TERRAN BASIC REFERENCE MANUAL

For Version 0.4

First Edition

O'REALLY? Press

Contents

1 Introduction **5**

2 Basic Concepts **6**
2.1 Values and Types 6

3 The Language **7**
3.1 Line Format 7
3.2 Lexical Conventions 7
3.3 Constants 8

4 Commands and Statements **10**

5 Operators **11**
5.1 Overview 11
5.2 Precedence 11
5.3 Precautions 12
5.4 Arithmetic Operators 12
5.5 Logical Operators 12
5.6 Relational Operators 13
5.7 Assignment Operators 13
5.8 Bitwise Operators 13
5.9 Miscellaneous Operators 14

6 Functions **15**
6.1 Synopsis 15
6.2 In-depth description 18

7 Implementation **19**
7.1 Primitive Data Types 19

8 Index **20**

Chapter 1

Introduction

Terran BASIC (*TBASIC* afterwards) is an BASIC language heavily influenced by the Commodore Basic version 2 (the one used in the legendary *Commodore 64*), meant to be used with (but not necessarily) Lua to provide *retro* feeling for any virtual computers that run upon it, targeted especially the OpenComputers and ComputerCraft for Minecraft. BASIC is an non-structured*, general-purpose†, high-level‡ programming language which is meant to be easy to use.

*prone to, and really good at, making spaghetties

†does whatever computers do

‡at least the codes are written using ordinary letters, instead of 0s and 1s

Chapter 2

Basic Concepts

This chapter describes the basic concepts of the language.

2.1 Values and Types

BASIC is a *Dynamically typed language*, which means variables do not know which group they should barge in; only values of the variable do. In fact, there are no type definitions in the language. We do want our variables to feel themselves awkward.

There are five basic types: *nil*, *boolean*, *number*, *string* and *array*. *Nil* is the type of the literal value **NIL**, who likes to possess nothing (I think he's secretly a buddhist). *Boolean* is the type of the values that is either **TRUE** or **FALSE**, he knows no in-between. Both **NIL** and **FALSE** makes condition *false*, any other values, even the number 0, makes it *true*, because *0 is truly a number, ya See-barbarians*. *Number* represents real (double-precision floating-point) numbers. Operations on numbers follow the same rules of the underlying virtual machine[*], and such machines must follow the IEEE 754 standard[†]. *String* represents immutable[‡] sequences of bytes. However, you can't weave them to make something like *string array*[§].

2.1.1 Arrays are somewhat special; they can be work as both array and a number. When you read array variable as an array using index notation `MYARRAY(3)`, the variable will behave as an ordinary array. However, if you evaluate the array as a number, using `MYARRAY`, their first element is returned, without any errors. This feature is there to use FOR statement conveniently and *to screw ya debugging*.

[*] if you are not a computer person, just disregard
[†] ditto.
[‡] cannot be altered directly
[§] future feature…maybe…? Probably not…

Chapter 3

The Language

This chapter describes the lexis, the syntax, and the semantics of TBASIC. In other words, this chapter describes which words are good, how they can be jumbled up, and what the hell they could even mean.

3.1 Line Format

Program lines in BASIC program have the following format:

```
nnnnn BASIC-statement <carriage return>
```

Only one BASIC statement may be placed on a line. A program line always begins with a line number and ends with a carriage return.

3.2 Lexical Conventions

Names (also called *identifiers*) in TBASIC can be any string of letters, digits, and underscores, not beginning with a digit. Identifiers are used to name variables.

The following *keywords* are reserved and cannot be used as names:

ABORT	CBRT	DELETE	FOR	INPUT	LOG
ABORTM	CEIL	DIM	GET	INT	M_E
ABS	CHR	DO	GO	INV	M_PI
AND	CLR	END	GOSUB	LEFT	M_2PI
ASC	CLS	FALSE	GOTO	LEN	M_ROOT2
BACKCOL	COS	FLOOR	HTAB	LIST	MAX
BEEP	DEF	FN	IF	LOAD	MID

MIN	OR	RND	SIN	TAN	VAL
MINUS	PRINT	ROUND	SIZEOF	TEXTCOL	VTAB
NEW	REM	RUN	SQRT	THEN	XOR
NEXT	RENUM	SAVE	STEP	TO	
NIL	RETURN	SCROLL	STR	TEMIT	
NOT	RIGHT	SGN	TAB	TRUE	

TBASIC is a case-insensitive language, and so is the reserved word.

The following strings denote other tokens:

>>>	!	<=	<>	*	(
<<	;	=<	><	/	)
>>	==	>=	=	+	,
|	>	=>	:=	−	
&	<	!=	^	%	

3.3 Constants

TBASIC predefines some constants using its variable system. You can read from these variables, but any new assignment will have no effects and T-BASIC will *not* raise any errors.

Name	**Value**	**Remarks**
M_PI	3.1415926535898	Mathematical constant π
M_2PI	6.2831853071796	Mathematical constant 2π
M_E	2.7182818284591	Mathematical constant e
M_ROOT2	1.4142135623731	Mathematical constant $\sqrt{2}$
TRUE	*true*	Self explanatory
FALSE	*false*	do.
NIL	*nil*	do.

Name	Value	Remarks
_VERSION	0.4	Current TBASIC version

Chapter 4

Commands and Statements

Chapter 5

Operators

This chapter describes operators that can be used in TBASIC.

5.1 Overview

There are 33 operators in TBASIC version 0.4.

5.2 Precedence

Operators has something called *precedence*. It's not something hard despite of its name (especially if you are not a native English speaker, like me). It's just a thing that describes our language programmatically[*], which goes like *multiplication comes before the addition and subtraction*.

Order	Operators
1	:= =
2	OR
3	AND
4	|
5	XOR
6	&
7	== != <> ><
8	<= >= =< => < >
9	TO STEP
10	>>> << >>
11	;
12	+ −
13	* / %
14	NOT !
15	^ SIZEOF
16	MINUS

[*] in computer language

5.3 Precautions

Because of the way the language is desingned†, you must be careful when using $-$ operator.

Code	Evaluation	Remarks
`FOO - 1`	subtraction(FOO, 1)	Variable FOO subtracted by 1
`FOO -1`	FOO(MINUS 1)	Function FOO() with argument -1
`FOO-1`	FOOMINUS1	Syntax error

5.4 Arithmetic Operators

5.4.1 PLUS *number* $+$ *number* — adds two numbers

5.4.2 MINUS *number* $-$ *number* — subtracts two numbers

5.4.3 TIMES *number* * *number* — multiplies two numbers

5.4.4 DIVIDE *number* / *number* — divides two numbers

5.4.5 MODULO *number* % *number* — gets modulo (remainder) of two numbers

5.4.6 POWEROF *number a* ^ *number b* — gets *a* to the power of *b*, or a^b

5.5 Logical Operators

5.5.1 AND *condition 1* AND *condition 2* — returns **TRUE** if both conditions (boolean value) are true; **FALSE** otherwise

5.5.2 OR *condition 1* OR *condition 2* — returns **TRUE** if one or more conditions (boolean value) are true; **FALSE** if both conditions are false

5.5.3 NOT NOT *condition* — negates condition (boolean value)

†don't write like **`1+3*2+4/7`**, they look hideous

5.6 Relational Operators

5.6.1 EQUALTO *number* $==$ *number* — returns **TRUE** if two numbers represent same quantity; **FALSE** otherwise

5.6.2 NOTEQUALTO *number* $!=$ *number* — returns **TRUE** if two numbers represent different quantity; **FALSE** otherwise; aliases: $<>$, $><$

5.6.3 GREATERTHAN *number a* $>$ *number b* — returns **TRUE** if $a > b$ is satisfied; **FALSE** otherwise

5.6.4 LESSTHAN *number a* $<$ *number b* — returns **TRUE** if $a < b$ is satisfied; **FALSE** otherwise

5.6.5 GEQTHAN *number a* $>=$ *number b* — returns **TRUE** if $a \geq b$ is satisfied; **FALSE** otherwise; alias: $=<$

5.6.6 LEQTHAN *number a* $<=$ *number b* — returns **TRUE** if $a \leq b$ is satisfied; **FALSE** otherwise; alias: $=>$

5.7 Assignment Operators

5.7.1 AS *variable* $=$ *value* — assign *value* to *variable*; alias: $:=$

5.8 Bitwise Operators

5.8.1 BAND *integer* & *integer* — performs bitwise AND on two integers

5.8.2 BOR *integer* | *integer* — performs bitwise OR on two integers

5.8.3 BNOT ! *integer* — performs bitwise NOT on integer

5.8.4 BXOR *integer* XOR *integer* — performs bitwise XOR on two integers

5.8.5 LSHIFT *integer* $<<$ *offset* — performs left shift on *integer* by *offset*

5.8.6 RSHIFT *integer* $>>$ *offset* — performs signed right shift on *integer* by *offset*; if leftmost bit is 1, bits on the left side are padded with 1

5.8.7 URSHIFT *integer* $>>>$ *offset* — performs unsigned right shift on *integer* by *offset*

5.9 Miscellaneous Operators

5.9.1 CONCAT *string* ; *string* — concatenates two strings

5.9.2 MINUS $-$*number* — negates *number*, or $-n$

5.9.3 SIZEOF SIZEOF *array* — returns size of array

5.9.4 TO *number* TO *number* — creates integer sequence of specified range; `1 TO 4` will create an array of *1, 2, 3, 4*; `8 TO 3` will create *8, 7, 6, 5, 4, 3*

5.9.5 STEP *int array* STEP *number* — filters input array such that every $1 + n$th number is remained; `1 TO 10 STEP 3` will create an array of 1, 4, 7, 10

Chapter 6

Functions

This section describes built-in functions for the TBASIC.

Functions in TBASIC can take *argument*s. Arguments are the values you pass to functions. Functions can take one or more arguments. If the function only requires single argument, you can call the function with an argument without parentheses, like `GOTO 30`, but if the function takes two or more arguments, they must be surrounded with parentheses, such as `PRINT("HL", 3, "CONFIRMED")`.

6.1 Synopsis

Note: trigonometric functions assumes *radian* as its input; 2π is a full circle.

6.1.1 **ABORT** — halts current program

6.1.2 **ABORTM** *reason* — halts current program with message

6.1.3 **ABS** *number* — returns absolute value for the number

6.1.4 **ASC** *character* — returns ASCII code for the character

6.1.5 **BACKCOL** *number* — sets background colour of the terminal (if applicable)

6.1.6 **BEEP** *pattern* — make computer produce beeping sound (does not work with CC)

6.1.7 **CBRT** *number* — returns cubic root of the number

6.1.8 **CEIL** *number* — returns round-up of the number

6.1.9 **CHR** *ASCII-code* — returns character for the code

6.1.10 CLR — deletes all variables declared

6.1.11 CLS — clears screen buffer (if supported)

6.1.12 COS *radian* — returns trigonometric cosine for the number

6.1.13 DEF FN *name*, *code* — defines new function

6.1.14 DIM *name*, *(d1[, d2[, d3…]])* — defines new array with given dimensions

6.1.15 END — successfully exits program

6.1.16 FLOOR *number* — returns round-down of the number

6.1.17 FOR *a = x FROM y[STEP z]* — starts counted loop with counter *a* that goes from *x* to *y*, with optional step. Use NEXT *a* to make a loop

6.1.18 GET *variable[, variable…]* — get a single character from the keyboard and saves the code of the character to the given variable(s)

6.1.19 GOSUB *line* — jumps to a subroutine. Use RETURN to jump back

6.1.20 GOTO *line* — jumps to a line

6.1.21 HTAB *amount* — moves output cursor horizontally by given amount (if applicable)

6.1.22 IF *condition* THEN *command* — evaluates condition and executes command if the condition is true

6.1.23 INPUT *TODO: Future feature*

6.1.24 INT *number* — returns integer part of the number

6.1.25 INV *number* — returns (1.0 / number)

6.1.26 LEFT *string*, *number* — returns substring of leftmost *number* characters

6.1.27 LEN *string* — returns length of the string

6.1.28 LIST *[from[, to]]* — prints out commands that have entered (shell function)

6.1.29 LOAD *path* — loads the file as a command (shell function)

6.1.30 LOG *number* — returns natural logarithm of the number

6.1.31 MAX *number, number[, …]]* — returns the biggest of the numbers

6.1.32 MID *string*, *start*, *end* — returns substring of the string

6.1.33 MIN *number, number[, …]]* — returns the smallest of the numbers

6.1.34 NEW — deletes all the commands that have entered (shell function)

6.1.35 NEXT *variable* — advances variable which is used by FOR loop

6.1.36 PRINT *string* — prints string to the terminal

6.1.37 RAD *degree* — converts degree to radian

6.1.38 REM — marks current line as remarks, or “comments”

6.1.39 RETURN — returns subroutine called by GOSUB

6.1.40 RIGHT *string*, *number* — returns substring of rightmost *number* characters

6.1.41 RND — returns random decimal number that is $0.0 \leq n < 1.0$

6.1.42 ROUND — returns half-up of the number

6.1.43 RUN — executes the commands that have entered (shell function)

6.1.44 SAVE *path* — saves the command that have entered to the file (shell function)

6.1.45 SCROLL *amount* — scrolls the terminal by given amount

6.1.46 SGN *number* — returns sign of the number; -1.0 for $n < 0$, 1.0 for $n > 0$, 0.0 otherwise.

6.1.47 SIN *radian* — returns trigonometric sine for the number

6.1.48 SQRT *number* — returns square root of the number

6.1.49 STR *number* — returns string representation of the numerical value

6.1.50 TAB *amount* — same as HTAB

6.1.51 TAN *radian* — returns trigonometric tangent for the number

6.1.52 TEXTCOL *number* — sets text colour of the terminal (if applicable)

6.1.53 TEMIT *frequency*, *length* — emits tone of given frequency for given length (in seconds) (does not work with CC)

6.1.54 VAL *string* — returns numerical representation of the string

6.1.55 VTAB *amount* — moves output cursor vertically by given amount (if applicable)

6.2 In-depth description

Chapter 7

Implementation

7.1 Primitive Data Types

7.1.1 Nil Nils are identical to Null for most commonly used languages.

7.1.2 Boolean Any number, including 0, must be evaluated as True. Only the *NIL* and *FALSE* are evaluated as False.

7.1.3 Number Numbers must be implemented using double-precision floating-point, and must follow the IEEE 754 standard. Extra precision (just as x86 would do) may be used.

7.1.4 Arrays Arrays can have as many as 255 dimensions, and it must be supported.

Chapter 8

Index