be able to be changed in the configuration file.

If run in non-interactive mode then Standard mode is the default as it is envisaged that this will only be used by experienced users.

Including other files. (Standard mode only)

So as to make maintenance of programs more easy and to help with a semi-modular approach to programming, programmers can include other files into the program. These may contain structure definitions or procedures. for this purpose two statements can be used:

```
INCLUDEPATH "URI"[,"URI"[…]]
```

This sets the places the interpreter will look for the files to include using the INCLUDE statement, e.g.

```
INCLUDE support_procs.bas
```

It is suggested that there is only one INCLUDEPATH statement used in any one program and that it appears at the very beginning so as to aid maintainability.

Labels.

Labels appear as the first item on a line and mark a line in the code for future reference by GOTO and other programming constructs.

In Beginner mode every line must start with a numeric label which has a value higher than any previous labels.

In Standard mode labels are optional and must have a name starting with either a number or a '-'. Labels may only contain alphanumeric characters and the underscore character, '_'.

Variables.

Variable names MUST start with a alphabetic character and can only contain alphabetic, numeric and underscore characters. A suffix can be appended so as to give the variable a specific type, e.g. string. Without a suffix character the variable defaults to a floating point value.

Suffixes are:

$	string

```
        %       integer
        @       pointer
```

By default a floating point values are held and processed as single precision numbers and conform to IEEE floating point standards. Integer values are held using the natural "signed int" size of the architecture.

However, the "PRECISION" statement at the beginning of the program can be used to change this behaviour. Using the argument "HIGH" will change the size of both integer and floating point storage. Integer values will be stored as "signed long" and floating point values as double precision. Using the argument "STANDARD" will enforce the lower precision state.

Pointers are special variables, they reference other variables or procedures. If evaluated as a number value in an assignment they will give the number of references the variable it refers to currently has. If evaluated as a string in an assignment then it returns a description of the type of variable or procedure it references.

If a pointer is passed to a procedure and within the procedure a local variable is assigned to it then upon the return from the procedure the value is undefined.

Special constants and variables.

ARGC%, ARGV$()

When a program is started any arguments given after the RUN or LRUN command are passed in the array ARGV$() and the number of arguments passed can be found in the ARGC% integer variable. In non-interactive mode these are the arguments given on the command line.

ENVVAR%, ENVVAR$()

Any environment variables set by the system before the interpreter began can be found in the ENVVAR$() array. The size of the array is set in the ENVVAR% integer variable.

NaN

This is a special constant value which can be used to test if a variable contains an invalid floating point value, such as that returned by the square root of -1.

Pi

This special constant holds the value of Pi to the highest precision of the current floating point format.

Compound variables.

Compound variables (structures) can be created using the "DEFine STRUCTure" command to create a template and then creating special variables with the "STRUCTure" command:

```
DEFine STRUCTure name
Varian
[...]
END STRUCTure
```

```
STRUCTure name varnam[,varnam]
```

An array of structures can also be created using the STRUCTure command, e.g.

```
STRUCTure name varnam(3)
```

The values can be accessed using a "dot" notation, e.g.

```
DEFine STRUCTure person
name$
age
DIMention vitals(3)
END STRUCTure

STRUCTure person myself, friends(3)

myself.name$ = "Stephen"
myself.age = 30
myself.vitals(1) = 36
myself.vitals(2) = 26
myself.vitals(3) = 36

friends(1).name$ = "Julie"
friends(1).age = 21
friends(1).vitals(1) = 36
friends(1).vitals(2) = 26
friends(1).vitals(3) = 36
```

As with standard arrays, arrays of structures can be multi-dimentional.

Structures can contain any number of standard variables, static arrays types and other structures. However, only structures defined BEFORE the one being defined can be used. Structure definitions are parsed before execution of the program begins. Structure variable creation takes place during execution.

Expression evaluation.

An expression is a set of mathematical or string operations. Values can be immediate values, variables or constants. Operators include the comparison '=' which returns either the value zero if the values are non-equal or one if they equate.

During program execution an expression can be given as a string to the EVAL() function which will then evaluate the expression and return the value. If the expression is syntactically erroneous then the BADEXPSYNTAX exception is raised. By default this will cause the program to abort.

Loops.

FOR/NEXT/END FOR:

```
FOR assignment (TO expression [STEP expression] | UNTIL expression | WHILE
expression) [NEXT assignment]
..
[NEXT [var]]
..
END FOR [var]
```

The assignment flags the variable as the loop index variable. Loop index variables are normal variables.

The assignment and the evaluation of the assignment expression happen only once, when entering the loop. The test expressions get evaluated once every trip through the loop at the beginning. If the TO or UNTIL expressions evaluate to zero at the time of loop entry the commands within the loop do not get run.

The STEP operator can only be used if the loop index variable is either a floating point variable or an integer. The expression is evaluated to a floating point value and then added to the loop index variable. If the loop index variable is an integer then the value returned by the expression stripped of its factional part (as with ABS()) before being added to the variable.

WHILE/END WHILE:

```
WHILE expression [NEXT assignment]
...
[CONTINUE]
...
END WHILE
```

Equivalent to a FOR loop without an assignment using the WHILE variant e.g.

```
        x = 10
        WHILE x > 3 NEXT x += y / 3
        ...
        END WHILE
```

is equivalent to

```
        FOR x = 10 WHILE x > 3 NEXT x += y / 3
        ...
        END FOR
```

DO/UNTIL:

```
DO
...
[CONTINUE]
...
UNTIL expression
```

The commands within the loop are run until the expression evaluates to a non-zero value.

Functions and procedures.

A function is merely a special form of a procedure which MUST return a numeric value. The suffix of a procedure determines its type, in the same way as variable names.

```
DEFine PROCedure name[(parameter[,parameter[...]])]
...
[RETURN expression]
END PROCedure
```

```
DEFine FUNction name[(parameter[,parameter[...]])]
...
RETURN expression
END FUNction
```

Parameters are local names with reference the passed values by reference. This means that any modification of the parameters within the procedure will change the value of any variables passed to it.

Variables created within the procedure will be local to the current incarnation, allowing recursion. Variables with global scope are available within procedures but will be superseded by any local variables with the same name.

Mathematical operations.

Unless "PRECISION HIGH" has been used at the beginning of the program then all floating point operations will be done using IEEE single precision operations.

Unless the OVERFLOW or UNDERFLOW exceptions are set to be caught such exceptions will be ignored. All other floating point exceptions will cause the termination of the program unless caught using the "ON EXception" statement giving a procedure to call.

Device and file access.

Files and other devices are accessed via a standard URI and not the restrictive SuperBASIC format. This allows direct access to files on remote network resources, e.g.

```
LOAD "ftp://username:password@ftp.somewhere.net/pub/somecode.bas"
```

Comments.

In addition to the "REMark" statement a comment line can begin with the '#' character. This allows the use of the UNIX/Linux '#!' automatic interpreter selection to start a program in non-interactive mode.

Exceptions.

Unlike the partially implemented "ON ERROR" statement in SuperBASIC the "ON EXception" statement allows a more fine grained approach to exceptions generated by the system or the program. The general form of the statement is:

```
ON EXception exceptionname procname|IGNORE|ABORT
```

When an exception occurs, such as a division by zero or a syntactical error in a expression given to EVAL() the language's exception handler will by default call its own handling function or exit the program. The "ON EXception" statement allows the programmer to change the default action for the named exception type and either pass processing to a named procedure, ignore the exception or abort the program.

If an exception occurs and processing is passed to a procedure upon return from the procedure execution continues from the statement after that which caused the exception.

Some exceptions cannot be ignored but can be handled.

Table of exceptions:

Exception name	Can be ignored	Can be redirected	Default
Maths:			
OVERFLOW	*	*	ignore
UNDERFLOW	*	*	ignore
DIVBYZERO	*	*	abort
General:			
BADSUBSCRIPT		*	abort
BADEXPSYNTAX		*	abort