TeraByte Scripting Language

Reference

TeraByte Unlimited
Las Vegas, Nevada, USA
http://www.terabyteunlimited.com

Revision: 2017-12-08

TeraByte Scripting Language (TBScript) Language Reference

Overview

TBScript is a simple yet flexible scripting language, which allows you to automate many different types of tasks. TBScript is similar to the BASIC language and users familiar with BASIC should quickly become productive using TBScript.

TBScript is loosely typed in that variables do not need to be declared and the same variable can store data of any of the supported types. In addition, it is not case sensitive. Variables and symbols can use any combination of upper and lower case characters.

Structure

A TBScript file consists of one or more subroutines. All scripts must define one subroutine called MAIN. Execution starts at this subroutine. Any subroutine can return a value using the RETURN keyword.

```
// Here is a sample script.
// Text that follows // or ; are comments and
// are ignored by the interpreter.
// Execution begins at the following subroutine
sub main()
   printl(double(5))
end sub

// Here's another subroutine. It returns the
// argument value times two
sub double(val)
   return val * 2;
end sub
```

This example defines two subroutines, main and double. Double accepts an argument and returns that value times two. Main passes 5 to double and prints the result.

Subroutines are called by specifying the name of the subroutine followed by parentheses. If the subroutine takes arguments, they can be specified between the parentheses separated by commas. Subroutine calls may be included in expressions or assigned to a variable, or even used as arguments to other subroutines.

Variables

As stated earlier, variables in TBScript are loosely typed. Any variable can contain a 64-bit integer (32 bit for DOS real mode), floating point, or a string value. In addition, a variable can also be contain sub variables using the “dot” syntax or elements using the “bracket” syntax.

```
// The following line makes a an integer value
a = 452
// This makes it a floating point-value
a = 5.2
// And this one makes it a string
a = "This is a test!"
// These lines creates sub variables of a
a.i = 452
a.f = 5.2
a.s = "This is a test"
// This creates elements of the variable a
a[1] = 452
a[2] = 5.2
```

Note that any of the statements above will create the named variable if it has not already been created. This is also true when a variable is read.

NOTE: Variables are unique to the current subroutine. Variables in different subroutines with the same name are different variables.

Constants

Constants are fixed values. Here are a few examples of valid constants.

```
// This is an integer constant
55
// Here is a floating-point constant
75.2
// Here is a string constant
"This is a test"
// This is an integer constant in hexadecimal (leading 0x)
0x1F
// This is an integer constant in octal (leading 0)
010
```

String constants can contain special escape sequences to create unique characters. A caret (^) initiates an escape sequence. Here is a list of the escape sequences that are supported.

Escape Character	**Meaning**
“^a”	Bell (alert)
“^b”	Backspace
“^f”	Form feed
“^n”	New line
“^r”	Carriage return
“^t”	Tab
“^’”	Single quote
“^””	Double quote
“^^”	A single caret character

You can also specify characters by specifying the ASCII value of the character using either of the following two formats.

“^xNN”	Specifies an ASCII character where NN are two hexadecimal (base 16) digits.
“^NNN”	Specifies an ASCII character where NNN are three octal (base 8) digits.

Operators

Operators are used in expressions to modify or compare subexpressions. TBScript supports arbitrarily complex expressions and all of the following operators.

Unary operators:	+	Positive (the default)
	-	Negative
Assignment operator:	=	Assigns a value to a variable
Concatenation operator:	#	Appends one string to another
Math operators:	+	Adds one value to another
	-	Subtracts one value from another
	*	Multiplies one value by another
	/	Divides one value by another
	%	Divides one value by another and returns the remainder
Bitwise operators:	&	Bitwise AND
	|	Bitwise OR

	^	Bitwise XOR
Comparison operators:	=	Equal
	<>	Not equal
	<	Less than
	>	Greater then
	<=	Less than or equal
	>=	Greater than or equal
Logical operators:	AND	If both expressions are true
	OR	If either expression is true

Note: The concatenation, addition, subtraction, and bitwise operators all operate at the same precedence from left to right. (e.g. "Answer is:" # 3 & 3 results in 0, "Answer is:" # (3 & 3) results in “Answer is:3”)

Here are some examples:

```
A = 5 * ((2 + 3) – 1)
A = (5+3) & (7-2)
A = "This is " # "a test."
IF A > 0 AND B > 0 THEN
   // Both tests are true
END IF
```

Reference

This section provides a complete reference for all TBScript keywords and built-in subroutines in alphabetical order.

NOTE: Terms in the Usage section of the reference enclosed in square brackets ([]) indicate that the term is optional.

ARG, ARGC Subroutines

Usage:
a = ARG(n)
n = ARGC()

Description:
The ARG and ARGC subroutine are used to access any arguments that were passed on the command line (when the script was started). ARGC returns the number of arguments. ARG() returns the argument indicated by n, which can be in the range 1 through the value returned by ARGC.

In addition to the arguments described above, the ARG subroutine returns the fully qualified path of the script file when n = 0.

Example:
```
sub main()
   printl("Script name = ", arg(0))
   for i = 1 to argc()
      printl("Arg ", i, " = ", arg(i))
   next
end sub
```

ASC Subroutine

Usage:
n = ASC(s)

Description:
The ASC subroutine returns the ASCII value of the first character in the string s.

Example:
```
sub main()
   // Print ASCII value of "A"
   printl(ASC("A"))
end sub
```

BINARY Subroutine

Usage:
newvariant = BINARY(variant [[[[, type], startoffset], length], binvartoupdate])

Description:
This subroutine sets or extracts binary data types (used for binary file operations or uefi variables). When *variant* is a binary data type and *binvartoupdate* is not provided then this subroutine exacts data from *variant* to create a new variable of *type*. When *variant* is not a binary data type or *binvartoupdate* is provided then this subroutine returns a new binary data type. When not provided the optional parameters are zero. The values for *type* are as follows: 0=String, 1=Hex String, 2=Wide String, 3=Numeric, 4=Binary. The *startoffset* is the zero based starting offset to the data to extract from or set in *newvariant*. The *length* specifies the number of bytes of data to extract from or set in *newvariant*. When the *length* is zero it is assumed to be the same length as *variant*. When *binvartoupdate* is provided it will be used as the basis of *newvariant* to allow updating an existing binary variable. This subroutine was added in TBSVER 3 and enhanced with type, startoffset, length, and binvartoupdate in TBSVER 9.

Example:
```
sub main()
   bindata=BINARY("STRING")            // 53 54 52 49 4E 47
   bindata=BINARY(0, 3, 0, 1)          // 00
   bindata=bindata # bindata           // 00 00
   bindata=BINARY("3031323334", 1)     // 30 31 32 33 34
   bindata=BINARY("353637", 1, 5, 0, bindata) // 30 31 32 33 34 35 36 37
   word=BINARY(bindata, 3, 0, 2)      // 0x3130
   bindata=BINARY("55AA", 1, 1, 3)     // 00 55 AA 00
   printl(len(bindata))                // 4
end sub
```

BREAK Subroutine

Usage:
n = BREAK(n)

Description:
Enable (n=1) or disable (n=0) the ability to break out of the running of the script by use of the CTRL-C or CTRL-BREAK key on the keyboard.   The return value is the break value prior to setting the new value.

Example:
```
sub main()
   // Disable CTRL-C and CTRL-Break
   BREAK(0)
end sub
```

CHDIR Subroutine

Usage:
r=CHDIR(path)

Description:
Changes the current directory to the given path. This subroutine returns zero on success or a non-zero failure code.

CHR Subroutine

Usage:
s = CHR(n)

Description:
The CHR subroutine returns a string with a single character, which has the ASCII value of the number n.

Example:
```
sub main()
   // Print "A"
   printl(CHR(65))
end sub
```

CLS Subroutine

Usage:
CLS()

Description:
Clears the screen and positions the text cursor at the top, left corner of the screen

CONST Keyword

Usage:
CONST name = value

Description:
Defines a constant symbol.

Constants are similar to variables except a) They are defined in your script outside of any subroutines, and b) Their value cannot be changed. Constants are useful, for example, when you write a script that uses a value in several places, but you want to be able to easily change that value at one location.

There are also default constants: TBSVER contains the version string of the script engine; TBSENV contains "DOS", "LINUX", "WINDOWS", or “UEFI” depending on which type of environment the script is running on; TBSAPPPATH contains the path name to the folder containing the main application.

Example:
```
const A = 100

sub main()
   printl("The value of A is ", A)
end sub
```

DIRECTVIDEO Subroutine

Usage:
DIRECTVIDEO([n])

Description:
This subroutine is used to set the DOS environment to either write directly to video memory or to use the BIOS. By default direct video mode is enabled as it’s much faster. If you have a need to use BIOS video then use this subroutine to turn off direct video mode. This subroutine was added in TBSVER 4.

Example:
```
sub main()
   directvideo(0) // turn off
   directvideo()  // turn on
   directvideo(1) // turn on
end sub
```

EXEC Subroutine

Usage:
EXEC(s[,f])

Description:
Executes a shell command. The string s can be any valid shell command. Returns the return code (errorlevel) of the command. The optional f parameter determines the format used to pass the parameters to external programs. It was added to maintain backwards compatibility. By default (or zero) the parameters are parsed by exec and then passed to the program, otherwise if set to one (1) the raw non-parsed parameters are passed.

Example:
```
sub main()
   exec("program ^"^"param one^"^"" ---param2) // old format
   exec("program ^"param one^" --param2", 1)   // new format is easier
   exec("script.tbs")
   exec("shellcommand")
end sub
```

ExitLoop Keyword

Usage:
ExitLoop

Description:
Exits out of a While/Wend or For/Next loop.

Example:
```
sub main()
   // similar to a repeat/until loop
   while 1
     keyval=GetKey()
     printl("You entered key code ", keyval)
     if keyval=asc("q") then
       exitloop
     end if
   wend
end sub
```

EXT Subroutine

Usage:
EXT(s)

Description:
Executes script extensions that may exist in a product.

Example:
```
sub main()
   ext("extcmd param1 param2")
end sub
```

(The remainder of this page has been intentionally left blank)

FINDFIRST, FINDNEXT Subroutines

Usage:
f = FINDFIRST([s])
f = FINDNEXT(f)

Description:
Use these subroutines to iterate through system files.
The optional argument to FINDFIRST indicates the filespec used to filter the files returned. If the argument is omitted, “*.*” is used.

The value returned by FINDFIRST can then be passed to FINDNEXT repeatedly to iterate through all the files matching the filespec.

The value returned is the name of the file. Both subroutines return an empty filename (“”) when there are no more matching files. The returned value has several members that contain additional information about the current file. These members are NAME, DATE, TIME, SIZE, ATTRIB, CDATE, CTIME, ADATE, ATIME, MODE and SFN (if different than NAME). TBSVER 5 adds sortable date and time values of DATETIME, CDATETIME, ADATETIME.

In order to close the internal find handle you should empty the variable holding the returned value when you abort the find operation before an empty filename(“”) is obtained. (e.g. f=" ")

Example:
```
sub main()
   f = findfirst("*.*")
   while len(f) > 0
      c = c + 1
      print(" ", f.date)
      print(" ", f.time)
      print(" ", f.size)
      print(" ", f.attrib)
      printl(f.name)
      f = findnext(f)
   wend
   printl(c, " file(s)")
end sub
```

(The remainder of this page has been intentionally left blank)

FOR..TO..NEXT Keywords

Usage:

```
FOR var = start TO end
        statements
NEXT
```

Description:
Use a FOR loop to execute a block of statements a specified number of times.

Initially, var is set to the value specified by start. Then, each time the block of statements are executed, var is incremented. When var is greater than end, execution continues on the next statement after the NEXT keyword. If end is less than start, the block of statements is never executed.

NOTE: The start and end values are evaluated only once and the resulting values are cached. So, for example, if the loop modifies values used in the end expression, this will not alter the number of times the block of statements is executed.

NOTE 2: Unlike the BASIC language, the name of the variable is not required nor allowed after the NEXT statement.

Example:

```
sub main()
   for i = 1 to 10
      printl("This is line ", i)
   next
end sub
```

GETCWD Subroutine

Usage:
d=GETCWD([d:path])

Description:
Gets the current working directory of the given drive in path or the current drive if no drive letter provided. This function returns an empty string on error.

GETDATE Subroutine

Usage:
s = GETDATE()

Description:
The GETDATE subroutine returns the current date as a string.

Example:

```
sub main()
   // Extract components of current date
   date = getdate()
   month = mid(date, 1, 2)
   day = mid(date, 4, 2)
   year = mid(date, 7, 4)
end sub
```

GETDATETIME Subroutine

Usage:
s = GETDATETIME([datetimevalue])

Description:
The GETDATETIME subroutine returns a date and time string value based on the current locale setting. If datetimevalue is not provided then the current date and time are used. The datetimevalue parameter is a numeric value based on either Unix time or Windows file time. Large values are assumed to be Windows file time, smaller values Unix time. NOTE: The Right(TBSVER,3)=”x16” version of TBScript does not support the datetimevalue parameter and will return an empty string.

Example:
```
sub main()
   // Extract components of current date/time
   datetime = getdatetime()
   month = mid(time, 1, 2)
   day = mid(time, 4, 2)
   year = mid(time, 7, 4)
   hour = mid(time, 12, 2)
   min = mid(time, 15, 2)
   sec = mid(time, 18, 2)
end sub
```

GETDRIVE Subroutine

Usage:
d=GETDRIVE()

Description:
Returns the current drive letter followed by a colon (e.g. "A:") or empty string if no current drive.

GETENV Subroutine

Usage:
s = GETENV(s)

Description:
Returns the value of the specified environment variable.

GETKEY Subroutine

Usage:
n = GETKEY([prompt[, timeout]])

Description:
The GETKEY subroutine returns the value of the next key pressed by the user. prompt is an optional prompt string that is displayed before waiting for the key press.

timeout is an optional argument that specifies a timeout period, in seconds. If the user does not press any key within the specified number of seconds, the GETKEY subroutine returns a value of 0 without waiting for a keystroke. If the timeout argument is omitted or is 0, the GETKEY subroutine waits for the next keystroke regardless of how long it takes. Note that the timeout argument can be specified only if the prompt argument is specified; however, prompt may be an empty string (“”).

GETSTR Subroutine

Usage:
s = GETSTR([prompt [, maxchars]])

Description:
Returns a string entered by the user. prompt is an optional prompt that is displayed before waiting for the user to enter a string. In addition, maxchars is an optional number that specifies the maximum length of the string that the user can enter.

NOTE: If maxchars is specified, the prompt argument must be included. If TXINIT is active then a newline is not automatically output after pressing enter (except under Windows which always outputs a newline).

GETTIME Subroutine

Usage:
s = GETTIME()

Description:
The GETTIME subroutine returns the current time as a string.

Example:
```
sub main()
   // Extract components of current time
   time = gettime()
   hour = mid(time, 1, 2)
   min = mid(time, 4, 2)
   sec = mid(time, 7, 2)
end sub
```

(The remainder of this page has been intentionally left blank)

GETSYSINFO Subroutine

Usage:
si = GETSYSINFO()

Description:
Returns information about the current system. The variable contains the following members: BIOSDate, BIOSVendor, BIOSVersion, SysFamily, SysManufacturer, SysProductName, SysSKU, SysVersion, SysUUID. TBSVER 6 adds two additional members: BIOSFeatures1, BIOSFeatures2. TBSVER 7 adds CPUCount. For each CPUCount an array (1 based) is provided as CPU[n] with the following members: ID, Cores, CoresEnabled, Threads, and Features. Note that the Core and Feature information may not be reported by the system; however the Threads member is valid if Cores is non-zero. The ID contains the contents of cpuid leaf 1 EAX (low) and EDX (high)values.

BIOSFeatures1		BIOSFeatures2		CPUFeatures	
Bit	**Meaning if Set**	**Bit**	**Meaning if Set**	**Bit**	**Meaning if Set**
4	ISA supported	0	ACPI supported	2	64-bit capable
5	MCA supported	1	USB legacy supported	3	Multi-Core
6	EISA supported	2	AGP supported	4	Hardware thread
7	PCI supported	3	I2O boot supported	5	Execute protection
8	PC Card (PCMCIA) supported	4	LS-120 boot supported	6	Enhanced virtualization
9	Plug and Play supported	5	ATAPI ZIP supported	7	Power/Performance control
10	APM supported	6	1394 boot supported		
11	BIOS is upgradable (Flash)	7	Smart battery supported		
12	BIOS shadowing allowed	8	BBS is supported		
13	VL-VESA supported	9	Fn key network boot supported		
14	ESCD available	10	Targeted content distribution		
15	Boot from CD supported	11	UEFI supported		
16	Selectable boot is supported	12	Virtual Machine		
17	BIOS ROM is socketed				
18	Boot from PC Card (PCMCIA) supported				
19	EDD supported				

Example:
```
sub main()

 si=getsysinfo()
 if (si) then
   printl("BIOS Date: ", si.biosdate)
   printl("BIOS Vendor: ", si.biosvendor)
   printl("BIOS Version: ", si.biosversion)
   printl("System Family: ", si.sysfamily)
   printl("System Manufacturer: ", si.sysmanufacturer)
   printl("System Product Name: ", si.sysproductname)
   printl("System SKU: ", si.syssku)
   printl("System Version: ", si.sysversion)
   printl("System UUID: ", si.sysuuid)
 else
   printl("Unable to obtain the system information")
 end if

end sub
```

GETUEFIVAR Subroutine

Usage:
binvar=UEFIGETVAR(varname, namespaceguid)

Description:
Retrieve a UEFI firmware variable (variables are case sensitive) and its attributes in *binvar.attributes*.   This subroutine is only available when the system booted using UEFI. In linux, the efivarfs must be mounted at /sys/firmware/efi/efivars. The linux command to mount is: mount -t efivarfs none /sys/firmware/efi/efivars. On failure the returned *binvar* is zero bytes in length and contains a member *binvar.errno* to indicate the error code.

Example:
```
sub main()
   t = getuefivar("Timeout", "{8BE4DF61-93CA-11D2-AA0D-00E098032B8C}")
   if (len(t) > 0) then
     printl("Boot Timeout = ", t)
   else
     printl("Unable to retrieve Boot Timeout. Error:", t.errno)
     t = ""   // remove variable
   end if
end sub
```

GLOBAL Keyword

Usage:
GLOBAL name = value

Description:
Defines a global variable.

Global variable are similar to regular variables except they are defined in your script outside of any subroutines.

Example:
```
global A = 100

sub main()
   printl("The value of A is ", A)
   ChangeA()
   printl("The value of A is ", A)
end sub

sub ChangeA()
  A=200
End sub
```

GOTO Keyword

Usage:
GOTO label

Description:
Use the GOTO keyword to jump to another line in the current script. A GOTO line is identified by a symbol followed by a colon (:). The GOTO statement and the label being jumped to must be within the same subroutine.

NOTE: Caution must be taken when jumping into or out of a loop such as a FOR..NEXT or WHILE..WEND loop. For example, if you jumped into a FOR..NEXT loop, execution would continue until the NEXT is encountered, which would produce a “NEXT without FOR” error because the FOR keyword was skipped.

Example:
```
sub main()
```

```
   printl("This line gets executed")
   goto jump
   printl("This line does not get executed")
jump:
   printl("This line also gets executed")
end sub
```

HEX Subroutine

Usage:
s = HEX(n)

Description:
The HEX subroutine returns a string hexadecimal representation of the number n.

Example:

```
sub main()
   // Print F
   printl(HEX(15))
end sub
```

IF..THEN..ELSEIF..ELSE..END IF Keywords

Usage:

```
IF expression THEN
        statements
[ELSEIF expression2 THEN]
        statements
[ELSE]
        statements
END IF
```

Description:
Use the IF keyword to execute a block of statements only if a condition is true.

Optionally, you can also specify additional blocks that are executed only if the previous conditions are false and a new condition is true (ELSEIF), or that are executed only if all other blocks are false (ELSE).

Example:

```
sub main()
   a = 10
   b = 0
   c = 0

   if a > 5 then
      printl("a > 5")
   elseif b > 5 then
      printl("b > 5")
   elseif c > 5 then
      printl("c > 5")
   else
      printl("a, b, and c < 5")
   end if
   // note the following difference due to b being numeric variable
   if b="X" then
     print("b = 0")
   end if
   if "X"=b then
     print("X = b")
   end if
end sub
```

INCLUDE Keyword

Usage:
INCLUDE “filename”

Description:
Use this keyword to reference another file in your script. The include keyword was added in TBSVER 7 and must be used outside of any subroutines.

Example:

```
include "my_common_subroutines.inc" // includes my_sqrt subroutine

sub main()
   printl("The square root of 81 is ", my_sqrt(81))
end sub
```

INSTR Subroutine

Usage:
n = INSTR(s1, s2)

Description:
Use INSTR to find a substring within a string.

INSTR returns the 1-based index of the start of s2 within s1. For example, INSTR(“find”, “in”) returns 2. INSTR returns 0 if the substring was not found.

NOTE: The comparison is case sensitive, which means that INSTR(“find”, “IN”) returns 0.

ISDRIVE Subroutine

Usage:
n = ISDRIVE(s)

Description:
ISDRIVE returns 1 if the drive indicated by s is a valid disk drive. Otherwise, 0 is returned. Only the first character in s is examined so strings like “c”, “C:”, and “c:\temp” all produce the same result.

NOTE: If the specified drive is an existing drive but is not ready (for example, if a floppy drive has no disk in it), ISDRIVE returns 0.

Example:

```
sub main()
   for i = 1 to 26
      s = chr(asc("@") + i)
      if isdrive(s) then
         printl("Drive ", s, ":")
      end if
   next
end sub
```

ISSTRTYPE Subroutine

Usage:
n = ISSTRTYPE(s, t)

Description:
ISSTRTYPE returns 1 if the string type matches the type (t) requested. Use 0 for integer check, 1 for decimal, 2 for alphabetic, 3 for alpha-numeric.

Example:
```
sub main()
   s[1] = "1234"
   s[2] = "23.4"
   s[3] = "abc"
   s[4] = "123abc"

   for i = 1 to 4
     printl("String ^"", s[i],"^"")
     printl("  IsInt: ", IsStrType(s[i],0)
     printl("  IsDec: ", IsStrType(s[i],1)
     printl("  IsAlpha: ", IsStrType(s[i],2)
     printl("  IsAlphaNum: ", IsStrType(s[i],3)
     printl("")
   next
end sub
```

LCASE Subroutine

Usage:
s = LCASE(s)

Description:
Returns a lower case version of a string.

LEFT Subroutine

Usage:
s = LEFT(s, n)

Description:
Returns a string with the left-most characters of s. The number of characters to return is indicated by n. If n is greater than or equal to the length of the string, then the entire string is returned. For example, LEFT(“Test”, 2) returns “Te”.

LEN Subroutine

Usage:
n = LEN(s)

Description:
Returns the number of characters in a string.

LOF Subroutine

Usage:
n = LOF(n)

Description:
Returns the length of an open file. n is a number returned by OPEN.

MID Subroutine

Usage:
s = MID(s, pos [, len])

Description:
Returns a substring of a string. pos specifies the 1-based index of the start of the substring. len specifies the number of characters to return. If len is omitted, the rest of the string is returned.

For example, MID("Test string", 6, 3) returns "str", and MID("Test string", 6) returns "string".

MKDIR Subroutine

Usage:
r=MKDIR(path)

Description:
Creates a new directory. This subroutine returns zero on success or a non-zero failure code.

OCT Subroutine

Usage:
s = OCT(n)

Description:
The OCT subroutine returns a string octal representation of the number n.

Example:
```
sub main()
   // Print 17
   printl(OCT(15))
end sub
```

(The remainder of this page has been intentionally left blank)

OPEN, CLOSE Subroutines

Usage:
n = OPEN(name [,"in" | "in-out" | "in-out-trunc" | "uin" | "uin-out" | "uin-out-trunc" [, “binary”]])
CLOSE(n)

Description:
The OPEN and CLOSE subroutines are used to open a file for access and then close it. The optional open methods specify how the file should be opened. The "in" option opens an existing file as read-only; "in-out" (default) opens or creates a file that can be read or written; "in-out-trunc" will truncate an existing file to zero or create a new file that can be read or written. The "uin" variety of open methods in non-binary mode will look for a Unicode BOM at the beginning of the file and automatically translate the data as needed. The optional “binary” parameter is available in TBSVER 2 or later and treats the data to read/write as binary data (not text strings).

Opened files that are not read-only can be written to using WRITEL, and all files can be read from using READL. The current version only supports reading and writing lines of text.

This subroutine returns -1 if there was a problem opening the file and sets member .errno containing a failure code.

Note: Although the script interpreter will make sure that all opened files are closed eventually, you should explicitly close any files you open. This will prevent you from running out of file handles if your script needs to open several files.

Example:
```
sub main()
   // Open a file
   f = open("file1.txt", "uin-out")

   // Move to the end of any existing text
   seek(f, lof(f))

   // Write 50 lines of text
   for i = 1 to 50
      s = "This is test line " # i # "!!!"
      writel(f, s)
   next

   // Write one blank line
   writel(f)

   // Close file
   close(f)
end sub
```

PAD Subroutine

Usage:
s = PAD(s, n [,0|1|2])

Description:
Returns a string that contains at least n characters. When the input string is less than n characters it can be justified left (default) (0), middle (1), or right (2) by providing a third parameter. If the input string length is greater than or equal to n then the input string is simply returned.

```
sub main()
   // Print [   test   ]
   printl("[", PAD("test", 10, 1), "]")
end sub
```

PRINT, PRINTL Subroutines

Usage:
PRINT(s [, …])
PRINTL(s [, …])

Description:
Use these subroutines to print text to the screen. The difference between PRINT and PRINTL is that PRINTL prints a new line after all text (it moves the text cursor to the start of the next line).

Both subroutines take any number and type of arguments.

Example:
```
sub main()
   a = 5
   b = "Test"
   c = 52.9
   printl("a = ", a, ", b = ", b, ", c = ", c)
end sub
```

RAND Subroutine

Usage:
r = RAND([seed])

Description:
Returns a pseudo-random number from 0 to 32767. You can optionally provide a seed to generate a new sequence. The pseudo-random numbers generated are NOT cryptographically strong.

READL Subroutine

Usage:
s = READL(n [,size])

Description:
Reads a line of text (or data) from an open file. n must be a value returned by the OPEN subroutine. The optional size value is available in TBSVER 2 or later and limits the amount of data read. On failure this subroutine returns an empty string and sets member .errno containing the failure code to differentiate from a blank line being read.

RENAME Subroutine

Usage:
s = RENAME(oldname, newname)

Description:
Renames a file. This subroutine returns zero on success or a non-zero failure code.

RETURN Keyword

Usage:
RETURN [v]

Description:
Use the RETURN keyword to exit the current subroutine and return to the subroutine that called it. A RETURN statement has the same effect as encountering an END SUB.

If an expression is included after the RETURN keyword, the value of that expression is returned to the calling subroutine. If a value is returned from the MAIN subroutine, that value sets the script return code (errorlevel).

RIGHT Subroutine

Usage:
s = RIGHT(s, n)

Description:
Returns a string with the right-most characters of s. The number of characters to return is indicated by n. If n is greater than or equal to the length of the string, then the entire string is returned. For example, RIGHT(“Test”, 2) returns “st”.

RMDIR Subroutine

Usage:
r=RMDIR(path)

Description:
Removes an empty directory. This subroutine returns zero on success or a non-zero failure code.

RMFILE Subroutine

Usage:
r=RMFILE(filepath)

Description:
Deletes a file. This subroutine returns zero on success or a non-zero failure code.

SEEK Subroutine

Usage:
SEEK(n, offset)

Description:
Jumps to a position within an open file. So that the new position will be used for reading or writing. n is the value returned by OPEN. offset is the location to jump to.

NOTE: Be aware that the file routines translate newline characters to carriage return, line feeds pairs. This means that an offset may not work as expected under some circumstances. SEEK is mostly useful for doing things like moving to the beginning or end of a file.

SETATTR Subroutine

Usage:
r=SETATTR(filepath, attribute)

Description:
Changes the attributes of a file to match attribute. This subroutine returns non-zero on success or zero on failure.

SETDRIVE Subroutine

Usage:
r=SETDRIVE(drvltr)

Description:
Changes the current drive to drvltr. Only the first character is used so "A:" is the same as "A" or "Apple". This subroutine returns zero (FALSE) on error or non-zero (TRUE) on success.

SETENV Subroutine

Usage:
SETENV(env, val)

Description:
Use SETENV to set an environment variable. If the environment variable already exists, the existing variable is modified. Otherwise, it is created.

Example:
```
sub main()
   setenv("path", "C:\")
   printl(getenv("path")
end sub
```

SETLOCALE Subroutine

Usage:
SETLOCALE(locale)

Description:
Use this subroutine to set the current locale. This setting affects the format of date and time strings created by other subroutines.

The locale argument may be any of the following values:

0	Date and time strings will be created using the default format.
1	Date and time strings will be created using the ISO 8601 format.

SETUEFIVAR Subroutine

Usage:
r =UEFISETVAR(varname, namespaceguid[, bindata, attributes])

Description:
Set a UEFI firmware variable (variable names are case sensitive).   This subroutine is only available when the system booted using UEFI. In linux, the efivarfs must be mounted at /sys/firmware/efi/efivars.   The linux command to mount is: mount -t efivarfs none /sys/firmware/efi/efivars. When *bindata* is not provided the variable is deleted. The return value is zero on success otherwise an error code is returned.

WARNING: This function does not prevent you from deleting variables or setting invalid data. Using invalid data or deleing the wrong variables can prevent your system from booting until the firmware is reset to factory defaults. Contact the system manufacturer for instructions on resetting factory defaults.

Example:
```
const UEFI_VAR_NV = 1
const UEFI_VAR_BS = 2
const UEFI_VAR_RT = 4

sub main()
   a = UEFI_VAR_NV+UEFI_VAR_BS+UEFI_VAR_RT // attributes
   t = binary(2, 3, 0, 2)                  // 16-bit value for number 2
   e = setuefivar("Timeout","{8BE4DF61-93CA-11D2-AA0D-00E098032B8C}", t, a)
   if (e = 0) then
     printl("Boot Timeout Set to", binary(t, 3))
   else
     printl("Unable to set Boot Timeout.  Error:", e)
   end if
end sub
```

SLEEP Subroutine

Usage:
SLEEP(seconds)

Description:
Use the SLEEP subroutine to pause for the specified number of seconds. SLEEP returns after the specified number of seconds has passed.

SUB..END SUB Keywords

Usage:
SUB subname
END SUB

Description:
Defines a subroutine with the given name.

Example:
```
sub main()
   printl("In main()")
   test1()
   printl("In main()")
end sub

sub test1()
   printl("In test1()”)
   test2()
   printl("In test1()")
end sub

sub test2()
   printl("In test2()”)
end sub
```

TXASCII Subroutine

Usage:
TXASCII(0|1)

Description:
Enables output of ASCII characters under Windows.

TXCURSORTYPE Subroutine

Usage:
TXCURSORTYPE(0|1|2)

Description:
Sets the shape of the text cursor. 0=None, 1=Block, 2=Underline.

TXGETBLOCK Subroutine

Usage:
b = TXGETBLOCK(x1, y1, x2, y2)

Description:
Returns a reference to a saved area of the text console. The variable value returns 0 or 1 to indicate failure or success.

TXGETINFO Subroutine

Usage:
ti = TXGETINFO()

Description:
Returns information about the current text console. The variable contains the following members:

ViewLeft – X location of the current screen view. 1 = left most position.
ViewTop – Y location of the current screen view. 1 = top most position.
ViewWidth – Width of the current screen view.
ViewHeight – Height of the current screen view.
Width – Width of the entire available text console.
Height – Height of the entire available text console.
Attr – Current text attribute.
CurMode – Current text mode.

TXGOTOXY Subroutine

Usage:
TXGOTOXY(x,y)

Description:
Moves the text cursor to the coordinates x and y. (1,1) is the upper-left most position.

TXINITSubroutine

Usage:
TXINIT()

Description:
Initialize TBScript to use the various text mode subroutines. This must be called at least once before calling any of the other TX based subroutines.

Once this mode is enabled there are some differences that you should note:

1 – GetStr will not output a newline after input (expect under Windows). You must manually do it.
2 – Using a newline (^n) character for output will not include the carriage return under DOS.
3 – Outputting a newline under Linux will clear (using current color) text to the end of the current line.

If these differences are problematic then you'll need to design your own GetStr type subroutine using GetKey(). You can use the TBSENV variable to determine the environment the script is running in.

TXMODE Subroutine

Usage:
TXMODE (m)

Description:
Sets the video text mode. Setting the video mode is only relevant when used in the DOS environment. 0=Black and White 40 columns, 1=Color 40 columns, 2=Black and White 80 columns, 3=Color 80 columns, 7=Monochrome 80 columns, 64=EGA/VGA 43/50 lines.

TXOUTCH Subroutine

Usage:
TXOUTCH(c [,repeat])

Description:
Outputs a character to the current cursor location and optionally repeats it.

TXPUTBLOCK Subroutine

Usage:
r = TXPUTBLOCK(b [,x [,y]])

Description:
Write a blocked of saved text back to the console. If x or y are provided the text is placed at those coordinates otherwise the original location is used. The returned value indicates 0 or 1 to indicate failure or success. The block (b) stays allocated until cleared by assigning another value to it (e.g. b="")

TXSETATTR Subroutine

Usage:
TXSETATTR (attribute)

Description:
Sets the current text attribute to use on the next TX output subroutine. It's common to use a hexadecimal number when specifying attributes due to the clarity it provides. For example, white text on a blue background would be specified as 0x1F (1 being blue and F (15) being white).

Text Attribute 8-Bit Encoding	
Bits	**Usage**
0-3	foreground color (0 to 15)
4-6	background color (0 to 7)
7	blink-enable bit

Standard Colors	
Value	**Description**
0	Black
1	Blue
2	Green
3	Cyan
4	Red
5	Magenta
6	Brown
7	Light Gray
8	Dark Gray
9	Light Blue
10	Light Green
11	Light Cyan
12	Light Red
13	Light Magenta
14	Yellow
15	White

Example:
```
sub main()
   txinit()
   txsetattr(0x1F) // set white text on blue background
   printl("This prints in color")
   txterm()
end sub
```

TXTERM Subroutine

Usage:
TXTERM()

Description:
Terminates the use of the various text console subroutines. This should be called before ending the script if TXINIT() was used.

TXWHEREX Subroutine

Usage:
x =TXWHEREX ()

Description:
Returns the X location of the text cursor. The left most position is 1.

TXWHEREY Subroutine

Usage:
y=TXWHEREY(s)

Description:
Returns the Y location of the text cursor. The top most position is 1.

UCASE Subroutine

Usage:
s = UCASE(s)

Description:
Returns an upper case version of a string.

WHILE..WEND Keywords

Usage:

```
WHILE expression
        statements
WEND
```

Description:
Use a WHILE loop to execute a block of statements as long as an expression is true.

Example:

```
sub main()
   a = 1
   while a <= 25
      printl("This is test line ", a)
      a = a + 1
   wend
end sub
```

WRITEL Subroutine

Usage:
WRITEL(n [, s])

Description:
Writes a line of text to an open file. n is the value returned by OPEN. s is the line of text to write to the file. If s is omitted, a blank line is written to the file. This subroutine returns zero on success or a non-zero failure code.