

TeraByte Scripting Language

Reference

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

Copyright © 2007-2022 TeraByte, Inc. All Rights Reserved

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()
    print1(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 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 than
`<=` 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 extracts 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, optional parameters are assumed to be zero. The values for *type* are as follows: 0=String, 1=Hex String, 2=Wide String, 3=Numeric, 4=Binary, 5=Narrow String. 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, Narrow String in TBSVER 14.

Example:

```

sub main()
  // Note: It's recommended to use type 2 or 5 to know the string type.
  // non-Unicode versions return narrow strings
  bindata=BINARY("STRING")           // 53 54 52 49 4E 47
  // Unicode versions return wide strings
  bindata=BINARY("STRING")           // 53 00 54 00 52 00 49 00 4E 00 47 00
  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 [, changedrive])

Description:

Changes the current directory to the given path. This subroutine returns zero on success or a non-zero failure code. changedrive was added in TBSVer 12 and if provided, the drive is also changed if specified in the path.

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.

CONCTL Subroutine

Usage:

r = CONCTL("command")

Availability:

Version 15 (Windows)

Description:

Allows moving, hiding, and getting the position of the console window. *command* can be one of the following:

show=n – Hide/show console window. Values for n are: 0=hide, 1=show
pos=x,y,w,h – Position console window at x, y with width w and height h.
query – Get position, size, and visible state of console window. Returns the following members:
 .x – horizontal position
 .y – vertical position
 .width – width of window
 .height – height of window
 .shown – visible state (0=hidden, 1=visible)

Note that the size and/or position of the console window may be adjusted to keep the window inside the screen area when it's positioned. If necessary, use the WinGetMonitors subroutine to get the screen area for the monitor(s).

Return value is true/false (1/0) for success/failure of specified command.

Example:

```
sub main()  
  conctl("show=0") // hide console window  
  conctl("pos=100,10,1000,550") // move and resize console window  
  con = conctl("query") // get console info  
  printl("Console position: " # con.x # ", " # con.y)  
  printl("Console size: " # con.width # " x " # con.height)  
  printl("Console visible: " # con.shown)  
end sub
```

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; TBSENVCH contains "UNICODE" if Windows Unicode version; 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:

```
r = EXT(s [, capture [, guiprogvar [, guitextvar]]])
```

Description:

Executes script extensions that may exist in a product. An error code is returned if there is one. The optional *capture* parameter (added in version 15) can be true (1) to have the function return the output from the command; otherwise false (0) is presumed. The return variable has the captured output in the .extcap member (there is a 64K character limit). If a “pause” option is used the output won’t be captured because the command will be interactive (e.g. “dir /p”, “type filename /p”, etc.). The Windows version 16 adds the *guiprogvar* and *guitextvar* parameters for use by commands that provide progress feedback to also update the GUI as well.

Example:

```
sub main()
  ext("extcmd param1 param2")

  // read value from INI file
  r = ext("list ini settings.ini options username", 1)
  p = instr(r.extcap, "^n")
  if (p>1) then
    val = left(r.extcap, p-1)
  else
    val = r.extcap
  end if
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)
    print(f.name)
    f = findnext(f)
  wend
  print(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(datetime, 1, 2)  
  day = mid(datetime, 4, 2)  
  year = mid(datetime, 7, 4)  
  hour = mid(datetime, 12, 2)  
  min = mid(datetime, 15, 2)  
  sec = mid(datetime, 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 variables 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

```

GUIButton Subroutine

Usage:

s = GUIButton(dialog, x, y, width, height, title, ...)

Availability:

Version 15 (Windows)

Description:

Add a button to a dialog. The GUIButton variadic (...) parameters are listed below for quick reference. Additional options, details, and examples can be found in *Windows GUI Controls: Styles & Images Reference*.

Default, CmdLink, DefCmdLink, Split, DefSplit, Flat, Multiline, Push, TxLeft, TxCenter, TxRight, TxBottom, TxTop, TxVCenter, Right, Image, SetImage={metadata}image_file

Example:

```

sub main()
...
    button=GUIButton(dialog, 10, 10, 60, 24, "OK", "SetImage={16}myicon.ico")
...
end sub

```

GUICheckBox Subroutine

Usage:

s = GUICheckBox(dialog, x, y, width, height, title, ...)

Availability:

Version 15 (Windows)

Description:

Add a checkbox to a dialog. The GUICheckBox variadic (...) parameters are listed below for quick reference. Additional options, details, and examples can be found in *Windows GUI Controls: Styles & Images Reference*.

3State

Example:

```

sub main()
...
    checkbox=GUICheckBox(dialog, 10, 10, 150, 24, "CheckBox1")
    GUISetValue(checkbox, 1) // check
...
end sub

```

GUICombo Subroutine

Usage:

s = GUICombo(dialog, x, y, width, height, title, ...)

Availability:

Version 15 (Windows)

Description:

Add a combo control (List and Input combined or dropdown list) to a dialog. The height value is used to determine the size of the dropdown menu when open. The GUICombo variadic (...) parameters are listed below for quick reference. Additional options, details, and examples can be found in *Windows GUI Controls: Styles & Images Reference*.

Simple, Sort, Upper, Lower, Dropdown, DropDownList

Example:

```

sub main()

```



```

...
combo=GUICombo(dialog, 10, 10, 200, 150, "Combo", "dropdown")
GUIInsertItem(combo, "Item 1", 0)
GUIInsertItem(combo, "Item 2", 1)
...
end sub

```

GUIDialog Subroutine

Usage:

s = GUIDialog(dialog, x, y, width, height, title, ...)

Availability:

Version 15 (Windows)

Description:

Create a dialog on which to place controls. The GUIDialog variadic (...) parameters are listed below for quick reference. Additional options, details, and examples can be found in *Windows GUI Controls: Styles & Images Reference*.

Align-Center

The tab order of the controls in the dialog is the order in which they were created. Each control created for a dialog must be assigned to a variable (will be ignored, otherwise). While dialog and control variables are cleaned up when they go out of scope, it's recommended to use the unset keyword to clear the variables when no longer needed. There is a maximum of 10 dialogs at one time, each with a maximum of 200 controls.

Multiple dialog windows can be created and controlled either in the main event loop or their own even loop. Note that you will need to disable the "parent" dialog (or certain controls in it) if you don't want them active when a "child" dialog is open and then reenable the "parent" dialog (or disabled controls) when the "child" dialog closes.

See the GUIEventWait and GUIGetEvent subroutines for additional examples.

Example:

```

sub main()
...
// create dialog
dialog=GUIDialog(10, 10, 600, 400, "My Dialog", "align-center")
...
// event loop
...
// done with controls, clear them
unset dialog
...
end sub

```

GUIEnable Subroutine

Usage:

GUIEnable(dialog_or_control, true_false)

Availability:

Version 15 (Windows)

Description:

Enables or disables the specified dialog or control (0=disable, 1=enable).

Example:

```

sub main()
...
button=GUIButton(dialog, 10, 10, 60, 24, "OK", "default")
GUIEnable(button, 0) // disable button
...
end sub

```

GUIEventWait Subroutine

Usage:

r = GUIEventWait(dialog [, milliseconds])

Availability:

Version 15 (Windows)

Description:

Wait for an event from the dialog or any of its controls. Optionally, specify a timeout period in milliseconds for the subroutine to return if no event has occurred.

Return values: -1 = error, 0 = timeout, 1 = event occurred.

Example:

```
sub main()
  dialog=GUIDialog(10, 10, 600, 400, "My Dialog", "align-center")
  button=GUIButton(dialog, 10, 10, 60, 24, "OK", "default")

  // event loop
  while 1
    event=GUIEventWait(dialog, 5000)
    if event=0 then
      printl("timed out")
    elseif event=1 then
      // process dialog events
      dlgevent=GUIGetEvent(dialog)
      if (dlgevent & 0x8000) then
        exitloop // close requested
      end if
      // process control events
      if (GUIGetEvent(button) & 1) then
        exitloop
      endif
    else
      printl("error on wait")
      exitloop
    end if
  wend

  unset dialog
  unset button
end sub
```

GUIGetEvent Subroutine

Usage:

bits = GUIGetEvent(dialog_or_control)

Availability:

Version 15 (Windows)

Description:

Returns events that have occurred for a given dialog or control since the last time the subroutine was called on it. The return value is bit oriented and consists of:

- 0x0001 – Clicked
- 0x0002 – Double Clicked
- 0x0004 – Changed (not sent for checkbox and radiobutton controls)
- 0x0008 – Pushed
- 0x0010 – Unpushed
- 0x0020 – Got Focus
- 0x0040 – Lost Focus
- 0x0080 – Dropdown selected (e.g., on split button)
- 0x0100 – Dialog: Menu item selected

0x4000 – Dialog: Context menu requested
0x8000 – Dialog: Close request

Note that events may come in together (e.g., 0x21 – Got Focus & Clicked) or separately (e.g., 0x1 Clicked followed by 0x2 Double-clicked).

Example:

```
sub main()
  // inside event loop
  ...
  event=GUIGetEvent(dialog)
  if (event & 0x8000) then
    printl("Close Requested")
  end if

  // context menu requested
  if (event & 0x4000) then
    ctrlid = GUIGetValue(dialog) // get ID of control requesting menu
    if (ctrlid = mylistview) then
      GUIMenu(dialog, -1, -1, 0, "{10}Item A|{11}Item B|{12}Item C")
    end if
  end if

  // process menu items
  if (event & 0x100) then
    menuid = GUIGetValue(dialog) // get ID value of selected menu item
    if (menuid = 20) then
      // process "Item A" from split button menu
    end if
  end if

  // regular button
  if (GUIGetEvent(button) & 1) then
    printl("Button Clicked")
  end if

  // split button
  ev = GUIGetEvent(sbutton)
  if (ev & 1) then
    printl("Button part of split button clicked")
  elseif (ev & 0x80) then
    printl("Dropdown part of split button clicked")
    // display context menu
    p=GUIPos(sbutton) // next line should all be on one line
    GUIMenu(dialog, p.x+p.width-10, p.y+p.height-10, 0,
      "{20}Item A|{21}Item B")
  end if
  ...
end sub
```

GUIGetText Subroutine

Usage:

text = GUIGetText(dialog_or_control)

Availability:

Version 15 (Windows)

Description:

Retrieve the text for a dialog or control that supports text. For DateTime the format used is "YYYY-MM-DD hh:mm:ss". For List control, returns view style (*icon*, *smallicon*, *list*, *details*).

Example:

```
sub main()  
...  
text=GUIGetText(input)  
printl("You entered:" # text)  
...  
end sub
```

GUIGetValue Subroutine

Usage:

value = GUIGetValue(dialog_or_control [, which])

Availability:

Version 15 (Windows)

Description:

Returns the values for a dialog or control that supports values. The *which* parameter is used for a List or Tree control to indicate which data to return ("checked", "selected", or "state=*n*" items). The value returned depends on the control type as follows:

Dialog: Return value depends on last event: Menu selection returns ID of menu item selected. Context menu requested returns the ID of the dialog or control requesting the menu.

DateTime: FILETIME value (64-bit number).

Input: Max characters allowed.

Checkbox: -1=Indeterminate, 0=Not Checked, 1=Checked

RadioButton: -1=Indeterminate, 0=Not Checked, 1=Checked

Progress: Current value (0-100).

Combo: The index of the currently selected item.

List/Tree: The number of member array items returned. The individual values for a List/Tree are returned in a member array. For example, if it returns 2 then there will be a value.[1] and value.[2] to indicate which item values are checked, selected, or match the state.

Example:

```
sub main()  
...  
r=GUIGetValue(checklist, "checked")  
printl("Checked Identifiers:")  
for i=1 to r  
    printl(r.[i])  
next  
...  
r=GUIGetValue(statelist, "state=4")  
printl("Items set to state 4:")  
for i=1 to r  
    printl(r.[i])  
next  
...  
end sub
```

GUIGroupBox Subroutine

Usage:

s = GUIGroupBox(dialog, x, y, width, height, title, ...)

Availability:

Version 15 (Windows)

Description:

Add a group box to a dialog. GUIGroupBox does not have any control-specific variadic (...) parameters. General options, details, and examples can be found in *Windows GUI Controls: Styles & Images Reference*.

Example:

```
sub main()
...
groupbox=GUIGroupBox(dialog, 10, 10, 100, 80, "Group Box")
...
end sub
```

GUIInput Subroutine**Usage:**

s = GUIInput(dialog, x, y, width, height, title, ...)

Availability:

Version 15 (Windows)

Description:

Add an input box (edit control) to a dialog. The GUIInput variadic (...) parameters are listed below for quick reference. Additional options, details, and examples can be found in *Windows GUI Controls: Styles & Images Reference*.

ReadOnly, Password, Multiline, Lower, Upper, Number, TxLeft, TxCenter, TxRight

Example:

```
sub main()
...
input=GUIInput(dialog, 10, 10, 200, 20, "Text", "bgcolor=0", "fgcolor=0xFFFFFFFF")
...
end sub
```

GUIInsertItem Subroutine**Usage:**

GUIInsertItem(control, text, identifier [, iconidx [, after_identifier [, parent_identifier]]])

Availability:

Version 15 (Windows)

Description:

Insert an item into a List, Tree or Combo control.

For a List in detail mode, the *text* parameter can specify multiple column's text separated by a newline character (^n). The *iconidx* parameter can specify multiple column icon indexes separated by a newline character (^n). To skip an icon for a column (first column can't be skipped), use a space between the separators (e.g., to skip the second column: "1^n ^n3" or "1| |3").

For Tree, the *iconidx* parameter can have two values separated by a newline character (^n) to indicate the image and selected image respectively.

For List/Tree controls the *identifier* must be 1 or greater; for a Combo control the *identifier* is the zero-based index to insert at (-1 to add to end).

Note: In addition to the newline character (^n), a vertical bar (|) can be used as the separator for the GUI based subroutines if enabled with the GUIVertBarSep subroutine.

Example:

```
sub main()
...
GUIVertBarSep(1)
GUIInsertItem(list, "Item1_A|Item1_B", 1);
GUIInsertItem(list, "Item3_A|Item3_B", 3, 1);
GUIInsertItem(list, "Item2_A|Item2_B", 2, "2|2", 1);

GUIInsertItem(tree, "Root Item", 1, 0)
GUIInsertItem(tree, "Root Child", 2, "1|0", 0, 1)
```

```

GUIInsertItem(tree, "Child Child", 3, 1, 0, 2)
...
end sub

```

GUIList Subroutine

Usage:

s = GUIList(dialog, x, y, width, height, title, headers, images, ...)

Availability:

Version 15 (Windows)

Description:

Add a list control to a dialog. Columns can be created by passing the *headers* string with optional metadata prefix. The metadata format is {column_width specifiers}. The specifiers control the column's alignment, data type for sorting, and default sort. The alignment options available are L to left justify, R to right justify, or C to center. The data type options are U for unsigned integer, I for a signed integer. The sort options are > to sort from A to Z or < to sort from Z to A. Separate multiple headers in the string with a newline character (^n). The images follow the format of the ImageList found in *Windows GUI Controls: Styles & Images Reference*. The GUIList variadic (...) parameters are listed below for quick reference. Additional options, details, and examples can be found in *Windows GUI Controls: Styles & Images Reference*.

List View styles: Icon, SmallIcon, List, Details

Left, Top, AutoArrange, Sort, SortRev, SingleSel, KeepSel, NoSortHeader, NoHeader, AutoAutoArrange, AutoCheckSelect, AutoSizeColumns, BorderSelect, CheckBoxes, ColumnSnapPoints, DoubleBuffer, FlatSB, FullRowSelect, GridLines, HideLabels, LabelTip, SimpleSelect, SnapToGrid, SubItemImages, TransparentBkgnd, TransparentShadow

TBScript specific: filliconbg, filliconselbg, fillsubitemiconbg, fillsubitemiconselbg

If the style is not specified, Icon is used. The style can be changed after creating the control by using the GUISetText subroutine.

Note: In addition to the newline character (^n), a vertical bar (|) can be used as the separator for the GUI based subroutines if enabled with the GUIVertBarSep subroutine.

Example:

```

sub main()
...
il="{16~small}iconstrip.bmp^ncustom.ico"
list=GUIList(dialog,10,10,200,150,"",{99<}ColA^n{95r}ColB",il,"details","SubItemImages")
GUIInsertItem(list, "Item1 A|Item1 B", 1, "1^n5")
...
// list using 4 states
GUIVertBarSep(1)
il="{state}iconstates4.png" // strip of 4 images
stlist=GUIList(dialog,10,10,200,150,"",{il,"list","checkboxes","singleselect")
GUIInsertItem(stlist, "Item A", 1)
GUIInsertItem(stlist, "Item B", 2)
GUISetValue(stlist, "1", "state=3") // set item ID 1 to state 3
GUISetValue(stlist, "2", "state=2") // set item ID 2 to state 2
...
end sub

```

GUIMenu Subroutine

Usage:

GUIMenu(dialog, x, y, menuflag, "menu_items")

Availability:

Version 15 (Windows)

Description:

Displays a popup/context menu in the specified *dialog*. x and y specify the location of the menu (use -1 for both to have the program determine the location). *menuflag* specifies the Windows TrackPopupMenu() flags to use for the menu (if necessary, refer to <https://docs.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-trackpopupmenu> for details). The metadata format for the menu items is {item_id_and_options} and precedes the menu item text. Separate menu items in the string with a newline character (^n) or vertical bar (|), if enabled. The following options are available:

- Separator line (*underscore character*)

- * Default
- + Checked
- () Radio button (*left & right parenthesis*)
- Disabled (*hyphen character*)
- > Start sub-menu
- < Return from sub-menu (previous menu)
- [Right-to-left text (for right-to-left languages)

If no menu item ID is specified, the ID is the position starting at 1 (a return value of 0 from the menu means no selection was made); for each sub-menu, the starting number is a multiple of the next 100 (e.g., item 1 of a second menu is 101).

To set a hotkey for the menu item precede the character with & (to use the & character in the menu text use &&).

Several example menus are shown below. For an example showing handling menu events see the GUIGetEvent subroutine.

Example:

```
sub main()
...
GUIVertBarSep(1)
// simple menu with position set, separator and checked item
GUIMenu(dialog, 25, 100, 0, "Item &A|Item &B|{_}|{+}&Checked item")
// ID values set, simple sub-menu
GUIMenu(dialog, -1, -1, 0, "{10}Item A|{>}Sub-menu|{12}Sub-menu item")
// shows default, radiobutton, checked, and disabled items
GUIMenu(dialog, -1, -1, 0, "{*}Default|{(+)Radiobutton|{+}Checked|{-}Disabled")
// more complicated example using multiple options (should all be on one line)
GUIMenu(dlg, -1, -1, 0, "{1024}Test Item &1|{>}Test Item &2|
    {*+0x22}On SubMenu 1^ttab|On SubMenu 2|{>}Next|On Next 1|{-}On Next 2|
    {<>}Another submenu|Item 1|{<}|{<}Test Item &3")
...
end sub
```

GUIPos Subroutine

Usage:

```
p = GUIPos(dialog_or_control)
GUIPos(dialog_or_control, new_x [, new_y [, new_width [, new_height]]])
```

Availability:

Version 15 (Windows)

Description:

When specifying just the dialog or control the current position and size is returned (.x, .y, .width, .height). Otherwise, specify the new position and size to move or resize the dialog or control.

Example:

```
sub main()
...
button=GUIButton(dialog, 10, 10, 60, 24, "OK", "default")
p = GUIPos(button) // get button position and size
printl("button (X,Y WxH): " # p.x # ", " # p.y # " " # p.width # "x" # p.height)
GUIPos(button, p.x+10, p.y+10) // reposition button
...
end sub
```

GUIProgress Subroutine

Usage:

```
s = GUIProgress(dialog, x, y, width, height, title, ...)
```

Availability:

Version 15 (Windows)

Description:

Add a progress indicator to a dialog. The GUIProgress variadic (...) parameters are listed below for quick reference. Additional options, details, and examples can be found in *Windows GUI Controls: Styles & Images Reference*.

Marquee, Smooth, SmoothReverse, Vertical

Example:

```
sub main()  
...  
progress=GUIProgress(dialog, 10, 10, 100, 25, "", "smooth")  
GUISetValue(progress, 32) // set position to 32%  
...  
end sub
```

GUIRadioButton Subroutine

Usage:

s = GUIRadioButton(dialog, x, y, width, height, title, ...)

Availability:

Version 15 (Windows)

Description:

Add a radio button to a dialog. The GUIRadioButton variadic (...) parameters are listed below for quick reference. Additional options, details, and examples can be found in *Windows GUI Controls: Styles & Images Reference*.

Group

Example:

```
sub main()  
...  
radio1=GUIRadioButton(dialog, 20, 20, 80, 24, "Radio 1", "group")  
radio2=GUIRadioButton(dialog, 20, 40, 80, 24, "Radio 2")  
...  
end sub
```

GUIRemoveItem Subroutine

Usage:

GUIRemoveItem(control, identifier)

Availability:

Version 15 (Windows)

Description:

Remove an item from a List, Tree or Combo control. Specifying an *identifier* of 0 (zero) for a List/Tree control will remove all items in the control. The *identifier* for a Combo is the zero-based index.

Example:

```
sub main()  
...  
GUIRemoveItem(list, 2)  
...  
end sub
```

GUISetFocus Subroutine

Usage:

GUISetFocus(dialog_or_control)

Availability:

Version 15 (Windows)

Description:

Sets the keyboard focus to the specified dialog or control.

Example:

```
sub main()  
...
```



```

button=GUIButton(dialog, 10, 10, 60, 24, "Start")
GUISetFocus(button)
...
end sub

```

GUISetText Subroutine

Usage:

GUISetText(dialog_or_control, text)

Availability:

Version 15 (Windows)

Description:

Set the text for a dialog or control that supports text. To set the date and time for a DateTime control the format used is "YYYY-MM-DD hh:mm:ss" (when only needing to set the time you must still include a valid date). To set the view style for a List control you can specify one of the following: *icon*, *smallicon*, *list*, *details*.

Example:

```

sub main()
...
// set text for an Input control
GUISetText(input, "New Text")
// set view style for List control
GUISetText(mylist, "details")
...
end sub

```

GUISetValue Subroutine

Usage:

GUISetValue(control, value, ...)

Availability:

Version 15 (Windows)

Description:

Set the value of a control. The value depends on the control type as follows:

- DateTime: FILETIME value (64-bit number)
- Input: Max characters allowed.
- Checkbox: -1=Indeterminate, 0=Not Checked, 1=Checked
- RadioButton: -1=Indeterminate, 0=Not Checked, 1=Checked
- Progress: Value (0-100)
- Combo: The index of the currently selected item or -1 if custom data entered.
- List /Tree: List of identifiers separated by a newline (^n) character or vertical bar (|), if enabled, to set the state of the items.

For the List/Tree state to change is controlled by the variadic (...) parameters:

- "Reset" – Reset the state of all items to opposite of the state to set/clear.
- "Check" – Check item(s).
- "Unchecked" – Uncheck item(s).
- "State=*n*" – Where *n* is the desired state value. The range of *n* depends on how many states were configured (max. 15). The "Reset" parameter is not supported when setting states. You will need to set each item's state to an appropriate value.
- "Select" – Select item(s).
- "Unselect" – Unselect item(s).
- "Focus" – Set focus to item within the control.

Example:

```

sub main()
...
GUISetValue(list, "1^n2", "Check", "Reset")
GUISetValue(checkbox, 1)

```

```

...
GUISetValue(statelist, "1^n3^n5", "state=3")
GUISetValue(statelist, "2^n4", "state=1")
...
end sub

```

GUIShow Subroutine

Usage:

GUIEnable(dialog_or_control, value)

Availability:

Version 15 (Windows)

Description:

Sets the show state of the specified dialog or control. *value* can be any of the SW_ values from the Win32 ShowWindow() function. Note that some states may not be applicable.

0 – SW_HIDE	6 – SW_MINIMIZE
1 – SW_NORMAL	7 – SW_SHOWMINNOACTIVE
2 – SW_SHOWMINIMIZED	8 – SW_SHOWNA
3 – SW_MAXIMIZE	9 – SW_RESTORE
4 – SW_SHOWNOACTIVATE	10 – SW_SHOWDEFAULT
5 – SW_SHOW	11 – SW_FORCEMINIMIZE

Example:

```

sub main()
...
button=GUIButton(dialog, 10, 10, 60, 24, "OK", "default")
GUIShow(button, 0) // hide button
...
end sub

```

GUIStatic Subroutine

Usage:

s = GUIStatic(dialog, x, y, width, height, title, ...)

Availability:

Version 15 (Windows)

Description:

Add text or an image to a dialog box. When "Image" is provided the title is the image file to load. The GUIStatic variadic (...) parameters are listed below for quick reference. Additional options, details, and examples can be found in *Windows GUI Controls: Styles & Images Reference*.

WhiteFrame, WhiteRect, BlackFrame, BlackRect, GrayFrame, GrayRect, Center, Right, RightJust, Simple, Sunken, WordEllipsis, EndEllipsis, PathEllipsis, NoPrefix, NoWrap, Etched, EtchedHor, EtchedVert, EMF, SizeToControl, SizeToImage, CenterImage, Image

Example:

```

sub main()
...
static1 = GUIStatic(dialog, 10, 10, 100, 40, "myimage.bmp", "image")
static2 = GUIStatic(dialog, 10, 60, 100, 40, "My Image", "fontsize=15")
...
end sub

```

GUITimeDate Subroutine

Usage:

s = GUITimeDate(dialog, x, y, width, height, title, ...)

Availability:

Version 15 (Windows)

Description:

Add a date and time picker to a dialog. The GUITimeDate variadic (...) parameters are listed below for quick reference. Additional options, details, and examples can be found in *Windows GUI Controls: Styles & Images Reference*.

LongDate, ShortDate, ShortDate4, ShowNone, Time, RightAlign, UpDown, TimeDate, ShortTime

Example:

```
sub main()  
...  
date=GUITimeDate(dialog, 10, 10, 120, 24, "")  
...  
end sub
```

GUITree Subroutine**Usage:**

s = GUITree(dialog, x, y, width, height, title, images, ...)

Availability:

Version 15 (Windows)

Description:

Add a tree control to a dialog. The images follow the format of the ImageList found in *Windows GUI Controls: Styles & Images Reference*. The GUITree variadic (...) parameters are listed below for quick reference. Additional options, details, and examples can be found in *Windows GUI Controls: Styles & Images Reference*.

Buttons, Lines, CheckBoxes, Edit, FullRow, LinesAtRoot, RTOL, AlwaysSel, SelExpand

Example:

```
sub main()  
...  
GUIVertBarSep(1)  
il="{16}icon0.ico|icon1.ico|icon2.ico"  
tree=GUITree(dialog, 10, 10, 300, 200, "", li, "buttons", "checkboxes")  
GUIInsertItem(tree, "Root Item", 1, 0)  
GUIInsertItem(tree, "Root Child", 2, 1, 0, 1)  
...  
end sub
```

GUIVertBarSep Subroutine**Usage:**

GUIVertBarSep(true_false)

Availability:

Version 15 (Windows)

Description:

Enable or disable the ability to use the vertical bar (|) as the separator for the GUI based subroutines in addition to the newline character (^n).

Example:

```
sub main()  
...  
GUIVertBarSep(1)  
...  
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 [, codepage])

Description:

Use INSTR to find a substring within a string. "codepage", added in version 10, can be a specific code page or 0 for current output code page (only accurate in Windows version).

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
```

ISSTRYPE Subroutine

Usage:

n = ISSTRYPE(s, t)

Description:

ISSTRYPE 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 [, codepage])

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". "codepage", added in version 10, can be a specific code page or 0 for current output code page (only accurate in Windows version).

LEN Subroutine

Usage:

n = LEN(s [, codepage])

Description:

Returns the number of characters in a string. "codepage", added in version 10, can be a specific code page or 0 for current output code page (only accurate in Windows version).

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 [, codepage]])

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. "codepage", added in version 10, can be a specific code page or 0 for current output code page (only accurate in Windows version).

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). Starting in TBSVER 11 sharing options -denynone, -denyall, -denyread, -denywrite can be appended to the methods. The sharing options only take effect in Windows, DOS and any environment under a TBOSDT directly mounted drive (0: - 9:). Example: "uin-out-denywrite"

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 [, codepage]])

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. "codepage", added in version 10, can be a specific code page or 0 for current output code page (only accurate in Windows version).

```
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 or expands the amount of data read. The default size is 512, prior to TBSVER 17 the default size was 128. On failure this subroutine returns an empty string and sets member errno containing the failure code to differentiate from a blank line being read. In text mode, a CRLF is converted to a newline character and returned with the string.

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 [,codepage])

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". "codepage", added in version 10, can be a specific code page or 0 for current output code page (only accurate in Windows version).

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.

SETCP Subroutine

Usage:

r=SETCP(whichcp, cpvalue)

Availability:

Version 10

Description:

Changes the code page for a given item. The values for whichcp are: 0=Get Code Page, 1=Console Code Page, 2=Conversion Code Page. When whichcp is zero the cpvalue indicates which code page to return, either 1

(console) or 2 (conversion). The console values are only accurate under Windows. It's not recommended to change the conversion code page to a value other than 65001 (UTF8) or -1 (Auto Determine) or problems can arise from foreign characters.

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. Note that even with locale set, decimal numbers will continue to accept use of '.'.

The locale argument may be any of the following values:

- | | |
|---|--|
| 0 | Date and time strings will be created using the default format. Locale set to default (Ver 10+). |
| 1 | Default date and time strings will be created using the ISO 8601 format. |
| 2 | Default date and time without zero prefix (Ver 10+). |
| 3 | Default time removing space between time and AM/PM (Ver 10+). |
| 4 | Default time to use lower case AM/PM (Ver 10+). |
| 5 | Locale pulled in from environment (not related to date/time strings) (Ver 10+). |
| 6 | Use OS locale for Date/Time (Ver 10+). |

SETUEFIVAR Subroutine

Usage:

r=UEFISSETVAR(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 deleting 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 | -milliseconds)

Description:

Use the SLEEP subroutine to pause for the specified number of seconds. SLEEP returns after the specified number of seconds has passed. Starting in version 15 you can use a negative value to delay in milliseconds.

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.

UNSET Keyword

Usage:

UNSET v

Description:

Use to clear a variable. Usage is recommended to clear GUI dialog and control variables when no longer needed.

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
```

WinGetMonitors Subroutine**Usage:**

info = WinGetMonitors()

Availability:

Version 15 (Windows)

Description:

Get the Windows screen and monitor information. The return value is set to the number of monitors found and contains the following members:

- .x0 - virtual screen left most
- .x1 - virtual screen right most
- .y0 - virtual screen top most
- .y1 - virtual screen bottom most
- .monitor[] – 1 based array of monitors
 - .x0 - virtual left of monitor
 - .x1 - virtual right of monitor
 - .y0 - virtual top of monitor
 - .y1 - virtual bottom of monitor
 - .flags - monitor flags from DISPLAY_DEVICES.StateFlags
 - .name - monitor name

Example:

```
sub main()
  m=WinGetMonitors()
  printl("virtual screen=" # m.x0 # "," # m.y0 # "," # m.x1 # "," # m.y1)
  for i=1 to m
    // split for this example not to wrap
    print("monitor " # i # "=" # m.monitor[i].x0 # "," # m.monitor[i].y0)
    printl("," # m.monitor[i].x1 # "," # m.monitor[i].y1)
  next
  m=0
end sub
```

WRITEL Subroutine**Usage:**

WRITEL(n [, s [,new_line_type]])

Description:

Writes a line of text or binary data to an open file. n is the value returned by OPEN. s is the line of text or binary data to write to the file. When the file is opened in text mode a CRLF will be appended to the line of text written. Added in version 17, the *new_line_type* controls the type of new line appended. By default, a CRLF combination will be used, when *new_line_type* is 1, a single newline character will be used and when 0, no newline will be added. Stream implementations of this function will use the standard C handling of new lines; for full and consistent control, use binary mode. This subroutine returns zero on success or a non-zero failure code.

Windows GUI Controls: Styles & Images Reference

This section provides a reference for the supported styles, images, and imagelists that can be used with the Windows GUI controls created using TBScript.

Available styles are a sub-set of the standard Windows control styles. Certain styles and style combinations may not be supported by a specific control as determined by Windows. If necessary, more details on Windows controls and their styles can be found on [Microsoft's site](#). Options listed below indicate whether specific to standard **Windows** control styles (names may differ) or **TBScript**.

The following options are available for Windows GUI controls and dialogs:

Font Control
TBScript: FontSize= <i>point_size</i> , Font= <i>font_name</i> , Bold, Italics, Strikeout, Underline
In general, when specifying a font you should also specify the size or it may not take effect. For dialogs, the "Strikeout" and "Underline" options are not supported.
Example: <code>s = GUIStatic(dlg, 10, 10, 60, 16, "my text", "font=calibri", "fontsize=10", "bold")</code>
Ex Styles
Windows: StaticEdge, ClientEdge, WindowEdge
Colors
TBScript: BgColor= <i>rgb</i> , FGColor= <i>rgb</i>
List / View Controls Only: BgColorSel= <i>rgb</i> , FGColorSel= <i>rgb</i> , BgColorKeepSel= <i>rgb</i> , FGColorKeepSel= <i>rgb</i>
<i>rgb</i> in hex format (0xFF0000=Red, 0x00FF00=Green, 0x0000FF=Blue). For some controls you may need to specify both the background and foreground colors to have them show (e.g., foreground color may not show if background not also specified).
For list and tree controls you can specify the background and foreground colors to use for the selected item as well as for the keep selected state (selected state shown when control doesn't have focus).
Example: <code>s = GUIStatic(dlg, 10, 10, 60, 16, "my text", "BgColor=0xCCDDFF", "FGColor=0xFF0000")</code>

The following options are available for the Windows GUI dialogs:

Dialog	Created using: GUIDialog()
TBScript: Align-Center	

The following options are available for the various Windows GUI controls:

Button	Button control	Created using: GUIButton()
Windows: Default, CmdLink, DefCmdLink, Split, DefSplit, Flat, Multiline, Push, TxLeft, TxCenter, TxRight, TxBottom, TxTop, TxVCenter, Right, Image		
TBScript: SetImage={ <i>metadata</i> } <i>image_file</i> , SetImageList={ <i>metadata</i> } <i>image_files</i>		
See below for details on specifying images for buttons.		

Static	Static control	Created using: GUIStatic()
Windows: WhiteFrame, WhiteRect, BlackFrame, BlackRect, GrayFrame, GrayRect, Center, Right, RightJust, Simple, Sunken, WordEllipsis, EndEllipsis, PathEllipsis, NoPrefix, NoWrap, Etched, EtchedHor, EtchedVert, EMF, SizeToControl, SizeTolmage, CenterImage, Image		
Input	Edit control	Created using: GUIInput()
Windows: ReadOnly, Password, Multiline, Lower, Upper, Number, TxLeft, TxCenter, TxRight		
TimeDate	Date and Time Picker control	Created using: GUITimeDate()
Windows: LongDate, ShortDate, ShortDate4, ShowNone, Time, RightAlign, UpDown, TimeDate, ShortTime		
CheckBox	CheckBox control	Created using: GUICheckBox()
Windows: 3State		
RadioButton	RadioButton control	Created using: GUIRadioButton()
Windows: Group		
Progress	Progress Bar control	Created using: GUIProgress()
Windows: Marquee, Smooth, SmoothReverse, Vertical		
Combo	ComboBox control	Created using: GUICombo()
Windows: Simple, Sort, Upper, Lower, Dropdown, DropDownList		
List	List View control	Created using: GUIList()
Windows (List View Styles): Icon, SmallIcon, List, Details		(Icon style is used if no style specified.)
Windows: Left, Top, AutoArrange, Sort, SortRev, SingleSel, KeepSel, NoSortHeader, NoHeader, AutoAutoArrange, AutoCheckSelect, AutoSizeColumns, BorderSelect, CheckBoxes, ColumnSnapPoints, DoubleBuffer, FlatSB, FullRowSelect, GridLines, HideLabels, LabelTip, SimpleSelect, SnapToGrid, SubItemImages, TransparentBkgnd, TransparentShadow		
TBScript: The following options only apply to the “Details” style and fill the area around the icons to match the background color: filliconbg - fills in icon area for unselected items (main items, column 0) filliconselbg - fills in icon area for selected items (main items, column 0) fillsubitemiconbg - fills in icon area for unselected sub-items fillsubitemiconselbg - fills in icon area for selected sub-items		
Tree	Tree View control	Created using: GUITree()
Windows: Buttons, Lines, CheckBoxes, Edit, FullRow, LinesAtRoot, RToL, AlwaysSel, SelExpand		
ImageList	Additional options available for controls using an ImageList (e.g., List, Tree)	
Windows: Mask, ColorDDB, Color8, Color16, Color24, Color32		
Default if none specified is Color32.		

Supported Image File Types

The following image file types are supported: .ICO, .BMP, .PNG, .JPG, .GIF

When specifying an image file, both relative and absolute paths are supported.

List/Tree Controls: Metadata and Image Format for ImageLists

The metadata and image format for ImageList based controls (List/Tree) are provided at the start of each image list with no preceding whitespace. The metadata is placed in between {curly brackets} as follows: [size][~][which]. Either *size* or *which* must be provided. The *which* value for a List can be: *small*, *normal* or *state* and for a Tree: *normal* or *state* and correspond to the following:

- small* - Icons used for List control *list*, *details*, and *smallicon* view styles.
- normal* - Icons used for Tree control and List control *icon* view style.
- state* - Icons used for checked/unchecked states (index 0=unchecked, 1=checked) or for custom multiple states. The control must use the "checkboxes" parameter. The number of images in the *state* imagelist determines the number of states (max. 15). The state value will increment when clicked, returning to 1 after max is reached. For example, if you wish to display custom images for unchecked and checked states the *state* imagelist should have only two images.

The ~ character can be provided to indicate loading of a .bmp should change colors matching the lower-left pixel to match the system Window color (.bmp must be 8-bit or less). If no *which* is provided the default is *normal*. The default icon size is 16 for *state/small* and 32 for *normal*. For .dll and .exe files you must specify the 0-based icon indexes to use. Separate files and indexes with the newline character (^n) or the vertical bar (|), if enabled. If the same *which* value is specified multiple times (including defaults) in the list, only the last one will be used.

If you need to specify a mask for the icons you can provide the mask file just prior to the graphic file, separated with the newline character (^n) or the vertical bar (|), if enabled. The mask file must be a black & white .bmp file renamed with a .bmm extension (black indicates visible area). If less images exist in mask image than graphic file, the additional images will not be masked. Note that any transparency in the graphic file (such as in a PNG) will be removed when a mask is used.

List Examples:

```
// small imagelist from strip
"{small}c:\path\iconstrip.bmp"

// normal and small imagelists from multiple sources
"{32 normal}iconfile1.ico|iconresource.dll|15|{16~small}iconstrip.bmp"

// small imagelist and state icons from strips
"{small}iconstrip.png|{state}checkuncheck.png"

// small imagelist using mask file with .bmp strip
"{small}iconstripmask.bmm|iconstrip.bmp"

// using larger icons for both small and normal/icon view styles
"{32 small}iconstrip32.png|{64 normal}iconstrip64.png"
```

Tree Examples:

```
// imagelist from strip
"{16}iconstrip.bmp"

// imagelist from multiple files
"{16}treestrip.png|treestrip2.png"

// normal size icons pulled from .exe file using indexes 0,1,2,5
"{normal}iconresource.exe|0|1|2|5"

// size 16 imagelist and state icons from strips
"{16}iconstrip.png|{state}iconstate.png"
```

Static and Single Image Button Controls: Metadata and Image Format

The metadata and image format for a Static control or a single image Button control requires a single image to be provided. Specify the size in metadata, if necessary (image will be resized square).

Static/Button (single image) Examples:

```
"SetImage=c:\path\image.bmp"  
"SetImage={16}icon.ico"
```

Multi-image Button Control: Metadata and Image Format for ImageList

The metadata and image format for a multi-image Button control uses the *SetImageList={metadata}image_files* format and requires an image for each button state. The metadata is placed between {curly brackets} as follows: size [align]. The *size* value is the size of the image and must be specified (images should be square). The *align* value is optional and specifies the alignment to use and must be one of the following (default is center): *left*, *right*, *top*, *bottom*. Note that if center alignment is used and button text is included and uses text center alignment (default) that the image and text will overlap.

If an image strip is specified it should contain images for all six button states. If a single image is specified it will be used for all states. If multiple images are specified and an image for a state isn't provided no image will be shown in that state. You can specify the same image for multiple states (for example, you may specify different images for hot and disabled and use the normal image for the other four states). Separate the image files with the newline character (^n) or the vertical bar (|), if enabled. The six button states are as follows and must be specified in this order:

```
NORMAL  
HOT           (hover over)  
PRESSED  
DISABLED  
DEFAULTED    (has keyboard focus)  
STYLUSHOT
```

Button (multi-image, using imagelist) Examples:

```
// different image for each state, left alignment  
"SetImageList={16 left}normal.png|hot.png|pressed.png|disabled.png|defaulted.png|stylushot.png"  
  
// using only normal, hot, disabled, right alignment  
"SetImageList={16 right}button.png|buttonhot.png|button.png|buttondisabled.png|button.png|button.png"  
  
// using single 32 size image for all states, center alignment  
"SetImageList={32}start.png"  
  
// using size 32 strip containing images for all button states, center alignment  
"SetImageList={32}buttonstates.png"
```