PowerView Function Reference

Release 09.2023

Version 09-Oct-2023

History

31-01-2023	Solaris was removed as supported host OS.
11-Nov-2022	New function STRing.ESCapeQuotes().
07-Apr-2022	New function RCL.TCP.NrUsedCons().
07-Apr-2022	New function WINPAGE.LIST().
06-Apr-2022	New function WINPAGE.CURRENT().
05-Apr-2022	New function WINdow.LIST().
28-Mar-2022	New functions: EVAL.ADDRESS(), EVAL.BOOLEAN(), EVAL.FLOAT(), and EVAL.PARAM().
03-Mar-2022	New functions: CONVert.LINEAR11TOFLOAT() and CONVert.LINEAR16TOFLOAT().
20-Jan-2022	New function: STRing.TOKEN().
07-Jul-2021	New function MENU.EXIST().
26-Mar-2021	New function math.FMOD().

In This Document

This document lists all the *host-related functions*. These functions return information about the operating system and the TRACE32 PowerView GUI.

Each system has additional system-specific functions:

- **General Function Reference**
- **Stimuli Generator Function Reference**

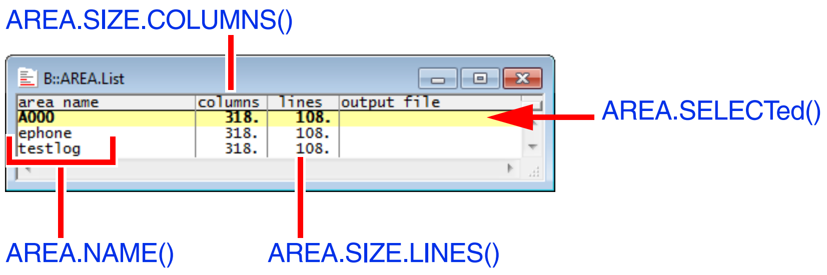
The following generic functions are available for all systems. The capital letters represent the short form of the function. Any function name can be used in its long or short form.

For example: **CONV**ert**.S**igned**L**ong() can be abbreviated to **CONV.SL**()

For more information about the long and short form, see **"Long Form and Short Form of Commands and Functions"** (ide_user.pdf).

AREA Functions

This figure provides an overview of the return values of some of the functions. For explanations of the illustrated functions and the functions not shown here, see below.



```
area name          columns   lines   output file
A000                  318.    108.
ephone                318.    108.
testlog               318.    108.
```

In This Section

See also

- ■ AREA
- ❏ AREA.COUNT()
- ❏ AREA.EXIST()
- ❏ AREA.LINE()
- ❏ AREA.MAXCOUNT()
- ❏ AREA.NAME()
- ❏ AREA.SELECTed()
- ❏ AREA.SIZE.COLUMNS()
- ❏ AREA.SIZE.LINES()

AREA.COUNT() Number of existing message areas

[build 79795 - DVD 02/2017]

Syntax: **AREA.COUNT()**

Returns the number of message areas which exist in the TRACE32 instance. After starting TRACE32 you will have just one message area, but you can create additional message areas with the command **AREA.Create**.

Return Value Type: Decimal value.

AREA.EXIST() Check if message area exists

[build 79795 - DVD 02/2017]

Syntax: **AREA.EXIST(*<area_name>*)**

Returns TRUE if a message area with the given name exists.

Parameter Type: String.

Return Value Type: Boolean.

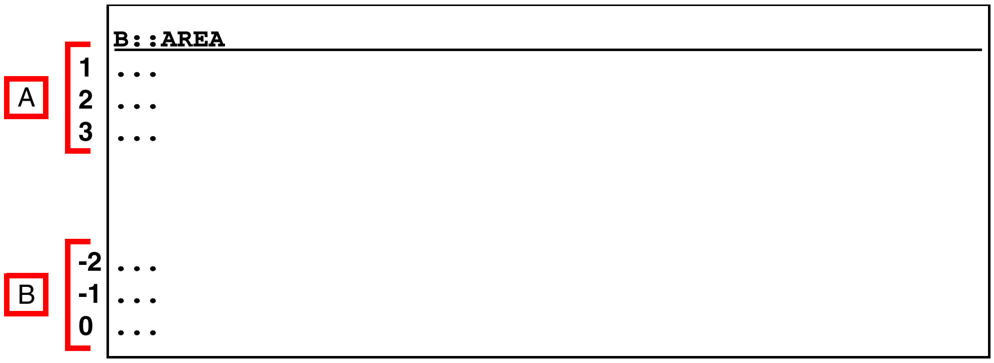
AREA.LINE() Extract line from message area

[build 79895 - DVD 02/2017]

Syntax: **AREA.LINE(***<area_name>*,*<line>***)**

Returns one line from a message area.

In order to use the function, you do not need to open the **AREA** window of a message area. You just need to open an **AREA** window if you want to display the message area output on screen.



A Positive numbers for *<line>* starting with 1 identify lines from the top of the message area.

B 0 and negative numbers for *<line>* identify lines from the bottom of the message area.

Parameter and Description:

<area_name>	**Parameter Type**: String.
<line>	**Parameter Type**: Decimal value.

Return Value Type: String.

AREA.MAXCOUNT() Maximal number of message areas

[build 79795 - DVD 02/2017]

Syntax: **AREA.MAXCOUNT()**

Returns the maximum number of message areas which can be created in your version of TRACE32.

Return Value Type: Decimal value.

AREA.NAME() Names of existing message areas

[build 79795 - DVD 02/2017] [Go to figure]

Syntax: **AREA.NAME(*<index>*)**

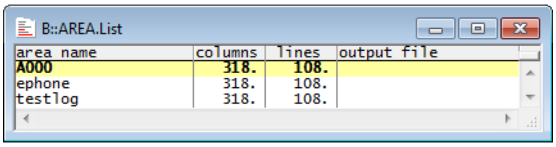
Returns the name of a message area specified by an *<index>*. The *<index>* corresponds to the line number in the **AREA.List** window. The *<index>* of the first line is 0.

Parameter Type: Decimal value.

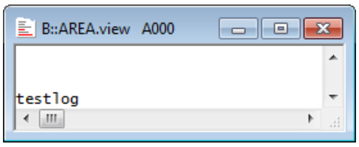
Return Value Type: String.

Example: In addition to the default message area A000, two more message areas are created. The function **AREA.NAME()** is then used to return the name of the message area that has the *<index>* 2.

```
AREA.Create ephone        ;create the message areas 'ephone'
AREA.Create testlog       ;and 'testlog'
AREA.List                 ;overview of existing message areas

AREA.Select A000          ;select the default message area A000 for output
AREA.view   A000          ;display the AREA window A000
PRINT AREA.NAME(2)        ;print the AREA window name that has the index 2
                          ;to the AREA window A000
                          ;(here: 'testlog', see AREA.List window)
```



```
area name   columns  lines  output file
A000           318.   108.
ephone         318.   108.
testlog        318.   108.
```



```
testlog
```

AREA.SELECTed() Name of active message area

[build 80344 - DVD 02/2017] [Go to figure]

Syntax: **AREA.SELECTed()**

Returns the area name of the currently active message area. A message area is made the active message area with the command **AREA.Select**.

Return Value Type: String.

AREA.SIZE.COLUMNS() Columns of a message area

[build 80344 - DVD 02/2017] [Go to figure]

Syntax: **AREA.SIZE.COLUMNS(***<area_name>***)**

Returns the number of columns a message area was created with. You can specify the number of columns with the **AREA.Create** command.

Parameter Type: String.

Return Value Type: Decimal value.

AREA.SIZE.LINES() Lines of a message area

[build 80344 - DVD 02/2017] [Go to figure]

Syntax: **AREA.SIZE.LINES(***<area_name>***)**

Returns the number of lines a message area was created with. You can specify the number of lines with the **AREA.Create** command.

Parameter Type: String.

Return Value Type: Decimal value.

CLOCK Functions

The **CLOCK.*()** functions have been renamed to **DATE.*()** functions, see **“DATE Functions”**, page 32.

CONFIG.SCREEN() Check if screen output is switched on

[build 22971 - DVD 11/2010]

Syntax:	**CONFIG.SCREEN()**

Returns FALSE if the screen output is switched off inside the file `config.t32` with `SCREEN=OFF`

Return Value Type: Boolean.

CONVert Functions

In This Section

See also

CONVert.ADDRESSTODUALPORT() Dualport access class

[build 75614 - DVD 02/2016]

Syntax: **CONVert.ADDRESSTODUALPORT(***<address>***)**

Converts an address into the dualport access class if possible. If the access class was already a dualport access class, the originally specified access class is returned.

Parameter Type: Address.

Return Value Type: Address.

Example:

```
PRINT ADDRESS.ADDRESSTODUALPORT(D:0x1000)     ; returns ED:0x1000
PRINT ADDRESS.ADDRESSTODUALPORT(ED:0x1000)    ; returns ED:0x1000
PRINT ADDRESS.ADDRESSTODUALPORT(ETB:0x0)      ; returns EETB:0x0
PRINT ADDRESS.ADDRESSTODUALPORT(EETB:0x0)     ; returns EETB:0x0
```

CONVert.ADDRESSTONONSECURE() Non-secure access class

32-bit and 64-bit ARM cores [build 75614 - DVD 02/2016]

Syntax: **CONVert.ADDRESSTONONSECURE(*<address>*)**

Converts an address into the non-secure access class if possible.

Parameter Type: Address.

Return Value Type: Address.

Example:

```
PRINT CONVert.ADDRESSTONONSECURE(AHB:0x0)     ; returns NAHB:0x0
PRINT CONVert.ADDRESSTONONSECURE(ZAHB:0x0)    ; returns NAHB:0x0
PRINT CONVert.ADDRESSTONONSECURE(NAHB:0x0)    ; returns NAHB:0x0
```

CONVert.ADDRESSTOSECURE() Secure access class

32-bit and 64-bit ARM cores [build 75614 - DVD 02/2016]

Syntax: **CONVert.ADDRESSTOSECURE(*<address>*)**

Converts an address into the secure access class if possible.

Parameter Type: Address.

Return Value Type: Address.

CONVert.BOOLTOINT() Boolean to integer

[build 21439 - DVD 04/2010]

Syntax: **CONVert.BOOLTOINT(*<bool>*)**

Converts a boolean value to an integer.

Parameter Type: Boolean.

Return Value Type: Hex value.

- TRUE becomes 1
- FALSE becomes 0

CONVert.CHAR() Integer to ASCII character

Syntax: **CONVert.CHAR(*<integer>*)**

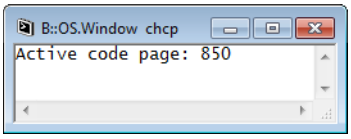
Converts an integer to an ASCII character. For values from 0 to 127, the result is an ASCII character on all operating systems. For values from 128 to 255, the result depends on the font setting in the config.t32. On Windows, the result additionally depends on the active console code page.

Parameter Type: Hex or decimal value.

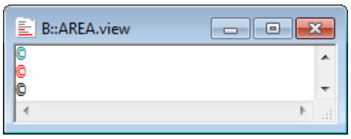
Return Value Type: ASCII value.

Example: If the number of the active console code page is 850, then the integers **`0xA9`** or **`169.`** are converted to the copyright character.

```
;the copyright (c) is returned as an ASCII value
PRINT %COLOR.TEAL CONVert.CHAR(169.)
PRINT %COLOR.RED  CONVert.CHAR(0xA9)

;an ASCII value can be converted to a string by concatenating the ASCII
;value with a string, an empty string in this example
PRINT ""+CONVert.CHAR(169.)
```



```
Active code page: 850
```



```
©
©
©
```

CONVert.FLOATTOINT() Float to integer

Syntax: **CONVert.FLOATTOINT(<*float*>)**

Converts a float value to an integer value.

Parameter Type: Float.

Return Value Type: Decimal value.

Example:

```
PRINT CONVert.FLOATTOINT(1.8887)                ; result 1
```

CONVert.HEXTOINT() Hex to integer

[build 64298 - DVD 09/2015]

Syntax: **CONVert.HEXTOINT(<*hex*>)**

Converts a hex value to an integer value. The function is the counterpart of **CONVert.INTTOHEX()**.

Parameter Type: Hex value.

Return Value Type: Decimal value.

Examples:

```
PRINT CONVert.HEXTOINT(0x42)                    ; result 66
```

Instead of using **CONVert.HEXTOINT()** you can also add 0. to a hex value to get an integer value.

```
PRINT 0x42+0.                                   ; the result is also 66
```

CONVert.INTTOBOOL() Integer to boolean

[build 50832 - DVD 02/2014]

Syntax: **CONVert.INTTOBOOL(*<integer>*)**

Converts an integer to a boolean value.

Parameter Type: Decimal or hex value.

Return Value Type: Boolean.

- FALSE if value==0
- TRUE if value!=0

CONVert.INTTOFLOAT() Integer to floating point value

[build 64326 - DVD 09/2015]

Syntax: **CONVert.INTTOFLOAT(*<integer>*)**

Converts an integer to a floating point value. The function is the counterpart of **CONVert.FLOATTOINT()**.

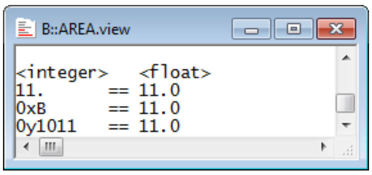
Instead of using **CONVert.INTTOFLOAT()**, you can also add 0.0 to an integer value to get a floating point value.

Parameter Type: Decimal or hex value.

Return Value Type: Float.

Examples:

```
PRINT "<integer>   <float>"
PRINT "11.       == " CONVert.INTTOFLOAT(11.)
PRINT "0xB       == " CONVert.INTTOFLOAT(0xB)
PRINT "0y1011    == " CONVert.INTTOFLOAT(0y1011)
```



```
<integer>    <float>
11.       == 11.0
0xB       == 11.0
0y1011    == 11.0
```

CONVert.INTTOHEX() Integer to hex

Syntax: **CONVert.INTTOHEX(*<integer>*)**

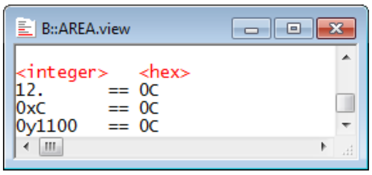
Converts an integer to a hex value.

Parameter Type: Decimal or hex or binary value.

Return Value Type: Hex value.

Examples:

```
PRINT %COLOR.RED "<integer>   <hex>"
PRINT "12.       == " CONVert.INTTOHEX(12.)
PRINT "0xC       == " CONVert.INTTOHEX(0xC)
PRINT "0y1100    == " CONVert.INTTOHEX(0y1100)
```



```
<integer>   <hex>
12.      == 0C
0xC      == 0C
0y1100   == 0C
```

CONVert.INTTOMASK() Compose bit-mask from integer value and mask

Syntax: **CONVert.INTTOMASK(*<value>*,*<mask>*)**

In TRACE32, there is a special data type which realize masks. Bit and hex masks differ only in the input format. The function converts two integers to a bitmask.
The first parameter stands for the value bits and the second parameter defines the mask bits.

Parameter and Description:

<value>	**Parameter Type**: Decimal or hex or binary value.
<mask>	**Parameter Type**: Decimal or hex or binary value.

Return Value Type: Bit mask.

Examples:

```
; all examples represent the same mask value and print 0y0101xxxx
; bitmask: 0y0101XXXX
; hexmask: 0x5X
; bits 6 and 4 must be 1
; bits 3…0      are don't care
PRINT 0x5X
PRINT CONVert.INTTOMASK(0y01010000,0y00001111)
PRINT CONVert.INTTOMASK(0x50,0x0f)
PRINT CONVert.INTTOMASK(0y01010000,0x0f)
PRINT CONVert.INTTOMASK(0x50,0y00001111)
PRINT CONVert.INTTOMASK(0y01010000,15.)
PRINT CONVert.INTTOMASK(80.,0y00001111)
```

CONVert.LINEAR11TOFLOAT() LINEAR11 to float

[build 128910 - DVD 02/2021]

Syntax: **CONVert.LINEAR11TOFLOAT(***<value>***)**

Converts a LINEAR11 (11bit signed mantissa / 5bit signed exponent) PMBus value to a floating point value.

Parameter Type: Decimal value.

Return Value Type: Float.

CONVert.LINEAR16TOFLOAT() LINEAR16 to float

[build 128910 - DVD 02/2021]

Syntax: **CONVert.LINEAR16TOFLOAT(***<mantssa>,<exponent>***)**

Converts a LINEAR16 (16bit signed mantissa / 5 bit signed exponent) PMBus value to a floating point value.

Parameter Type: Decimal value.

Return Value Type: Float.

CONVert.MASKMTOINT()

Syntax: **CONVert.MASKMTOINT(*<mask_value>*)**

Converts a mask to an integer and returns the mask part.

Parameter Type: Bit or hex mask.

Return Value Type: Hex value.

Examples:

```
; bitmask: 0y0101XXXX
; hexmask: 0x5X
; bits 6 and 4 must be 1
; bits 3…0     are don't care
PRINT CONVert.MASKMTOINT(0y0101XXXX)    ; prints 0f
PRINT CONVert.MASKMTOINT(0x5X)          ; prints 0f

; bitmask: 0y01XXXX10
; bits 6 and 2 must be 1
; bits 5…2     are don't care
PRINT CONVert.MASKMTOINT(0y01XXXX10)    ; prints 3c
```

CONVert.MASKTOINT() Bits set to 1 in given bit-mask

Syntax: **CONVert.MASKTOINT(*<value>*)**

Converts a bitmask to an integer and **CONVert.MASKTOINT()** returns the value part.

Parameter Type: Bit or hex mask.

Return Value Type: Hex value.

Examples:

```
; bitmask: 0y0101XXXX
; hexmask: 0x5X
; bits 6 and 4 must be 1
; bits 3…0      are don’t care
PRINT CONVert.MASKTOINT(0y0101XXXX)     ; prints 50
PRINT CONVert.MASKTOINT(0x5X)           ; prints 50

; bitmask: 0y01XXXX10
; bits 6 and 2 must be 1
; bits 5…2      are don’t care
PRINT CONVert.MASKTOINT(0y01XXXX10)     ; prints 42
```

CONVert.OCTaltoint() Octal to decimal

[build 46260 - DVD 08/2013]

Syntax: **CONVert.OCTaltoint("*<string>*")**

Converts a string of octal digits into a number.

Parameter Type: String.

Return Value Type: Decimal value.

Examples:

```
PRINT CONVert.OCTaltoint("71")                 ; prints 57

PRINT "0y" %BINary CONVert.OCTaltoint("71")   ; prints 0y00111001
```

CONVert.SignedByte() 1 byte to 8 bytes

Syntax: **CONVert.SignedByte(*<value>*)**

Converts a byte to a quad word with sign extension. A quad word has eight bytes.
Only the lowest byte (B0) from *<value>* is evaluated and all higher bytes will be ignored.

Parameter Type: Decimal or hex or binary value.

Return Value Type: Hex value.

Examples:

```
PRINT CONVert.SignedByte(0x70)                 ; result               70
PRINT CONVert.SignedByte(0x80)                 ; result 0ffffffffffffff80
PRINT CONVert.SignedByte(0x12345670)           ; result               70
PRINT CONVert.SignedByte(0x12345680)           ; result 0ffffffffffffff80
PRINT CONVert.SignedByte(0y101010000000)       ; result 0ffffffffffffff80

Data.dump Register(bp)+CONVert.SignedByte(Register(bx))
```

CONVert.SignedLong() 4 bytes to 8 bytes

Syntax: **CONVert.SignedLong(*<value>*)**

Converts a long word (four bytes) to a quad word with sign extension. A quad word has eight bytes.
Only the lowest four bytes (B0…B3) from *<value>* are evaluated and all higher bytes will be ignored.

Parameter Type: Decimal or hex or binary value.

Return Value Type: Hex value.

Examples:

```
PRINT CONVert.SignedLong(0x7766554433221100)   ; result         33221100
PRINT CONVert.SignedLong(0x7766554483221100)   ; result 0ffffffff83221100
PRINT CONVert.SignedLong(0x70561234)           ; result         70561234
PRINT CONVert.SignedLong(0x80561234)           ; result 0ffffffff80561234
```

CONVert.SignedWord() 2 bytes to 8 bytes

Syntax: **CONVert.SignedWord(*<value>*)**

Converts a word (two bytes) to a quad word with sign extension. A quad word has eight bytes.
Only the lowest two bytes (B0 and B1) from *<value>* are evaluated and all higher bytes will be ignored.

Parameter Type: Decimal or hex or binary value.

Return Value Type: Hex value.

Examples:

```
PRINT CONVert.SignedWord(0x7012)                 ; result             7012
PRINT CONVert.SignedWord(0x8012)                 ; result 0ffffffffffff8012
PRINT CONVert.SignedWord(0x12347056)             ; result             7056
PRINT CONVert.SignedWord(0x12348056)             ; result 0ffffffffffff8056
PRINT CONVert.SignedWord(0y00010010001101001000000001010110)
                                                 ; result 0ffffffffffff8056

Data.dump Register(bp)+CONVert.SignedWord(Register(bx))
```

CONVert.TIMEMSTOINT() Time to milliseconds

Syntax: **CONVert.TIMEMSTOINT(*<time>*)**

Converts *<time>* to milliseconds.

Parameter Type: Time value.

Return Value Type: Decimal value.

Example:

```
PRINT CONVert.TIMEMSTOINT(1.8887s)            ; result 1888
```

CONVert.TIMENSTOINT() Time to nanoseconds

[build 66884 - DVD 02/2016]

Syntax: **CONVert.TIMENSTOINT(*<time>*)**

Converts *<time>* to nano seconds.

Parameter Type: Time value.

Return Value Type: Decimal value.

Example:

```
PRINT CONVert.TIMENSTOINT(1.8887s)          ; result 1888700000
```

CONVert.TIMERAWTOINT() Time to TRACE32 timer ticks

[build 110672 - DVD 09/2019]

Syntax: **CONVert.TIMERAWTOINT(*<time>*)**

Converts *<time>* to TRACE32 timer ticks. One tick = 78.125ps

Parameter Type: Time value.

Return Value Type: Hex value.

CONVert.TIMESTOINT() Time to seconds

Syntax: **CONVert.TIMESTOINT(*<time>*)**

Converts *<time>* to seconds.

Parameter Type: Time value.

Return Value Type: Decimal value.

Example:

```
PRINT CONVert.TIMESTOINT(150000ms)          ; result 150
```

CONVert.TIMEUSTOINT() — Time to microseconds

Syntax: **CONVert.TIMEUSTOINT(<*time*>)**

Converts <*time*> to micro seconds.

Parameter Type: Time value.

Return Value Type: Decimal value.

Example:

```
PRINT CONVert.TIMEUSTOINT(20.1234s)        ; result 20123400
```

CONVert.TOLOWER() String to lower case

[build 27143 - DVD 06/2011]

Syntax: **CONVert.TOLOWER("*<string>*")**

Converts a string to lower case.

Parameter Type: String.

Return Value Type: String.

Example:

```
PRINT CONVert.TOLOWER("aBcDeF")             ; result abcdef
```

CONVert.TOUPPER() String to upper case

Syntax: **CONVert.TOUPPER("*<string>*")**

Converts a string to upper case.

Parameter Type: String.

Return Value Type: String.

Example:

```
PRINT CONVert.TOUPPER("aBcDeF")             ; result ABCDEF
```

DATE Functions

In This Section

See also

■ DATE	❏ DATE.DATE()	❏ DATE.DAY()	❏ DATE.MakeUnixTime()
❏ DATE.MONTH()	❏ DATE.SECONDS()	❏ DATE.TIME()	❏ DATE.TimeZone()
❏ DATE.UnixTime()	❏ DATE.UnixTimeUS()	❏ DATE.utcOffset()	❏ DATE.YEAR()

DATE.DATE() Current date

Syntax: **DATE.DATE()**
CLOCK.DATE() (deprecated)

Returns the current date.

Return Value Type: String.

Example:

```
;returns the current date, e.g. 5. Aug 2015
PRINT DATE.DATE()

;equivalent to the previous example:
PRINT FORMAT.UnixTime("j. M Y",DATE.UnixTime(),DATE.utcOffset())
```

DATE.DAY() Today's date

[build 13598 - DVD 10/2008]

Syntax: **DATE.DAY()**
CLOCK.DAY() (deprecated)

Returns the today's date.

Return Value Type: Decimal value.

Syntax:	**DATE.MakeUnixTime(***<year>***,***<month>***,***<day>***,***<hour>***,***<minute>***,***<second>***)**

Creates a Unix timestamp from a human readable date *given in UTC*. You can also use a local date, but then you have to subtract your local offset from UTC afterwards, e.g. with **DATE.utcOffset()**.

Parameter and Description:

<year>	**Parameter Type**: Decimal value. Four-digit representation of a year (e.g. **2000.** or **2004.**).
<month>	**Parameter Type**: Decimal value. Numeric representation of a month (**1.** … **12.**).
<day>	**Parameter Type**: Decimal value. Day of the month (**1.** … **31.**).
<hour>	**Parameter Type**: Decimal value. 24-hour format of an hour (**0.** … **23.**).
<minute>	**Parameter Type**: Decimal value. Minutes (**0.** … **59.**)
<second>	**Parameter Type**: Decimal value. Seconds (**0.** … **59.**)

NOTE:	Remember to append a dot so that the arguments are interpreted as decimal constants.

Return Value Type: Decimal value.

Examples:

```
;converts 21-July-2015, 12:45 given in UTC to the Unix time 1437482700
PRINT DATE.MakeUnixTime(2015.,7.,21.,12.,45.,0.)

;returns -11644473600.
PRINT DATE.MakeUnixTime(1601.,1.,1.,0.,0.,0.)

;returns the Unix time of last midnight
PRINT DATE.MakeUnixTime(DATE.YEAR(),DATE.MONTH(),DATE.DAY(),0.,0.,0.)
```

DATE.MONTH() Number of current month

[build 12210 - DVD 10/2008]

Syntax:	**DATE.MONTH()**
	CLOCK.MONTH() (deprecated)

Returns the number of the current month (1 to 12).

Return Value Type: Decimal value.

DATE.SECONDS() Seconds since midnight

Syntax:	**DATE.SECONDS()**
	CLOCK.SECONDS() (deprecated)

Returns the time since midnight in seconds.

Return Value Type: Decimal value.

DATE.TIME() Current time

Syntax:	**DATE.TIME()**
	CLOCK.TIME() (deprecated)

Returns the current time.

Return Value Type: String.

Example:

```
;returns the current time, e.g. 18:35:02
PRINT DATE.TIME()

;equivalent to the previous example:
PRINT FORMAT.UnixTime("H:i:s",DATE.UnixTime(),DATE.utcOffset())
```

DATE.TimeZone()   Time zone identifier and hh:mm:ss

Syntax: **DATE.TimeZone()**

Returns a three to five letter time zone identifier and the UTC offset in hh:mm:ss. The UTC offset is positive if the time zone is west of UTC and negative if east of UTC.

Return Value Type: String.

Example:

```
PRINT DATE.TimeZone()                            ; returns: CEST-02:00:00
```

DATE.UnixTime()   Seconds since Jan 1970

[build 64298 - DVD 09/2015]

Syntax: **DATE.UnixTime()**
CLOCK.UNIX() (deprecated)

Returns the time in UNIX format (in seconds since Jan 1970).

Return Value Type: Decimal value.

DATE.UnixTimeUS()   Microseconds since Jan 1970

[build 115649 - DVD 02/2020]

Syntax: **DATE.UnixTimeUS()**

Returns the elapsed Microseconds since the Unix epoch (1 January 1970 00:00:00 UTC) without leap seconds.

Return Value Type: Decimal value.

DATE.utcOffset() Offset of current local time to UTC

[build 64298 - DVD 09/2015]

Syntax: **DATE.utcOffset()**

The offset of the current local time to UTC, including an offset caused by daylight saving time. As in ISO 8601, positive values are east of UTC. Negative values are west of UTC.

Return Value Type: Decimal value.

Example:

```
;prints 7200. during summer in central Europe
PRINT DATE.utcOffset()

;prints the current local time, e.g. 12:03:52
PRINT FORMAT.UnixTime("H:i:s",DATE.UnixTime(),DATE.utcOffset())
```

DATE.YEAR() Current year

[build 12210 - DVD 10/2008]

Syntax: **DATE.YEAR()**
CLOCK.YEAR() (deprecated)

Returns the current year.

Return Value Type: Decimal value.

DIALOG Functions

In This Section

See also

■ DIALOG ❏ DIALOG.BOOLEAN() ❏ DIALOG.EXIST() ❏ DIALOG.STRing()
❏ DIALOG.STRing2()

DIALOG.BOOLEAN() Current boolean value of checkbox

Syntax: **DIALOG.BOOLEAN(*<label>*)**

Returns the current value of a dialog element of the type boolean, e.g. a checkbox.

Parameter Type: String. A user-defined label identifying a dialog element.

Return Value Type: Boolean.

Example:

```
DIALOG.view
(       ; define checkbox
        POS 33. 2. 10.
HEX:    CHECKBOX "HEX Value" "GOSUB SelectedOrCleared"
)
STOP

; <your_code>


SelectedOrCleared:
   ; checks whether the CHECKBOX with the label HEX was activated by the
   ; user or not

   IF DIALOG.BOOLEAN(HEX)==TRUE()
      PRINT "Selected"
   ELSE
      PRINT "Cleared"

RETURN
```

Syntax: **DIALOG.EXIST(*<label>*)**

Returns TRUE if a certain dialog element exists, FALSE otherwise.

Parameter Type: String. A user-defined label identifying a dialog element.

Return Value Type: Boolean.

Example:

```
DIALOG
(
     NAME "MyDlg"                 ; name of dialog window
     POS 1. 0.25 10.
HEX: CHECKBOX "HEX Value" ""      ; labeled checkbox, empty command
     POS 1. 1.5 10.
     BUTTON "Ok" "CONTinue"       ; continue in script
     CLOSE
     (
       DIALOG.END                 ; close and destroy dialog window
       CONTinue                   ; continue in script
     )
)
STOP
…
IF WINDOW.EXIST("MyDlg")          ; check if dialog window still exists
(
  IF DIALOG.EXIST(“HEX”)          ; check if label exists
  (
    PRINT "HEX value checked: " DIALOG.BOOLEAN("HEX”)
  )
  DIALOG.END                      ; close dialog now
)
…
ENDDO
```

DIALOG.STRing()

Current string value of dialog element, e.g. EDIT box

Syntax: **DIALOG.STRing(*<label>*)**

Returns the current string value of a dialog element, such as **EDIT**, **DYNTEXT**, or **DYNLTEXT**. If the dialog element is a list that supports multiple selections, then **DIALOG.STRing()** returns the currently *selected values*.

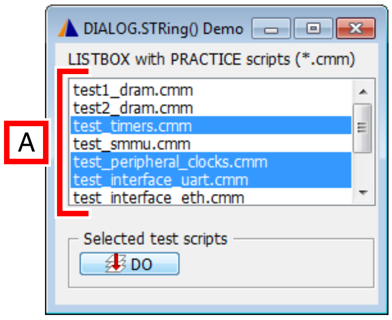
Parameter Type: String. User-defined label identifying a dialog element.

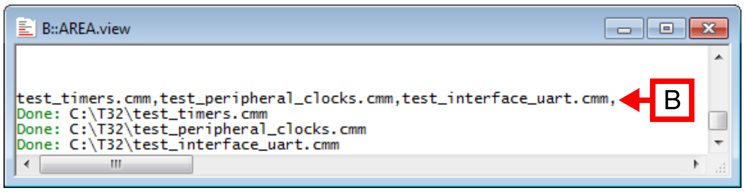
Return Value Type: String. Depending on the type of the dialog element, the string is an individual string or a parameter array consisting of multiple comma-separated values.

Example 1: The function **DIALOG.STRing()** is used to check whether the user has entered a value in the **EDIT** box or not.

```
DIALOG.view
(
        POS 17. 2. 15. 1.
CE:     EDIT "" ""
        POS 33. 2. 10.
HEX:    CHECKBOX "HEX Value" ""
)
…
; checks whether the EDIT box labeled CE is empty or not
IF DIALOG.STRing(CE)!=""
…
```

Example 2: A user has made multiple selections in a **LISTBOX**, a dialog element that supports multiple selections. Using the function **DIALOG.STRing()**, the selected values can be returned as a comma-separated parameter array. In addition, the example shows how to loop through the elements of the parameter array. In this example, the elements are file names of PRACTICE script test cases.





```
test_timers.cmm,test_peripheral_clocks.cmm,test_interface_uart.cmm,
Done: C:\T32\test_timers.cmm
Done: C:\T32\test_peripheral_clocks.cmm
Done: C:\T32\test_interface_uart.cmm
```

A The dialog element **LISTBOX** is labeled **myTESTS**.

B Parameter array returned by `DIALOG.STRing(myTESTS)+","`

```
&list=DIALOG.STRing(myTESTS)+","   ;return the list of selected test
                                   ;cases (*.cmm) as a parameter array
AREA.view
PRINT "&list"                      ;print the list to the AREA window

;loop through the list of selected test cases and scan for the first
;comma-separator
WHILE STRing.SCAN("&list",   ",",   0.)>-1.
(
   PRIVATE &testCase               ;declare macro for WHILE block

   ;split list at the first comma-separator to get the first test case
   &testCase=STRing.Split("&list",   ",",   0.)

   DO ~~\&testCase                 ;execute the first test case

   ;remove the first test case and its comma-separator from the list
   ;by replacing test case and comma-separator with an empty string ""
   &list=STRing.Replace("&list",   "&testCase,",   "",   0.)
)
```

DIALOG.STRing2() Comma-separated list of values, e.g. from LISTBOX

Syntax: **DIALOG.STRing2(*<label>*)**

Retrieves the *complete list of values* from a list dialog element, e.g. from a **LISTBOX**, **MLISTBOX**, **DLISTBOX**, and **COMBOBOX**.

Parameter Type: String. User-defined label identifying a list dialog element.

Return Value Type: String. The string is a parameter array containing all list items as comma-separated values.

ERROR Functions

The **ERROR** functions give access to an information structure of PRACTICE which contains data of the last occurred error.

The error structure can be cleared by the command **ERROR.RESet**.

In This Section

See also

❏ ERROR.ADDRESS()	❏ ERROR.CMDLINE()	❏ ERROR.FIRSTID()	❏ ERROR.ID()
❏ ERROR.MESSAGE()	❏ ERROR.OCCURRED()	❏ ERROR.POSITION()	

ERROR.CMDLINE() Erroneous command

[build 110397 - DVD 09/2019]

Syntax:	**ERROR.CMDLINE()**

Returns the command line content of the last occurred error. The buffer can be deleted with the command **ERROR.RESet**.

Return Value Type: String.

Example: See **ERROR.ID()**.

ERROR.FIRSTID() ID of first error

[build 105170 - DVD 02/2019]

Syntax:	**ERROR.FIRSTID()**

Return ID of first error encountered after last error reset.

Return Value Type: String.

Syntax: **ERROR.ID()**

Returns the search item string of the last error message for the online help. The search item can be deleted with the command **ERROR.RESet**.

Return Value Type: String.

Example:

```
  ERROR.RESet     ; clear PRACTICE error structure

l_system_up:
  SYStem.Up
  IF ERROR.OCCURRED()
  (
;    check for target power fail
     IF ERROR.ID()=="#emu_errpwrf"
     (
;       PRINT     "Please power up the target board!"
        DIALOG.OK "Please power up the target board!"
        GOTO      l_system_up
     )
     ELSE IF ERROR.ID()!=""
     (
        PRINT     "other error occurred: " ERROR.ID()
     )
     OPEN  #1 my_errorlog.txt  /Create /Write
     WRITE #1 "error - faulty cmd:" ERROR.CMDLINE()
     IF ERROR.POSITION()!=-1.
        WRITE #1 "                   ^">>ERROR.POSITION()
     WRITE #1 "error - message   :" ERROR.MESSAGE()
     CLOSE #1
  )
```

ERROR.MESSAGE() Error text

[build 110397 - DVD 09/2019]

Syntax:	**ERROR.MESSAGE()**

Returns the error message text if an error occurred since the last TRACE32 software start or since the last error structure reset by the command **ERROR.RESet**.

Return Value Type: String.

Example: See **ERROR.ID()**.

ERROR.OCCURRED() Error status

[build 76289 - DVD 09/2016]

Syntax:	**ERROR.OCCURRED()**

Returns TRUE if an error occurred since the last TRACE32 software start or since the last error structure reset by the command **ERROR.RESet**.

Return Value Type: Boolean.

Example: See **ERROR.ID()**.

ERROR.POSITION() Error position

[build 110397 - DVD 09/2019]

Syntax:	**ERROR.POSITION()**

Returns the error position inside the command line if an error occurred since the last TRACE32 software start or since the last error structure reset by the command **ERROR.RESet**.

Return Value Type: Decimal value.

-1	Indicates an unknown or an undetermined error position.

Example: See **ERROR.ID()**.

EVAL Functions

In This Section

See also

■ Eval	❏ EVAL()	❏ EVAL.ADDRESS()	❏ EVAL.BOOLEAN()
❏ EVAL.FLOAT()	❏ EVAL.PARAM()	❏ EVAL.STRing()	❏ EVAL.TIme()
❏ EVAL.TYPE()			

EVAL() Value of expression evaluated with Eval command

[build 17384 - DVD 12/2009]

Syntax:	**EVAL()**

Returns the value of the expression parameter from the last **Eval** command. Only for expression types boolean, binary, hex, integer and ASCII constant. For all other expression types the return value is 0.

Return Value Type: Hex value.

EVAL.ADDRESS() Address of expression evaluated with Eval cmd.

[build 125135 - DVD 02/2021]

Syntax:	**EVAL.ADDRESS()**

Returns the value of the expression parameter from the last **Eval** command. Only for expression type address. In all other cases the returned result is empty.

Return Value Type: Address.

EVAL.BOOLEAN() Boolean expression evaluated with Eval cmd. boolean

[build 125135 - DVD 02/2021]

Syntax:	**EVAL.BOOLEAN()**

Returns TRUE if the type of the expression parameter from the last **Eval** command is boolean.

Return Value Type: Boolean.

EVAL.FLOAT() Float value of expression evaluated with Eval cmd.

[build 125135 - DVD 02/2021]

Syntax: **EVAL.FLOAT()**

Returns the value of the expression parameter from the last **Eval** command. Only for expression type float. In all other cases the returned result is empty.

Return Value Type: Float.

EVAL.PARAM() Expression evaluated with Eval cmd.

[build 125139 - DVD 02/2021]

Syntax: **EVAL.PARAM()**

Returns the value of the expression parameter from the last **Eval** command, independently of the expression type.

Return Value Type: String.

EVAL.STRing() String composed by expression evaluated with Eval cmd.

[build 17384 - DVD 12/2009]

Syntax: **EVAL.STRing()**

Returns the value of the expression parameter from the last **Eval** command. Only for expression type string. In all other cases the returned string is empty.

Return Value Type: String.

EVAL.TIme() Value of time evaluated with Eval command

[build 110621 - DVD 09/2019]

Syntax: **EVAL.TIme()**

Returns the time value as evaluated by the last **Eval** command.

Return Value Type: Time value.

EVAL.TYPE()

Syntax: **EVAL.TYPE()**

Returns the type of the expression parameter from the last **Eval** command.

Return Value Type: Hex value.

Return Values	**Expression Types**
0x0001	Boolean
0x0002	binary value
0x0004	Hex value
0x0008	Decimal value
0x0010	Float
0x0020	ASCII value
0x0040	String
0x0080	Numeric range
0x0100	Address
0x0200	Address range
0x0400	Time value
0x0800	Time range
0x4000	Bit or Hex mask
0x8000	Empty/No expression parameter

Example:

```
ENTRY &delayvalue
Eval  &delayvalue                    ; evaluate user input value

IF EVAL.TYPE()!=0x400                ; time value entered?
   GOSUB err_no_timevalue
```

FALSE Function

FALSE() Boolean expression

[build 36180 - DVD 02/2012]

Syntax: **FALSE()**

Returns always the boolean value FALSE. It can be used for increasing the readability of PRACTICE scripts when initializing PRACTICE macros. The counterpart is **TRUE()**.

Return Value Type: Boolean.

Example:

```
&s_error_occurred=FALSE()            ; instead of
&s_error_occurred=(0!=0)
```

FILE Functions

In This Section

See also

❏ FILE.EOF()	❏ FILE.EOFLASTREAD()	❏ FILE.EXIST()	❏ FILE.OPEN()
❏ FILE.SUM()	❏ FILE.TYPE()	❏ __FILE__()	❏ __LINE__()

__FILE__() Path and file name of current PRACTICE script

[build 13023, DVD 10/2008]

Syntax: **__FILE__()**

An alias for **OS.PresentPracticeFile()**.

__LINE__() Number of script line to be executed next

[build 13023, DVD 10/2008]

Syntax: **__LINE__()**

Returns the line number of the command to be executed next in the currently active PRACTICE script.

FILE.EOF() Check if end of read-in file has been reached

[build 31361 - DVD 06/2011]

Syntax: **FILE.EOF(***<file_number>***)**

Function returns a boolean whether the **last READ** command **from a certain file** reached the file end or not.

Parameter Type: Decimal value.

Return Value Type: Boolean.

Example:

```
OPEN #1 myfile.txt   /Read
OPEN #2 yourfile.txt /Read
READ #2 %LINE &myline
READ #1 %LINE &myline1
WHILE !FILE.EOF(2)                ; EOF of yourfile.txt reached?
(
   PRINT "&myline"
   …
   READ #2 %LINE &myline          ; assigns all characters up to the
)                                 ; next EOL to &myline
CLOSE #2
```

FILE.EOFLASTREAD() Check if last read from file reached the end of the file

[build 31361 - DVD 06/2011]

Syntax: **FILE.EOFLASTREAD()**

EOF() - (deprecated)
[build 12285 - DVD 10/2008]

Function returns a boolean whether the **last READ** command reached the file end or not.

Return Value Type: Boolean.

Example:

```
OPEN #2 myfile.txt /Read
READ #2 %LINE &myline
WHILE !FILE.EOFLASTREAD()             ;EOF of myfile.txt reached?
(
   PRINT "&myline"
   …
   READ #2 %LINE &myline              ;assigns all characters up to the
)                                     ;next EOL to &myline
CLOSE #2
```

FILE.EXIST() Check if file exists

[build 31361 - DVD 06/2011]

Syntax: **FILE.EXIST(*<file>*)**

Returns TRUE if the file exists. Alias for **OS.FILE.EXIST()**.

Parameter Type: String.

Return Value Type: Boolean.

FILE.OPEN() Check if file is open

[build 36378 - DVD 08/2012]

Syntax: **FILE.OPEN(*<file_number>*)**

Returns TRUE if a file with *<file_number>* was opened with the command **OPEN** (and not yet closed).

Parameter Type: Decimal value.

Return Value Type: Boolean.

FILE.SUM() Get checksum from a file

[build 99438 - DVD 09/2018]

Syntax: **FILE.SUM()**

Gets the checksum of the last executed **SHA1SUM** command.

Return Value Type: String.

Example:

```
PRIVATE &sha1 &file
&file="myfile.bin"
&sha1="739078942296c3fe61dabca810ed4483d0f79885
&sha1="&sha1 3e99463f765b53348d5e0cf31c8c63d2acf5d81b"
&sha1="&sha1 8450ad62442629453331baf8b4345d1c77b73b2d"
SILENT.SHA1SUM "&file"
IF STRing.SCAN("&sha1",FILE.SUM(),0)==-1
    DIALOG.OK "Cecksum of file ""&file"" not known!"
```

FILE.TYPE()

File type of loaded file

Syntax: **FILE.TYPE(***<file>***)**

Detects the data format of the specified file. The detection algorithm is the same as used for **Data.LOAD.auto**. The returned string is the format name as used with **Data.LOAD.<file_format>**. If the file format is unknown, the function returns “BINARY”.

Parameter Type: String.

Return Value Type: String.

Examples:

```
LOCAL &file &format
&file="~~/demo/arm/compiler/arm/thumbbe.axf"
&format=FILE.TYPE("&file")

IF "&format"=="ELF"
  Data.LOAD.Elf "&file" /NoCODE
ELSE
  PRINT %ERROR "Error: Wrong file format, debug symbols not loaded!"
```

FORMAT Functions

In This Section

See also

❏ FORMAT.BINary()	❏ FORMAT.CHAR()	❏ FORMAT.Decimal()	❏ FORMAT.DecimalU()
❏ FORMAT.DecimalUZ()	❏ FORMAT.FLOAT()	❏ FORMAT.HEX()	❏ FORMAT.STRing()
❏ FORMAT.TIME()	❏ FORMAT.UDecimal()	❏ FORMAT.UDECIMALZ()	❏ FORMAT.UnixTime()

FORMAT.BINary() Numeric to binary value (leading spaces)

Syntax: **FORMAT.BINary(***<width>***,***<number>***)**

Formats a numeric expression to a binary number and generates an output string with a fixed length of *<width>* with leading zeros.
Values which needs more characters than *<width>* for their loss-free representation are not cut.

Parameter and Description:

<width>	**Parameter Type**: Hex or decimal value.
<number>	**Parameter Type**: Hex or decimal value. Numeric expression to be formatted.

Return Value Type: String.

Examples:

```
PRINT FORMAT.BINARY(8.,0x10)                    ; result "00010000"
PRINT FORMAT.BINARY(1.,0x10)                    ; result "10000" and not "0"
```

Syntax: **FORMAT.CHAR(***<value>***,***<width>***,***<fill_character>***)**

Converts an integer value to an ASCII character and generates an output string in a fixed length of *<width>*.

Parameter and Description:

<value>	**Parameter Type**: Hex or decimal value. Integer value to be converted.
<width>	**Parameter Type**: Hex or decimal value. Specifies the number of characters. When the output string length *<width>* is larger than 1 the output string will be enlarged by adding a necessary number of *<fill_character>*. • **0** : empty string returned • **1**..n : length of output string (left aligned) • **-1**..-n: length of output string (right aligned)
<fill_character>	**Parameter Type**: ASCII value. Defines the fill character.

Return Value Type: String.

Examples:

```
PRINT FORMAT.CHAR(0x61,0.,'-')                 ; result ""
PRINT FORMAT.CHAR(0x61,1.,'-')                 ; result "a"
PRINT FORMAT.CHAR('B',1.,'-')                  ; result "B"
PRINT FORMAT.CHAR(' '+35.,1.,'-')              ; result "C"
```

A positive value of *<width>* means left alignment.
A negative value of *<width>* means right alignment of *<value>*.

```
PRINT FORMAT.CHAR(0x61,10.,'*')        ; result "a*********"
PRINT FORMAT.CHAR(0x61,-10.,'*')       ; result "*********a"
PRINT FORMAT.CHAR(0x61,-10.,' ')       ; result "         a"
```

Syntax: **FORMAT.Decimal(*<width>*,*<number>*)**

Formats a numeric expression to a decimal number and generates an output string with a fixed length of *<width>* with leading spaces.
Values which needs more characters than *<width>* for their loss-free representation aren't cut.

Parameter and Description:

<width>	**Parameter Type**: Hex or decimal value. Specifies the smallest number of digits.
<number>	**Parameter Type**: Hex or decimal value. Numeric expression to be formatted.

It is recommended to use the string "." as suffix when the number is printed. This allows the user to identify the number clearly as a decimal number.

Return Value Type: String.

Examples:

```
PRINT FORMAT.Decimal(1.,0x12345)          ; result "74565" and not "5"
PRINT FORMAT.Decimal(8.,0x12345)          ; result "   74565"
PRINT FORMAT.Decimal(8.,0x12)+"."         ; result "      18."
```

```
&i=0                                  ; register R0…R31 will be set to
WHILE  &i<32.                         ; values 0x00000000…0x1f1f1f1f
(
  ; the macro &regname will contain the full register name e.g. R14
  &regname="R"+FORMAT.Decimal(1+((&i/10.)%1),&i)

  ; the macro &regno will contain the register number only e.g. 14
  ; &regno=FORMAT.Decimal(1+((&i/10.)%1),&i)
  &value=&i|&i<<8.
  &value=&value|&value<<16.
  Register.Set &regname &value

  ; Register.Set R&regno  &value
  &i=&i+1
)
```

FORMAT.DecimalU()   Numeric to unsigned decimal as string (leading spaces)

Syntax: **FORMAT.DecimalU(*<width>*,*<number>*)**

Formats a numeric expression to an unsigned decimal number and generates an output string with a fixed length of *<width>* with leading spaces.
Values which needs more characters than *<width>* for their loss-free representation aren't cut.

Alias for **FORMAT.UDECIMAL()**.

Parameter and Description:

<width>	**Parameter Type**: Hex or decimal value. Specifies the smallest number of digits.
<number>	**Parameter Type**: Hex or decimal value. Numeric expression to be formatted.

It is recommended to use the string "." as suffix when the number is printed. This allows the user to identify the number clearly as a decimal number.

Return Value Type: String.

Examples:

```
PRINT FORMAT.DecimalU(8.,0x12345)     ; result  "   74565"
PRINT FORMAT.DecimalU(8.,0x12)+"."    ; result  "      18."
PRINT FORMAT.DecimalU(1.,-1.)         ; result  "18446744073709551615"
                                      ; and not "5"
```

FORMAT.DecimalUZ() Numeric to unsigned decimal as string (leading zeros)

Syntax: **FORMAT.DecimalUZ(*<width>*,*<number>*)**

Formats a numeric expression to a fixed width unsigned decimal number with leading zeros.
Values which need more characters than *<width>* for their loss-free representation are not cut. Alias for **FORMAT.UDECIMALZ()**.

Parameter and Description:

<width>	**Parameter Type**: Hex or decimal value. Specifies the smallest number of digits.
<number>	**Parameter Type**: Hex or decimal value. Numeric expression to be formatted.

Return Value Type: String.

Example 1:

```
PRINT FORMAT.DecimalUZ(8.,0x1235)    ; result  "00004661"
PRINT FORMAT.DecimalUZ(1.,0x1235)    ; result  "4661" and not "1"
PRINT FORMAT.DecimalUZ(1.,-1.)       ; result  "18446744073709551615"
                                     ; and not "5"
```

Example 2: This PRACTICE script example counts from 00001 to 00108. The result is displayed in the default **AREA** window. To try this script, copy it to a `test.cmm` file, and then run it in TRACE32 (See "**How to...**").

```
LOCAL &idCounter
&idCounter=0x0

AREA.RESet    ;initialize AREA system
AREA.view     ;open the default AREA window

RePeat 108.
(;increment counter in steps of 1
  &idCounter=&idCounter+0x1
  PRINT FORMAT.DecimalUZ(5.,&idCounter)
)
```

To increase or decrease the number of lines displayed in an **AREA** window, use **AREA.Create**.

FORMAT.FLOAT() Floating point value to string

Syntax: **FORMAT.FLOAT(***<width>***,***<precision>***,***<number>***)**

Formats a floating point value to a text string. If not mentioned otherwise, the resulting string is right-aligned and padded with blank spaces so that the string has at least the number of characters specified with parameter *<width>*. The resulting string is not truncated even if the result is longer.

Parameter and Description:

<width>	**Parameter Type**: Hex or decimal value. • **> 0:** Minimum number of characters of the resulting string. • **0:** The resulting string is left-aligned and variable length. It contains the specified number of *<precision>* digits.
<precision>	**Parameter Type**: Hex or decimal value. • **> 0:** Number of digits after the decimal point. • **0:** Standard notation with variable number of precision digits. • **-1:** Scientific notation with variable number of precision digits.
<number>	**Parameter Type**: Float. The value to be formatted and displayed.

Return Value Type: String.

Examples:

```
PRINT FORMAT.FLOAT(0.,3.,12.34567)       ; result "12.346"
PRINT FORMAT.FLOAT(0.,5.,12.34567)       ; result "12.34567"
PRINT FORMAT.FLOAT(10.,3.,12.34567)      ; result "    12.346"
PRINT FORMAT.FLOAT(10.,5.,12.34567)      ; result "  12.34567"

PRINT FORMAT.FLOAT(12.,0.,129345.67)     ; result "   129345.67"
PRINT FORMAT.FLOAT(12.,-1.,129345.67)    ; result "129.34567e+3"

PRINT FORMAT.FLOAT(13.,-1.,129345.67E+3) ; result " 129.34567e+6"
```

Syntax: **FORMAT.HEX(***<width>***,***<number>***)**

Formats a numeric expression to a hexadecimal number and generates an output string with a fixed length of *<width>* with leading zeros if necessary.
Values which need more characters than *<width>* for their loss-free representation are not cut.

Parameter and Description:

<width>	**Parameter Type**: Hex or decimal value. Specifies the smallest number of digits.
<number>	**Parameter Type**: Hex or decimal value. Numeric expression to be formatted.

Return Value Type: String.

Example:

```
PRINT FORMAT.HEX(1.,12345.)                ; result "3039" and not "9"
PRINT FORMAT.HEX(8.,12345.)                ; result "00003039"
PRINT "0x"+FORMAT.HEX(8.,12345.)           ; result "0x00003039"
```

It is recommended to use the string “0x” as a prefix when the number is printed. This allows the user to clearly identify the number as a hex number.

Syntax: **FORMAT.STRing(***<source_string>***,***<width>***,***<fill_character>***)**

Generates an output string with a fixed length of *<width>*.

Parameter and Description:

<source_string>	**Parameter Type**: String. Strings that are too long will be cut. When the source string length is smaller than *<width>*, the output string will be enlarged by adding a necessary number of *<fill_character>*.
<width>	**Parameter Type**: Hex or decimal value. Specifies the number of characters. A positive value of *<width>* means left alignment of *<source_string>*. A negative value of *<width>* means right alignment of *<source_string>*. • **0** : empty string returned • **1**..n : length of output string (left aligned) • **-1**..-n: length of output string (right aligned)
<fill_character>	**Parameter Type**: ASCII value. Defines the fill character.

Return Value Type: String.

Examples:

```
PRINT FORMAT.STRing("abcdef",0.,'-')        ; result ""
PRINT FORMAT.STRing("abcdef",3.,'-')        ; result "abc"
PRINT FORMAT.STRing("abcdef",-3.,'-')       ; result "def"

PRINT FORMAT.STRing("abcdef",10.,'*')       ; result "abcdef****"
PRINT FORMAT.STRing("abcdef",-10.,'*')      ; result "****abcdef"
PRINT FORMAT.STRing("abcdef",-10.,' ')      ; result "    abcdef"
```

FORMAT.TIME() Time to string (leading spaces)

[build 70825 - DVD 9/2016]

Syntax: **FORMAT.TIME(***<width>***,***<precision>***,"***<unit>***",***<time>***)**

Formats a time value to a text string. If not mentioned otherwise, the resulting string is right-aligned and padded with blank spaces so that the string has at least the number of characters specified with parameter *<width>*. The resulting string is not truncated even if the result is longer.

Parameter and Description:

<width>	**Parameter Type**: Hex or decimal value. • **> 3:** Minimum number of characters of the resulting string, including two characters for the unit. • **<=3:** The resulting string is left-aligned and variable length. It contains the specified number of *<precision>* digits.
<precision>	**Parameter Type**: Hex or decimal value. Specifies the number of digits after the decimal point. If the value is zero, an integer value is displayed.
<unit>	**Parameter Type**: String. The time unit used for the resulting string. For a list of the available parameters, click here.
<time>	**Parameter Type**: Time value. The value to be formatted and displayed.

Parameters for <unit>:

ns	The unit of the resulting string is unit nanoseconds.
us	The unit of the resulting string is unit microseconds.
ms	The unit of the resulting string is unit milliseconds.
s	The unit of the resulting string is unit seconds. The output string is padded with one blank space at the right side for proper formatting/alignment with other units.
ks	The unit of the resulting string is unit kiloseconds.
auto	The unit is chosen automatically so that the value is 1000 > value >= 1.

Return Value Type: String.

FORMAT.UDECIMAL() Refer to FORMAT.DecimalU()

[build 25777 - DVD 11/2010]

Syntax: **FORMAT.UDecimal(***<width>***,***<number>***)**

Alias for **FORMAT.DecimalU()**.

FORMAT.UDECIMALZ() Refer to FORMAT.DecimalUZ()

[build 25777 - DVD 11/2010]

Syntax: **FORMAT.UDECIMALZ(*<width>*,*<number>*)**

Alias for **FORMAT.DecimalUZ()**.

FORMAT.UnixTime() Format Unix timestamps

[build 64298 - DVD 09/2015] [Examples]

Syntax: **FORMAT.UnixTime(*<formatstr>*,*<timestamp>*,*<utc_offset>*)**

Formats a Unix timestamp the same way PHP's data function formats timestamps.
See also http://php.net/manual/en/function.date.php

Parameter and Description:

<formatstr>	**Parameter Type**: String. Describes how to format the timestamp in *<timestamp>*. For a list of the available parameters, click here.
<timestamp>	**Parameter Type**: Decimal value. • The minimum year is **-67768100567971200.** The maximum year is **67767976233532799.** • Arguments for *<timestamp>* can also be other functions, e.g. **DATE.UnixTime()** or **OS.FILE.UnixTime()**. • As on 64-bit Linux systems, the year zero is taken into account (which is the astronomical year numbering).
<utc_offset>	**Parameter Type**: Decimal value. Is optional and describes the offset of a local time-zone to UTC. Set *<utc_offset>* to zero if you want a result in UTC or set it to an appropriate value if you prefer local times.

Parameters for <formatstr>:

a	Return value: *am* or *pm* in lower case.
A	Return value: *AM* or *PM* in upper case.
b	Seconds since midnight with leading zeros (Return values: 00000 … 86399). [build 69588 - DVD 02/2016] (TRACE32 specific parameter, not supported by PHP)
B	Unlike the PHP date function, **B** (swatch Internet time) is not supported.
c	ISO 8601 date (Return value example: "2015-07-20T11:46:51+00:00").
C	ISO 8601 date with trailing Z if *<utc_offset>* == 0 (Return value example: "2015-07-15T11:53:22Z").

d	Day of the month with leading zeros (Return values: 01 … 31).
D	Three-letter representation of the day of the week (Return values: Mon … Sun).
F	Long representation of a month (Return values: January … December).
g	12-hour format of an hour (Return values: 1 … 12).
G	24-hour format of an hour (Return values: 0 … 23).
h	12-hour format of an hour with leading zeros (Return values: 01 … 12).
H	24-hour format of an hour with leading zeros (Return values: 00 … 23).
i	Minutes with leading zeros (Return values: 00 … 59).
I	Unlike the PHP date function, I (daylight saving time flag) is not supported.
j	Day of the month (Return values: 1 … 31).
J	Day of the month with leading space (1 to 31).
l	Long representation of the day of the week (Return values: Monday … Sunday).
L	Returns 1 if it is a leap year, 0 otherwise.
m	Numeric representation of a month with leading zeros (Return values: 01 … 12).
M	Three-letter representation of a month (Return values: Jan … Dec).
n	Numeric representation of a month (Return values: 1 … 12).
N	ISO 8601 numeric representation of the day of the week (Return values: 1 for Monday, …, 7 for Sunday).
o	ISO 8601 year number. This has the same value as Y, except that if the ISO week number (W) belongs to the previous or next year, that year is used instead.
O	Greenwich mean time difference (GMT) in hours (Return value example: +0200).
P	Greenwich mean time difference (GMT) with colon between hours and minute (Return value example: +02:00).
r	RFC 2822 formatted date (Return value example: "Mon, 20 Jul 2015 17:02:08 +0200").
s	Seconds with leading zeros (Return values: 00 … 59).
S	English ordinal suffix for the day of the month, two characters (Return values: st, nd, rd or th). Can be combined with **j**.
t	Number of days in the given month (Return values: 28 … 31).
T or **e**	Timezone identifier (Return value examples: CET, PST, UTC).

u	Unlike the PHP date function, **u** (microseconds) is not supported.
U	Seconds elapsed since 1970-January-1,00:00:00 GMT.
w	Numeric representation of the day of the week (Return values: 0 for Sunday, …, 6 for Saturday).
W	ISO 8601 week number of year, weeks starting on Monday (Return values: 0 … 52)
y	Two-digit representation of a year (Return value examples: 00 or 04).
Y	Four-digit representation of a year (Return value examples: 2000 or 2004).
z	Day of the year (Return values: 0 … 365).
Z	Time zone offset in seconds. The offset for time zones west of UTC is negative, and for those east of UTC positive (Return values: -43200 … 50400).

Return Value Type: String.

Unlike the PHP date function, all other letter characters are replaced by a question mark.

To get a letter character in the output string you have to prefix it with a backslash. (e.g. "\Y\e\a\r: Y")

All non-letter characters (like punctuation marks or digits) do not require a backslash. However, if you like to have a backslash in the output, you have to prefix the backslash with a backslash.

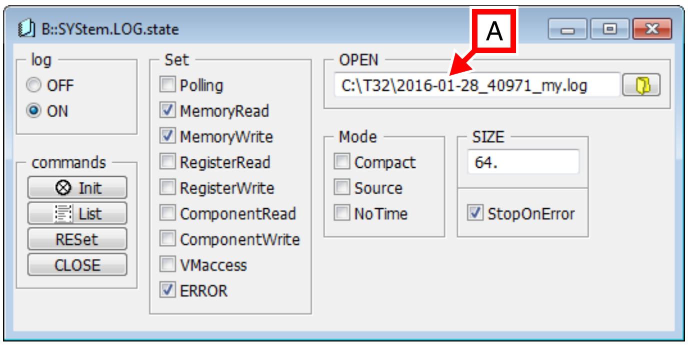
Examples

Example 1:

```
;current time of UTC
PRINT FORMAT.UnixTime("H:i:s",DATE.UnixTime(),0)

;current local date
PRINT FORMAT.UnixTime("d.m.Y",DATE.UnixTime(),DATE.utcOffset())

;current date of UTC in ISO 8601
PRINT FORMAT.UnixTime("c",DATE.UnixTime(),0)

;current date of Pacific Standard Time in RFC1123
PRINT FORMAT.UnixTime("D, d M Y H:i:s O",DATE.UnixTime(),-8*3600.)
```

Example 2: The following PRACTICE script shows how to precede a file name with a date-timestamp:

```
LOCAL &file
&file="_my"

;let's include seconds after midnight in the file name string as a
;simple precaution against duplicate log file names,
;see parameter b in the FORMAT.UnixTime() function below

;concatenate date-timestamp and file name
&file=FORMAT.UnixTime("Y-m-d_b", DATE.UnixTime(), 0.)+"&file"

;open the system log file for writing
SYStem.LOG.state
SYStem.LOG.OPEN  ~~\&file
```



B::SYStem.LOG.state

log: OFF [✓ ON]

Set: Polling, [✓ MemoryRead], [✓ MemoryWrite], RegisterRead, RegisterWrite, ComponentRead, ComponentWrite, VMaccess, [✓ ERROR]

OPEN: C:\T32\2016-01-28_40971_my.log

Mode: Compact, Source, NoTime

SIZE: 64.

[✓ StopOnError]

A System log file with date-timestamp. The file extension .log is added by default.

FOUND Functions

In This Section

See also

■ FIND	■ WinFIND	❏ FOUND()	❏ FOUND.COUNT()

FOUND() TRUE() if search item was found

Syntax: **FOUND()**

Boolean function set by **Data.Find**, **Trace.Find**, **FIND**, **ComPare** or memory-test commands.

Return Value Type: Boolean.

Example:

```
; Check if function sieve is found in trace recording

Trace.Find ADDRess sieve
IF FOUND()==TRUE()
(
  PRINT "Function sieve found at trace record no. " TRACK.RECORD()
)
```

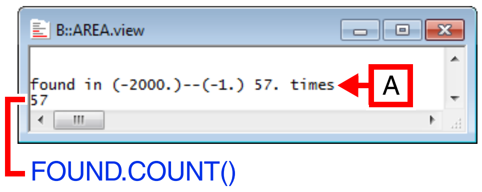
Syntax: **FOUND.COUNT()**

Returns the number of occurrences found by the commands **Trace.Find ... /ALL**, **Data.Find ... /ALL**, and **Data.FindCODE**.

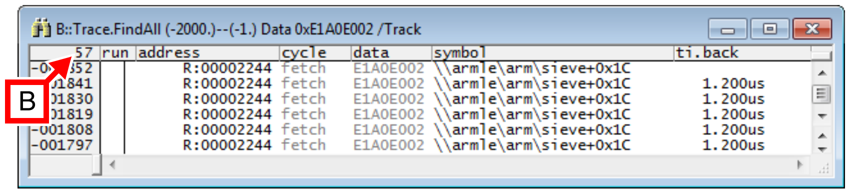
Return Value Type: Decimal value.

Example:

```
;list trace contents
Trace.List /Track

;find all occurrences of the specified data value in the record range
;(-2000.)--(-1.)
Trace.Find (-2000.)--(-1.) Data 0xE1A0E002 /All

;print the number of occurrences
PRINT FOUND.COUNT()

;display all occurrences
Trace.FindAll (-2000.)--(-1.) Data 0xE1A0E002 /Track
```



```
found in (-2000.)--(-1.) 57. times
57
```



```
     57 run address         cycle  data      symbol                   ti.back
-0[illegible]52       R:00002244  fetch  E1A0E002  \\armle\arm\sieve+0x1C
 01841       R:00002244  fetch  E1A0E002  \\armle\arm\sieve+0x1C        1.200us
 01830       R:00002244  fetch  E1A0E002  \\armle\arm\sieve+0x1C        1.200us
 01819       R:00002244  fetch  E1A0E002  \\armle\arm\sieve+0x1C        1.200us
-001808      R:00002244  fetch  E1A0E002  \\armle\arm\sieve+0x1C        1.200us
-001797      R:00002244  fetch  E1A0E002  \\armle\arm\sieve+0x1C        1.200us
```

A This line is automatically printed by the **Trace.Find ... /ALL** command, unless the line is suppressed with the **SILENT** pre-command.

B Equals the return value of **FOUND.COUNT()**.

GDB.PORT() Port number for communication via GDB interface

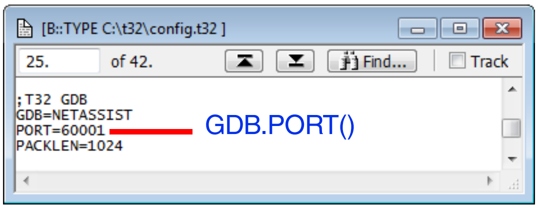
[build 59804 - DVD 02/2015]

Syntax: **GDB.PORT()**

Returns the port number used by the currently selected TRACE32 PowerView instance for communication via the GDB interface. Returns 0 if the port number is undefined.

Return Value Type: Decimal value.

Example: The port number is defined in the TRACE32 configuration file (default c:\t32\config.t32).



```
;T32 GDB
GDB=NETASSIST
PORT=60001
PACKLEN=1024
```

```
PRINT GDB.PORT()          ;returns 60001 because the InterCom setting in
                          ;the above configuration file reads:
                                  GDB=NETASSIST
                                  PORT=60001
```

HELP.MESSAGE()

Help search item

Syntax: **HELP.MESSAGE()**

Returns the search item string of the last error message for the online help (only applicable in PRACTICE scripts (*.cmm) in conjunction with the **ON ERROR** construct). Stepping through the script will cause an empty function return value.

Return Value Type: String.

Example:

```
    ON ERROR GOSUB ; install error handler in current PRACTICE stack frame
    (
       &help_message=HELP.MESSAGE()
       &error_occurred=1
       PRINT           "search item of error message in online help:"
       PRINT %CONTinue " &help_message"
       RETURN
    )

    LOCAL &help_message
    LOCAL &error_occurred

  l_system_up:
    &help_message=""
    &error_occurred=0
  ; a fail of SYStem.Up will call the ON ERROR routine above
    SYStem.Up
    IF &error_occurred==1
    (
  ;    check for target power fail
       IF "&help_message"=="#emu_errpwrf"
       (
  ;       PRINT     "Please power up the target board!"
          DIALOG.OK "Please power up the target board!"
          GOTO      l_system_up
       )
       ELSE IF "&help_message"!=""
       (
          PRINT     "other error occurred: &help_message"
          ON ERROR  ; remove current error handler from PRACTICE stack
          ENDDO
       )
    )

  ON ERROR   ;remove current error handler from PRACTICE stack
```

HOSTID() Host ID

Syntax: **HOSTID()**

Returns the host ID.

Return Value Type: Decimal value.

Example:

```
PRINT %Hex HOSTID()          ;print ethernet address of your host as hex
```

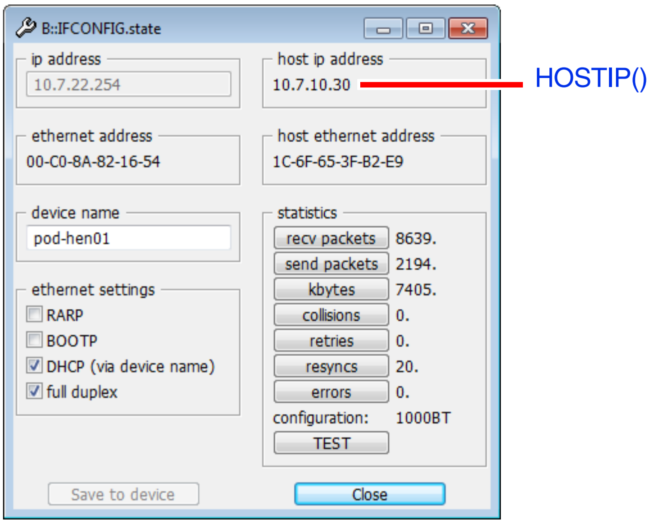
HOSTIP() Host IP address

Syntax: **HOSTIP()**

Returns the host IP address.

Return Value Type: Hex value. 32-bit.

Example: For the IPv4 address 10.2.10.26 (= 0x0A.0x02.0x0A.0x1A) you get 0x0A020A1A.



B::IFCONFIG.state

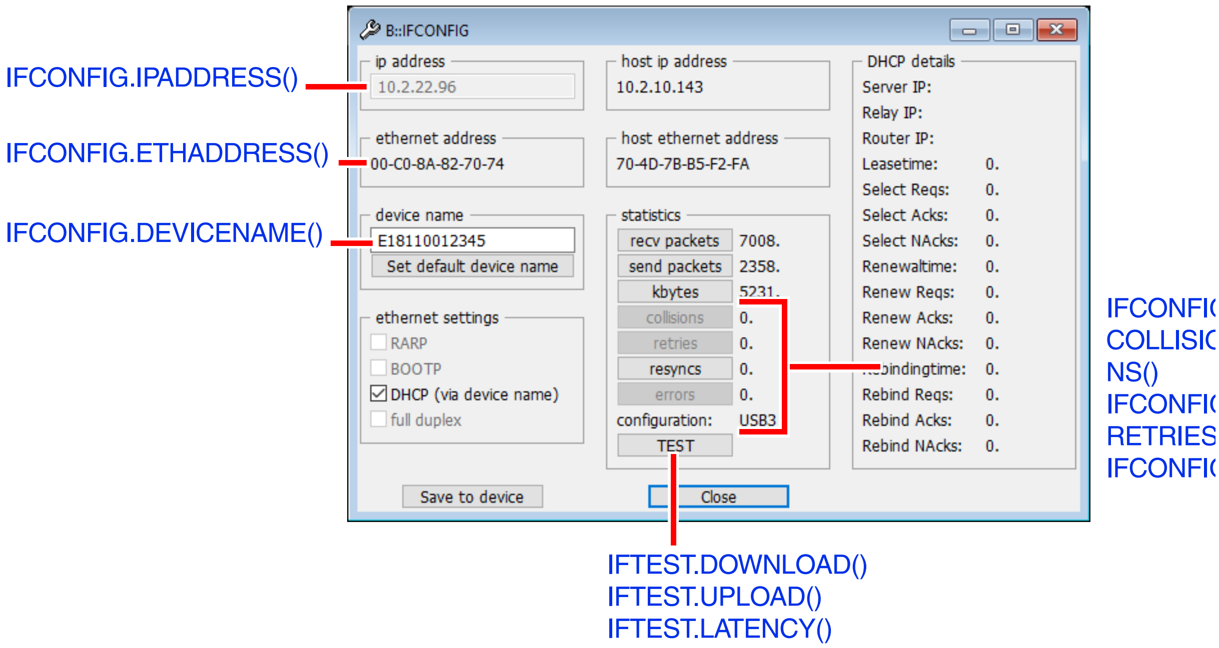
ip address: 10.7.22.254
host ip address: 10.7.10.30
ethernet address: 00-C0-8A-82-16-54
host ethernet address: 1C-6F-65-3F-B2-E9
device name: pod-hen01

statistics
recv packets 8639.
send packets 2194.
kbytes 7405.
collisions 0.
retries 0.
resyncs 20.
errors 0.
configuration: 1000BT

ethernet settings
☐ RARP
☐ BOOTP
☑ DHCP (via device name)
☑ full duplex

IFCONFIG and IFTEST Functions

This figure provides an overview of the return values of some of the functions. For explanations of the illustrated functions and the functions not shown here, see below.



ip address: 10.2.22.96
host ip address: 10.2.10.143
ethernet address: 00-C0-8A-82-70-74
host ethernet address: 70-4D-7B-B5-F2-FA
device name: E18110012345

ethernet settings
- [ ] RARP
- [ ] BOOTP
- [x] DHCP (via device name)
- [ ] full duplex

statistics

recv packets	7008.
send packets	2358.
kbytes	5231.
collisions	0.
retries	0.
resyncs	0.
errors	0.
configuration:	USB3

DHCP details

Server IP:	
Relay IP:	
Router IP:	
Leasetime:	0.
Select Reqs:	0.
Select Acks:	0.
Select NAcks:	0.
Renewaltime:	0.
Renew Reqs:	0.
Renew Acks:	0.
Renew NAcks:	0.
Rebindingtime:	0.
Rebind Reqs:	0.
Rebind Acks:	0.
Rebind NAcks:	0.

In This Section

See also

- ■ IFCONFIG
- ❏ IFCONFIG.CONFIGURATION()
- ❏ IFCONFIG.ERRORS()
- ❏ IFCONFIG.IPADDRESS()
- ❏ IFCONFIG.RETRIES()
- ❏ IFTEST.LATENCY()
- ❏ IFCONFIG.COLLISIONS()
- ❏ IFCONFIG.DEVICENAME()
- ❏ IFCONFIG.ETHernetADDRESS()
- ❏ IFCONFIG.RESYNCS()
- ❏ IFTEST.DOWNLOAD()
- ❏ IFTEST.UPLOAD()

IFCONFIG.COLLISIONS() Collisions since start-up

[build 6086 - DVD 11/2006] [Go to figure.]

Syntax: **IFCONFIG.COLLISIONS()**

Returns the number of collisions since start-up.

Return Value Type: Decimal value.

IFCONFIG.CONFIGURATION() Connection type

[Go to figure.]

Syntax:	**IFCONFIG.CONFIGURATION()**

Returns the connection type, e.g. `USB2` or `1000BT`.

Return Value Type: String. An empty string is returned if no TRACE32 devices are attached. To detect if TRACE32 is running on the instruction set simulator, use **SIMULATOR()**.

IFCONFIG.DEVICENAME() Name of TRACE32 device

[Go to figure.]

Syntax:	**IFCONFIG.DEVICENAME()**

The TRACE32 software can connect to TRACE32 devices through Ethernet or USB. This function returns the device name shown in the **IFCONFIG.state** window - regardless of whether the connection is an Ethernet or USB connection.

Entering a new device name and clicking **Save to device** in the **IFCONFIG.state** window changes the device name in the internal memory of the TRACE32 device.

Return Value Type: String.

See also: **NODENAME()**.

IFCONFIG.ERRORS() Errors since start-up

[build 6086 - DVD 11/2006] [Go to figure.]

Syntax:	**IFCONFIG.ERRORS()**

Returns the number of errors since start-up.

Return Value Type: Decimal value.

IFCONFIG.ETHernetADDRESS() MAC address of TRACE32 device

[build 62516 - DVD 09/2015] [Go to figure.]

Syntax:	**IFCONFIG.ETHernetADDRESS()**

Returns the MAC address of the TRACE32 device as a single 48-bit number if it is connected via Ethernet.

Return Value Type: Hex value. Returns zero if the TRACE32 device is not connected via Ethernet.

IFCONFIG.IPADDRESS() IP address of TRACE32 device

[Go to figure.]

Syntax: **IFCONFIG.IPADDRESS()**

Returns the IP address of the TRACE32 device if it is connected via Ethernet.

Return Value Type: String. Returns an empty string if the TRACE32 device is not connected via Ethernet.

See also: **IFCONFIG.DEVICENAME()**.

IFCONFIG.RESYNCS() Resyncs since start-up

[build 6086 - DVD 11/2006] [Go to figure.]

Syntax: **IFCONFIG.RESYNCS()**

Returns the number of resyncs since start-up.

Return Value Type: Decimal value.

IFCONFIG.RETRIES() Retries since start-up

[build 6086 - DVD 11/2006] [Go to figure.]

Syntax: **IFCONFIG.RETRIES()**

Returns the number of retries since startup. The same value and other data are displayed in the **IFCONFIG.state** (interface statistic) window.

Return Value Type: Decimal value.

Example:

```
IF IFCONFIG.RESYNCS()>1000.
   PRINT "poor network quality"
```

IFTEST.DOWNLOAD() Download in KByte/sec

[build 6086 - DVD 11/2006] [Go to figure.]

Syntax:	**IFTEST.DOWNLOAD()**

Returns the download result of the last executed **IFCONFIG.TEST** command in KByte/sec.

Return Value Type: Decimal value.

IFTEST.LATENCY() Latency in microseconds

[build 6086 - DVD 11/2006] [Go to figure.]

Syntax:	**IFTEST.LATENCY()**

Returns the latency result of the last executed **IFCONFIG.TEST** command in microseconds.

Return Value Type: Time value.

IFTEST.UPLOAD() Upload in KByte/sec

[build 6086 - DVD 11/2006] [Go to figure.]

Syntax:	**IFTEST.UPLOAD()**

Returns the upload result of the last executed **IFCONFIG.TEST** command in KByte/sec.

Return Value Type: Decimal value.

InterCom Functions

In This Section

See also

- ■ InterCom
- ❏ InterCom.GetPracticeState()
- ❏ InterCom.PING()
- ❏ InterCom.PODPORTNAME()
- ❏ InterCom.PORT()
- ❏ InterCom.GetGlobalMacro()
- ❏ InterCom.NAME()
- ❏ InterCom.PODPORT()
- ❏ InterCom.PODPORTNUMBER()

InterCom.GetGlobalMacro() Exchange strings between PowerView instances

[build 119109 - DVD 09/2020]

Syntax: **InterCom.GetGlobalMacro(***<name>* | *<host:port>*,"*<macro name>*"**)**

This function is intended to transfer strings from one PowerView instance to another via InterCom.

Return Value Type: String.

Example:

- You have two PowerView instances running.
- The first instance uses InterCom Port 10000.
- The second instance uses InterCom Port 10001.
- In the SECOND instance you run this PRACTICE script.

```
GLOBAL &myMacro
&myMacro="Hello World"
```

In the FIRST instance you now can get access to the content of the global macro named **&myMacro** of the SECOND instance with the function **InterCom.GetGlobalMacro()**.

For example, execute this command in the FIRST instance:

```
PRINT "&"+"myMacro ==
>"+InterCom.GetGlobalMacro(localhost:10001,"myMacro")+"<"
```

This command should print "**&myMacro == >Hello World<**" in the AREA message window.

NOTE:	When specifying the name of the global macro with "*<macro name>*", you MUST NOT prefix the macro name with a '&'.

InterCom.GetPracticeState() PRACTICE run-state on other instance

[build 120851 - DVD 09/2020]

Syntax:	**InterCom.GetPracticeState(***<intercom_name>* | [*<host>***:**]*<port_number>***)**

Return Value Type: String.

Return Value and Description:

none	No script running.
running	Script is running (STOP button can be pressed).
stopped	There is a PRACTICE script, but it is currently stopped.
dialog	There is a PRACTICE script, but it executed something like **DIALOG.YESNO** and is now waiting for user input.
err_*	PRACTICE script is currently stopped, because an error was encountered.
err_exec	Execution of a command resulted in an error.
err_syntax	There is was a syntax error (nothing was executed).
err_unknown	Script was stopped because of a currently unkown error (should not happen...).

InterCom.NAME() InterCom name of this TRACE32 instance

[build 94280 - DVD 09/2018]

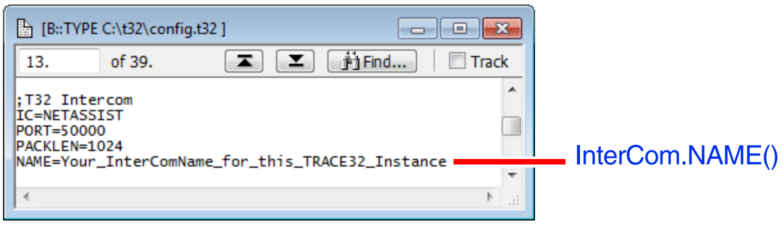
Syntax:	**InterCom.NAME()**

Returns the InterCom name of *the currently selected TRACE32 PowerView instance*. The InterCom name is used for communication with remote TRACE32 PowerView instances via the InterCom interface.

An InterCom name can be created in the config file (by default config.t32). See example 1. Alternatively, an InterCom name can be created with the commands **InterCom.NAME** or **InterCom.ENable**. See example 2.

Return Value Type: String.

Example 1:



```
;T32 Intercom
IC=NETASSIST
PORT=50000
PACKLEN=1024
NAME=Your_InterComName_for_this_TRACE32_Instance
```

```
PRINT InterCom.NAME()       ;returns:
                            ;Your_InterComName_for_this_TRACE32_Instance
                            ;because this InterCom name is used in the
                            ;above config file
```

Example 2:

```
InterCom.NAME firstInst     ;assigns the InterCom name 'firstInst' to the
                            ;current TRACE32 PowerView instance

PRINT InterCom.NAME()       ;returns: firstInst
```

See also: **InterCom.PODPORTNAME()**.

InterCom.PING() Check if ping is successful

Syntax: **InterCom.PING(***<intercom_name>* | [*<host>***:**]*<port_number>***)**

Returns TRUE if the ping was successful and FALSE if it failed. A runtime error occurs if the InterCom name or InterCom UDP port number could not be found and resolved, e.g. wrong port number used.

Parameter Type: String.

Return Value Type: Boolean.

Example:

```
; second debugger already started?
IF InterCom.PING(secondInst)==TRUE()
   PRINT "2. TRACE32 PowerView GUI alive"
```

InterCom.PODPORT()

Syntax: **InterCom.PODPORT(*<index>*)**

Returns the InterCom UDP port number of *any TRACE32 PowerView instance* that is connected to the same PowerDebug hardware module or the same MCI Server (`PBI=MCISERVER` in the config file).

The port number is used for communication with remote TRACE32 PowerView instances via the InterCom interface.

<instance>	**Parameter Type**: Decimal value. The valid range for *<index>* is: • 0 <= index < **InterCom.PODPORTNUMBER()**.

Return Value Type: Decimal value. In case an instance does not have InterCom configured, the function returns 0.

Example: See **InterCom.PODPORTNUMBER()**.

See also: **InterCom.PORT()**.

Syntax:	**InterCom.PODPORTNAME(*<index>*)**

Returns the InterCom name of *any TRACE32 PowerView instance* that is connected to the same PowerDebug hardware module or the same MCI Server (`PBI=MCISERVER` in the config.t32 file).

The InterCom name is used for communication with remote TRACE32 PowerView instances via the InterCom interface.

<index>	**Parameter Type**: Decimal value. The valid range for *<index>* is: • 0 <= index < **InterCom.PODPORTNUMBER()**.

Parameter Type: Decimal value.

Return Value Type: String. In case an instance does not have InterCom configured, the function returns an empty string.

Example: See **InterCom.PODPORTNUMBER()**.

See also: **InterCom.NAME()**.

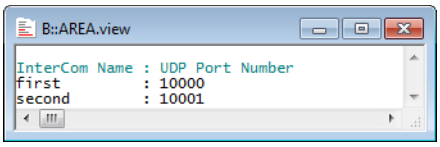
Syntax: **InterCom.PODPORTNUMBER()**

Returns the number of TRACE32 PowerView instances that are connected to the same PowerDebug hardware module or the same MCI Server (`PBI=MCISERVER` in the config.t32 file).

Return Value Type: Decimal value.

Example: This script iterates through the TRACE32 instances and prints their InterCom names and UDP port numbers to an **AREA.view** window. To format the output, the **PRINTF** command is used instead of the simple **PRINT** command.

```
&i=0.

AREA.view
PRINTF %COLOR.TEAL "%-14s: %s"  "InterCom Name"  "UDP Port Number"

RePeaT InterCom.PODPORTNUMBER()
(
   PRINTF "%-14s: %i"  InterCom.PODPORTNAME(&i)  InterCom.PODPORT(&i)
   &i=&i+1.
)
```



```
InterCom Name : UDP Port Number
first         : 10000
second        : 10001
```

InterCom.PORT()

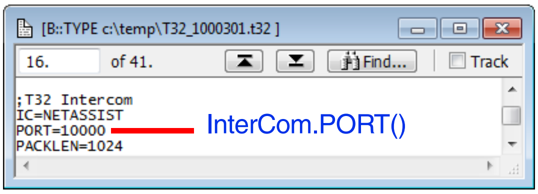
Syntax: **InterCom.PORT()**

Returns the InterCom UDP port number of *the currently selected TRACE32 PowerView instance*. The port number is used for communication with remote TRACE32 PowerView instances via the InterCom interface.

A port number can be created in the config file (by default config.t32). See example 1. Alternatively, a port number can be created with the commands **InterCom.PORT** or **InterCom.ENable**. See example 2.

Return Value Type: Decimal value. Returns 0 if the port number is undefined.

Example 1: The port number is defined in the TRACE32 configuration file (by default config.t32).



```
;T32 Intercom
IC=NETASSIST
PORT=10000
PACKLEN=1024
```

```
PRINT InterCom.PORT()    ;returns 10000 because the InterCom setting in
                         ;the above configuration file reads:
                         ;        IC=NETASSIST
                         ;        PORT=10000
```

Example 2: The **InterCom.PORT** command overrides the port number (`PORT=` in the config file) in favor of a new UDP port number.

```
InterCom.PORT 50000.     ;assigns the port number 50000 to the currently
                         ;selected TRACE32 PowerView instance

PRINT InterCom.PORT()    ;returns: 50000
```

See also: **InterCom.PODPORT()**.

LICENSE Functions

See also: **“VERSION Functions”** (general_func.pdf).

In This Section

See also

■ LICENSE	❏ LICENSE.DATE()	❏ LICENSE.FAMILY()	❏ LICENSE.FEATURES()
❏ LICENSE.getINDEX()	❏ LICENSE.GRANTED()	❏ LICENSE.HAVEFEATURE()	❏ LICENSE.MSERIAL()
❏ LICENSE.MULTICORE()	❏ LICENSE.RequiredForCPU()	❏ LICENSE.SERIAL()	

LICENSE.DATE() Expiration date of maintenance contract

[build 32168 - DVD 02/2012]

Syntax: **LICENSE.DATE(*<index>*)**

Returns the expiration date of the maintenance contract specified by index in the form YYYY/MM.

The persons responsible for license management can use this function in their scripts to check if the debugger licences are still valid and when licenses need to be renewed. Compare the value with **VERSION.DATE()** to check if the maintenance contract is still valid.

Parameter Type: Decimal value. The *<index>* (starting at 0) is related to the different licenses stored inside a debug cable.

Return Value Type: String.

See also: **LICENSE.getINDEX()**.

LICENSE.FAMILY() Name of the CPU family license

[build 32168 - DVD 02/2012]

Syntax: **LICENSE.FAMILY(*<index>*)**

Returns the name of the CPU family license via the serial number specified by index.
See also **LICENSE.getINDEX()**.

Parameter Type: Decimal value. The *<index>* (starting at 0) is related to the different licenses stored inside a debug cable.

Return Value Type: String.

LICENSE.FEATURES() List of features licensed

[build 32168 - DVD 02/2012]

Syntax: **LICENSE.FEATURES(*<index>*)**

Returns a comma-separated list of features licensed by the serial number specified by index (as shown in the **LICENSE.List** window).

The list of features refers to the features that are programmed into the debug cable, Nexus adapter, CombiProbe, and preprocessor. You can use the **STRing.SCAN()** function to analyze the comma-separated list of features.

The return values of **LICENSE.FEATURES()** and **STRing.SCAN()** can be used in scripts to check whether or not a debugger supports the desired CPU family, e.g. ARM 9. If not, the script could display a dialog box to notify the user, e.g. **DIALOG.OK**, and then exit.

Parameter Type: Decimal value. The *<index>* (starting at 0) is related to the different licenses stored inside a debug cable.

Return Value Type: String.

See also: **LICENSE.getINDEX()**.

LICENSE.getINDEX() Index of maintenance contract

[build 32168 - DVD 02/2012]

Syntax: **LICENSE.getINDEX()**

Returns the index of the currently used maintenance contract. The index can be used in these functions:

- **LICENSE.DATE()**
- **LICENSE.MSERIAL()**
- **LICENSE.SERIAL()**
- **LICENSE.FAMILY()**
- **LICENSE.FEATURES()**

Return Value Type: Decimal value.

LICENSE.GRANTED() License state

Syntax:	**LICENSE.GRANTED(***<product>***,***<version>***)**

Returns an integer value that reflects the current license state of a product-version combination.

Parameter and Description:

<product>	**Parameter Type**: String. License product name, e.g. as given in a lauterbach-*.lic file. For example: “t32.trace.x86”
<version>	**Parameter Type**: String. License version, e.g. as given in a lauterbach-*.lic file. For example: “2013.05” If the version string is empty, e.g. “”, then TRACE32 will try to auto-fill in the version string, based on the product type.

Return Value Type: Decimal value.

Return Value and Description:

0	License found.
1	License not found.
2	License temporarily not available.
3	License permanently not available.
16	*<product>* name was an empty string.

Example:

```
PRINT LICENSE.GRANTED("t32.trace.x86","2013.05")
```

See also: **LICENSE.REQuest** command.

LICENSE.HAVEFEATURE() Checks if license is stored in debugger hardware

[build 45728 - DVD 08/2013]

Syntax:	**LICENSE.HAVEFEATURE("***<name>***")**

Returns TRUE if a specified feature license is available in the used Lauterbach debugger hardware.

Parameter Type: String.

Return Value Type: Boolean.

Example:

```
PRINT LICENSE.HAVEFEATURE("arm9")
```

LICENSE.MSERIAL() Serial number of the maintenance contract

[build 32168 - DVD 02/2012]

Syntax: **LICENSE.MSERIAL(*<index>*)**

Returns the serial number of the maintenance contract specified by *<index>*. The function can be used to check if a user has a temporary or regular maintenance contract.

Parameter Type: Decimal value. The *<index>* (starting at 0) is related to the different licenses stored inside a debug cable.

Return Value Type: String.

See also: **LICENSE.getINDEX()**.

LICENSE.MULTICORE() Check if multicore debugging is licensed

[build 68535, DVD 2016/02]

Syntax: **LICENSE.MULTICORE()**

Returns TRUE if multicore debugging is licensed.

Return Value Type: Boolean.

LICENSE.RequiredForCPU() License required for selected CPU

[build 128901 - DVD 02/2021]

Syntax: **LICENSE.RequiredForCPU()**

Returns the licenses required for the CPU seleted in SYStem.CPU as a name.
The result is empty for SYStem.CPU NONE
The result is also empty if you are not using a PowerDebug or HostMCI or ESI

Return Value Type: String.

[build 32168 - DVD 02/2012]

Syntax:	**LICENSE.SERIAL(***<index>***)**
	VERSION.LICENSE() (deprecated)

Returns the serial number of the debug cable specified by *<index>*.

Parameter Type: Decimal value. The *<index>* (starting at 0) is related to the different licenses stored inside a debug cable.

Return Value Type: String.

See also: **LICENSE.getINDEX()**.

LOG.DO.FILE() Get log file used by LOG.DO

[build 99855 - DVD 09/2018]

Syntax:	**LOG.DO.FILE()**

Returns the name of the log file which is currently in use by the command **LOG.DO**. If no log is open, the function returns an empty string.

Return Value Type: String.

Mathematical Functions

In This Section

See also

❏ math.ABS()	❏ math.FABS()	❏ math.FCOS()	❏ math.FEXP()
❏ math.FEXP10()	❏ math.FINF()	❏ math.FLOG()	❏ math.FLOG10()
❏ math.FMAX()	❏ math.FMIN()	❏ math.FNAN()	❏ math.FPOW()
❏ math.FSIN()	❏ math.FSQRT()	❏ math.MAX()	❏ math.MIN()
❏ math.SIGN()	❏ math.SIGNUM()	❏ math.TimeMAX()	❏ math.TimeMIN()

math.ABS() Absolute value of decimal value

[build 64326 - DVD 2015/09]

Syntax: **math.ABS(***<integer>***)**

Calculates the absolute value of an integer. Negative values are inverted.

Parameter Type: Decimal value.

Return Value Type: Decimal value.

math.FABS() Absolute value of floating point number

[build 69272 - DVD 2016/02]

Syntax: **math.FABS(***<float>***)**

Calculates the absolute value of a floating point number. Negative values are inverted.

Parameter Type: Float.

Return Value Type: Float.

math.FCOS() Cosine of an angle given in radian

Syntax:	**math.FCOS(**<*float*>**)**
	FCOS() (deprecated)

Returns cosine of an angle given in radian.

Parameter Type: Float.

Return Value Type: Float.

math.FEXP() Exponentiation with base e (Euler's number)

Syntax:	**math.FEXP(**<*float*>**)**
	FEXP() (deprecated)

Returns the exponentiation of base e (Euler’s number) by given exponent.

Parameter Type: Float.

Return Value Type: Float.

math.FEXP10() Exponentiation with base 10

Syntax:	**math.FEXP10(**<*float*>**)**
	FEXP10() (deprecated)

Returns the exponentiation of base 10 by given exponent.

Parameter Type: Float.

Return Value Type: Float.

math.FINF() Positive infinity

Syntax:	**math.FINF()**
	FINF() (deprecated)

Returns inf() IEEE representation for positive infinity.

Return Value Type: Float.

Example:

```
PRINT 1000000./math.FINF()  ;result 0.0
```

math.FLOG() — Natural logarithm of given value

Syntax: **math.FLOG(***<float>***)**
FLOG() (deprecated)

Returns natural logarithm (to base of Euler's number) of given value.

Parameter Type: Float.

Return Value Type: Float.

math.FLOG10() — Logarithm to base 10 of given value

Syntax: **math.FLOG10(***<float>***)**
FLOG10() (deprecated)

Returns logarithm to base 10 of given value.

Parameter Type: Float.

Return Value Type: Float.

math.FMAX() — Return the larger one of two floating point values

[build 66893 - DVD 02/2016]

Syntax: **math.FMAX(***<float1>***,***<float2>***)**

Compares two floating point parameters and returns the larger one of the two values.

Parameter Type: Float.

Return Value Type: Float.

Examples:

```
PRINT math.FMAX(3.9,4.1)     ; result 4.1

PRINT math.FMAX(-3.9,-4.1)   ; result -3.9
```

math.FMIN() Return the smaller one of two floating point values

[build 66893 - DVD 02/2016]

Syntax: **math.FMIN(*<float1>*,*<float2>*)**

Compares two floating point parameters and returns the smaller one of the two values.

Parameter Type: Float.

Return Value Type: Float.

Examples:

```
PRINT math.FMIN(3.9,4.1)      ; result 3.9

PRINT math.FMIN(-3.9,-4.1)    ; result -4.1
```

math.FMOD() Floating-Point Modulus

[build 133635 - DVD 09/2021]

Syntax: **math.FMOD(*<x>*,*<y>*)**

Compute the floating-point modulus, i. e. the remainder of the division x / y. The returned value z has the same sign as x. It holds that z = x - k * y for some integer value k.

Parameter and Description:

<x>	**Parameter Type**: Float.
<y>	**Parameter Type**: Float.

Return Value Type: Float.

Examples:

```
ECHO math.FMOD(7.0,2.5) ; prints 2.0 because 2.0 = 7.0 - 2 * 2.5
ECHO math.FMOD(1.0,0.0) ; prints NAN
ECHO math.FMOD(1.0,0.1) ; prints 54.2101086242752217e-21 because 0.1
                        ; cannot be represented exactly as a Float
```

math.FNAN() Not a number value

Syntax:	**math.FNAN()**
	FNAN() (deprecated)

Returns nan() - **N**ot **A** **N**umber value.

Return Value Type: Float.

math.FPOW() Y-th power of base x

[build 64326 - DVD 2015/09]

Syntax:	**math.FPOW(***<float_x>***,***<float_y>***)**

Calculates the y-th power of base x.

Parameter Type: Float.

Return Value Type: Float.

math.FSIN() Sine of an angle given in radian

Syntax:	**math.FSIN(***<value>***)**
	FSIN() (deprecated)

Returns sine of an angle given in radian.

Parameter Type: Float.

Return Value Type: Float.

math.FSQRT() Square-root of given value

Syntax:	**math.FSQRT(***<value>***)**
	FSQRT() (deprecated)

Returns square-root of given value.

Parameter Type: Float.

Return Value Type: Float.

math.MAX() Return the larger one of two decimal values

[build 66893 - DVD 02/2016]

Syntax: **math.MAX(***<integer1>***,***<integer2>***)**

Compares two integer parameters and returns the larger one of both values.

Parameter Type: Decimal value.

Return Value Type: Decimal value.

Examples:

```
PRINT math.MAX(5.,2.)         ; result 5

PRINT math.MAX(-5.,-2.)       ; result -2
```

math.MIN() Return the smaller one of two decimal values

[build 66893 - DVD 02/2016]

Syntax: **math.MIN(***<integer1>***,***<integer2>***)**

Compares two integer parameters and returns the smaller one of both values.

Parameter Type: Decimal value.

Return Value Type: Decimal value.

Examples:

```
PRINT math.MIN(5.,2.)         ; result 2

PRINT math.MIN(-5.,-2.)       ; result -5
```

math.SIGN() Return -1 or +1 depending on argument

[build 64326 - DVD 2015/09]

Syntax: **math.SIGN(***<integer>***)**

Returns -1 for negative values and +1 for positive values.

Parameter Type: Decimal value.

Return Value Type: Decimal value.

Examples:

```
PRINT math.SIGN(-300.)        ; result -1
PRINT math.SIGN(500.)         ; result 1
PRINT math.SIGN(+700.)        ; result 1
```

math.SIGNUM()   Return -1 or 0 or +1 depending on argument

[build 64326 - DVD 2015/09]

Syntax: **math.SIGNUM(***<integer>***)**

Returns -1 for negative values, 0 for a zero value and +1 for positive values.

Parameter Type: Decimal value.

Return Value Type: Decimal value.

math.TimeMAX()   Return the larger one of two time values

[build 66893 - DVD 02/2016]

Syntax: **math.TimeMAX(***<time1>***,***<time2>***)**

Compares two time parameters and returns the larger one of the two values.

Parameter Type: Time value.

Return Value Type: Time value. It is measured in seconds.

Examples:

```
PRINT math.TimeMAX(120ms,1.5s)              ; result 1.500000000s

PRINT math.TimeMAX(-120ms,-1.5s)            ; result -0.120000000s
```

math.TimeMIN()   Return the smaller one of two time values

[build 66893 - DVD 02/2016]

Syntax: **math.TimeMIN(***<time1>***,***<time2>***)**

Compares both time parameters and returns the smaller one of both values.

Parameter Type: Time value.

Return Value Type: Time value. It is measured in seconds.

Examples:

```
PRINT math.TimeMIN(120ms,1.5s)                 ; result 0.120000000s

PRINT math.TimeMIN(-120ms,-1.5s)               ; result -1.500000000s
```

MENU.EXIST() Check if menu name exists

Syntax:	**MENU.EXIST(**<*name*>**)**

Returns TRUE if a menu with a specified name exists.

Parameter Type: String. Menu names are case-insensitive.

Return Value Type: Boolean.

NODENAME Function

NODENAME() Node name of connected TRACE32 device

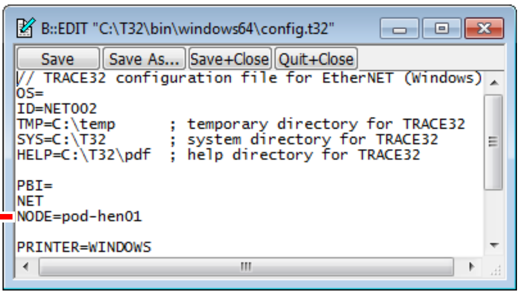
Syntax: **NODENAME()**

TRACE32 software can connect to TRACE32 devices through Ethernet or USB. The **NODENAME()** function returns the node name of the connected TRACE32 device.

Return Value Type: String.

- For **Ethernet** connections, the node name of the connected TRACE32 device is returned. The node name itself is a setting in the config.t32 file, as shown below.
- For **USB** connections, just an empty string is returned.

NODENAME()

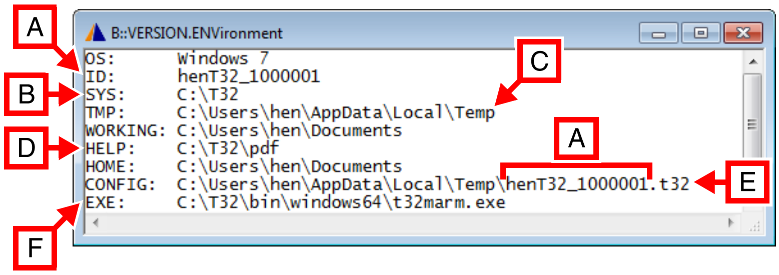


```
// TRACE32 configuration file for EtherNET (Windows)
OS=
ID=NET002
TMP=C:\temp      ; temporary directory for TRACE32
SYS=C:\T32       ; system directory for TRACE32
HELP=C:\T32\pdf  ; help directory for TRACE32

PBI=
NET
NODE=pod-hen01

PRINTER=WINDOWS
```

See also: **IFCONFIG.DEVICENAME()**

OS Functions

This figure provides an overview of the return values of some of the OS functions. For explanations of the illustrated functions and the functions not shown here, see below.



```
OS:      Windows 7
ID:      henT32_1000001
SYS:     C:\T32
TMP:     C:\Users\hen\AppData\Local\Temp
WORKING: C:\Users\hen\Documents
HELP:    C:\T32\pdf
HOME:    C:\Users\hen\Documents
CONFIG:  C:\Users\hen\AppData\Local\Temp\henT32_1000001.t32
EXE:     C:\T32\bin\windows64\t32marm.exe
```

A	OS.ID()	**B**	OS.PresentSystemDirectory()
C	OS.PresentTemporaryDirectory()	**D**	OS.PresentHELPDirectory()
E	OS.PresentConfigurationFile()	**F**	OS.PresentExecutableFile()

Alternatively use: VERSION.ENVironment()

The **VERSION.ENVironment** command opens the above window, displaying the TRACE32 environment settings.

In This Section

See also

- ❏ OS.ACCESS()
- ❏ OS.DIR()
- ❏ OS.DIR.ACCESS()
- ❏ OS.ENV()
- ❏ OS.FILE.ABSPATH()
- ❏ OS.FILE.ACCESS()
- ❏ OS.FILE.BASENAME()
- ❏ OS.FILE.DATE()
- ❏ OS.FILE.DATE2()
- ❏ OS.FILE.EXIST()
- ❏ OS.FILE.EXTENSION()
- ❏ OS.FILE.JOINPATH()
- ❏ OS.FILE.LINK()
- ❏ OS.FILE.NAME()
- ❏ OS.FILE.PATH()
- ❏ OS.FILE.readable()
- ❏ OS.FILE.REALPATH()
- ❏ OS.FILE.SIZE()
- ❏ OS.FILE.TIME()
- ❏ OS.FILE.UnixTime()
- ❏ OS.FIRSTFILE()
- ❏ OS.ID()
- ❏ OS.NAME()
- ❏ OS.NEXTFILE()
- ❏ OS.PORTAVAILABLE.TCP()
- ❏ OS.PORTAVAILABLE.UDP()
- ❏ OS.PresentConfigurationFile()
- ❏ OS.PresentDemoDirectory()
- ❏ OS.PresentExecutableDirectory()
- ❏ OS.PresentExecutableFile()
- ❏ OS.PresentHELPDirectory()
- ❏ OS.PresentHomeDirectory()
- ❏ OS.PresentLicenseFile()
- ❏ OS.PresentPracticeDirectory()
- ❏ OS.PresentPracticeFile()
- ❏ OS.PresentSystemDirectory()
- ❏ OS.PresentTemporaryDirectory()
- ❏ OS.PresentWorkingDirectory()
- ❏ OS.RETURN()
- ❏ OS.TIMER()
- ❏ OS.TMPFILE()
- ❏ OS.VERSION()
- ❏ OS.Window.LINE()

OS.DIR() Check if directory exists

Syntax: **OS.DIR(***<directory_name>***)**

Returns TRUE if the directory exists.

Parameter Type: String.

Return Value Type: Boolean.

OS.DIR.ACCESS() Access rights to directory

[build 30295 - DVD 06/2011]

Syntax: **OS.DIR.ACCESS(***<directory_name>***,"{***<access_right>***}")**
OS.ACCESS(*<directory_name>* |*<file>***,"{***<access_right>***}")** (deprecated)

Returns TRUE when the *<directory_name>* fulfills **all** of the requested **access rights**.

Parameter and Description:

<directory_name>	**Parameter Type**: String.
<access_right>	**Parameter Type**: String. • **R** : Permissions to list directory • **W**: Permissions to change entries of directory • **X** : Permissions to traverse/open directory • **D** : Permissions to delete any file or subdirectory in directory • **K** : Permissions to delete directory itself • **C** : Permissions to create files in directory • **A** or **S** : Permissions to create subfolders in directory

Return Value Type: Boolean.

NOTE: **OS.DIR.ACCESS()** will only check the permissions to read/write/delete/open a directory. The function does not guarantee that you will be really able to access the directory because it might be inaccessible for other reasons.
E.g. You can’t usually delete a directory if a file within this directory is has been opened exclusively at the same time (although you might have the proper access rights).

Example:

```
// returns TRUE if you are allowed to create or modify files
// inside directory "C:\Program Files"
PRINT OS.DIR.ACCESS("C:\Program Files","wc")
```

Syntax:	**OS.ENV(***<env_var>***)**

Returns the contents of an OS environment variable.

Parameter Type: String.

Return Value Type: String. If the environment variable is not defined, **OS.ENV()** returns an empty string.

Example: An extra button is added to the TRACE32 toolbar for a particular user on a particular computer if **OS.ENV()** returns the values Paul and PaulPC1 for the environment variables USER and COMPUTERNAME.

```
LOCAL &param1 &param2

&param1="USERNAME"
&param2="COMPUTERNAME"

IF OS.ENV(&param1)=="Paul"&&OS.ENV(&param2)=="PaulPC1"
(   ;Return the values of the env. variables in the TRACE32 message bar
    PRINT OS.ENV(&param1)+"  "+OS.ENV(&param2)

    ;Add button to toolbar: <tooltip text>  <tool image>      <command>
    MENU.AddTool "Special Test"     "ST,G"    "DO ~~~~/special-test.cmm"
)
```

OS.FILE.readable() Check if file can be opened for reading

Syntax: **OS.FILE.readable(<*file*>)**

Returns TRUE if the file can be opened for reading. This is useful on operation systems, where an existing file can maybe not opened because some other process is locking the file.

Parameter Type: String.

Return Value Type: Boolean.

Example: A file is opened for writing if it already exists, If not, the file is created. Then a timestamp is written to the file.

```
MKTEMP &file                    ; Creates an empty file with unique name
ECHO OS.FILE.readable("&file")  ; Returns here TRUE()
ECHO OS.FILE("&file")        ; Short for OS.FILE.readable(), Returns TRUE()
OPEN #1 "&file" /Write /Read    ; Opens file exclusively on MS Windows
ECHO OS.FILE.readable("&file")  ; Returns here FALSE()
ECHO OS.FILE("&file")           ; Returns here FALSE()
ECHO OS.FILE.EXIST("&file")     ; Returns TRUE()
ECHO FILE.EXIST("&file")        ; Returns TRUE()
CLOSE #1                        ; Closes the file for writing
```

OS.FILE.ABSPATH() Absolute path to file or directory

[build 70945 - DVD 09/2016]

Syntax: **OS.FILE.ABSPATH(<*file*>)**

Returns the absolute path to a file or directory. The absolute path does not contain any **.** or **..** or any leading tildes (~), nor any repeated path separators. In contrast to **OS.FILE.REALPATH()**, it does not check if the file or directory actually exists. The return value may contain symbolic links (or junction points).

Parameter Type: String.

Return Value Type: String.

[build 30295 - DVD 06/2011]

Syntax:	**OS.FILE.ACCESS(***<file>***,"{***<access_type>***}")**
	OS.ACCESS(*<file>* |*<directory>***,"{***<access_type>***}")** (deprecated)

Returns TRUE when the *<file>* fulfills **all** of the requested **access rights**.

Parameter and Description:

<file>	**Parameter Type**: String.
<access_type>	**Parameter Type**: String. • **R** : Read file • **W**: Write file • **X** : Execute file • **D** or **K**: Delete file • **C** : Change data of file • **A** or **S** : Append data to file

Return Value Type: Boolean.

NOTE:	**OS.FILE.ACCESS()** will only check the permissions to read/write/delete/execute a file. The function does not guarantee that you will be really able to access the file because it might be inaccessible for other reasons. E.g. If a file is exclusively opened by another application (file-lock) you can't write to the file even you might have the permission to write the file.

Example:

```
//returns TRUE if you are allowed to READ and WRITE to/from file test.cmm
PRINT OS.FILE.ACCESS("C:\t32\test.cmm","rw")
```

OS.FILE.BASENAME() Strip directory and suffix from filenames

Syntax: **OS.FILE.BASENAME(***<path>*[**,"**<suffix>**"**]**)**

Returns the pure name part of a file name, like the POSIX command "basename".
The second parameter specifies a string which is removed from the end of the result if it matches. This intended to remove a file extensions.

Parameter and Description:

<path>	**Parameter Type**: String.
<suffix>	**Parameter Type**: String. Remove trailing suffix from *<path>.* Special suffixes are `.*` - remove last file extension `.**` - remove all file extensions .

Return Value Type: String.

Example:

```
ECHO OS.FILE.BASENAME("/usr/bin/sort","")        ; -> "sort"
ECHO OS.FILE.BASENAME("include/stdio.h","")      ; -> "stdio.h"
ECHO OS.FILE.BASENAME("include/stdio.h",".h")    ; -> "stdio"
ECHO OS.FILE.BASENAME("include/stdio/.h",".h")   ; -> ".h"
ECHO OS.FILE.BASENAME(".h",".h")                 ; -> ".h"
ECHO OS.FILE.BASENAME("x.h",".h")                ; -> "x"
ECHO OS.FILE.BASENAME("any/str1","")             ; -> "str1"
ECHO OS.FILE.BASENAME("any/str1\","")            ; -> "str1"
ECHO OS.FILE.BASENAME("~~/","")                  ; -> "T32"
; use wildcards
ECHO OS.FILE.BASENAME("test.tar.gz",".*")        ; -> "test.tar"
ECHO OS.FILE.BASENAME("test.tar.gz",".**")       ; -> "test”
```

Function nesting, i.e. `OS.FILE.NAME(OS.PresentExecutableFile(),".exe")`, is not supported.

OS.FILE.DATE() Modification date and timestamp of file

Syntax: **OS.FILE.DATE(***<file>***)**

Returns the modification date and timestamp *<day>*.*<month>*.*<year> <hour>*:*<minutes>*:*<second>*

Parameter Type: String.

Return Value Type: String.

OS.FILE.DATE2() Modification date of file

Syntax: **OS.FILE.DATE2(*<file>*)**

Returns the modification date *<year>/<month>/<day>*

Parameter Type: String.

Return Value Type: String.

OS.FILE.EXIST() Check if file exists

[build 115311 - DVD 02/2020]

Syntax: **OS.FILE.EXIST(*<file>*)**

Returns TRUE if the file exists. Alias for **FILE.EXIST()**. Use the function **OS.FILE.readable()** instead, if you want to be sure that the file can be actually opened for reading.

Parameter Type: String.

Return Value Type: Boolean.

Example: A file is opened for writing if it already exists, If not, the file is created. Then a timestamp is written to the file.

```
;If the file exists in the temporary directory of TRACE32
IF OS.FILE(~~~/myfile.txt)==TRUE()
    OPEN #1 ~~~/myfile.txt /Append        ;Open the file for writing
ELSE
    OPEN #1 ~~~/myfile.txt /Create        ;Create the file and open it


WRITE #1 "Session start: "+CLOCK.TIME()
CLOSE #1                                  ;Close the file for writing
```

OS.FILE.EXTENSION() File name extension

Syntax: **OS.FILE.EXTENSION(*<file>*)**

Returns the extension part of the file name.

Parameter Type: String.

Return Value Type: String.

Example:

```
PRINT OS.FILE.EXTENSION("~~/t32.men")        ; result: .men
```

OS.FILE.JOINPATH() Join multiple paths

[build 124461 - DVD 02/2021]

Syntax: **OS.FILE.JOINPATH(**<path1>{,<pathN>}**)**

The function **OS.FILE.JOINPATH** joins folder paths using the os-dependent separator. Duplicate separators are removed from the resulting path. The resulting path ends with a separator in case of the directory/drive root, otherwise it is removed.

It's recommended to use slash ('/') as input separator for os-independent usage.

Return Value Type: String.

Example 1:

```
PRIVATE &sPwd
&sPwd=OS.PresentWorkingDirectory()
PRINT OS.FILE.JOINPATH("&sPwd","../","logs","trace32_current.log")

;Example:
    ; Windows: OS.PresentWorkingDirectory() = C:\T32
    ; Output:  "C:\T32\..\logs\trace32_current.log"

    ; Linux/MacOS: OS.PresentWorkingDirectory() = /home/user/t32
    ; Output: /home/user/t32/../logs/trace32_current.log"

PRINT OS.FILE.JOINPATH("dir1/","\dir2","dir3","/dir4\","file.txt")

; Output (Windows): dir1\dir2\dir3\dir4\file.txt
; Output (Linux): The output for Linux does not give a valid path
; since the backslash ('\') is not a valid character, refer
; to next  example

PRINT OS.FILE.JOINPATH("dir1/","/dir2","dir3","/dir4/","file.txt")
; Output (Windows): dir1\dir2\dir3\dir4\file.txt
; Output (Linux/MacOS): dir1/dir2/dir3/dir4/file.txt
```

Example 2:

```
; Recursive search of all *.elf files in the TRACE32 system directory
; OS.FIRSTFILE/NEXTFILE returns the relative path
; OS.FILE.JOINPATH is used to prefix the base path again

PRIVATE &sBase &sPath
&sBase=OS.PresentSystemDirectory() ; TRACE32 system directory
&sPath=OS.FILE.JOINPATH("&sBase","/**/","*.elf")
&sPath=OS.FIRSTFILE("&sPath")
WHILE "&sPath"!=""
(
  PRINT OS.FILE.JOINPATH("&sBase","&sPath")
  &sPath=OS.NEXTFILE()
)
```

OS.FILE.LINK() Real file name of file link

Syntax: **OS.FILE.LINK(<*file*>)**

Returns the real file name of a file link.

Parameter Type: String.

Return Value Type: String.

Example:

```
;return path and file name of this MS Windows shortcut
PRINT OS.FILE.LINK("~~/bin/t32marm.exe.lnk")

;result: C:\T32\bin\windows\t32marm.exe
```

```
ls -l /home/t32/a.h                    ; a.h -> ../sources/include/b.h

PRINT OS.FILE.LINK(/home/t32/a.h)      ; result "../sources/include/b.h"
```

OS.FILE.NAME()

Syntax: **OS.FILE.NAME(*<path>*)**

Returns the pure name part of a file name (including file extension if there is one).

Parameter Type: String.

Return Value Type: String.

Example: The function returns the file name of a TRACE32 executable from the string that consists of path and file name.

```
;Declare a PRACTICE macro (variable)
LOCAL &fname

;Returns path and file name of the active TRACE32 executable
&fname=OS.PresentExecutableFile()
PRINT "&fname"
                        ;Windows: e.g. C:\T32\bin\windows64\t32marm.exe
                        ;Linux:   e.g. /home/user/t32/t32marm

;Returns just the file name from the string assigned to &fname
PRINT OS.FILE.NAME(&fname)
                        ;Windows: t32marm.exe
                        ;Linux:   t32marm
```

Function nesting, i.e. `OS.FILE.NAME(OS.PresentExecutableFile())`, is not supported.

OS.FILE.PATH() Return path of file

Syntax: **OS.FILE.PATH(<*file*>)**

Returns the path name part of the file name or when the name does not contain a path the actual working directory. The resulting path does not include a trailing slash or backslash, unless the file is located in a root directory.

Parameter Type: String.

Return Value Type: String.

Examples:

```
PRINT OS.FILE.PATH("C:/temp/test.cmm")      ;returns "C:\temp"

CD C:/t32                                   ;change directory
PRINT OS.FILE.PATH("test.cmm")              ;returns "C:\t32"
```

Path contains slash or backslash if file is located in root directory:

```
PRINT OS.FILE.PATH("C:\t32\test.cmm")      ;returns "C:\temp"
PRINT OS.FILE.PATH("C:\test.cmm")          ;returns "C:\"
```

OS.FILE.REALPATH() Canonical absolute path to file or directory

[build 70945 - DVD 09/2016]

Syntax: **OS.FILE.REALPATH(<*file*>)**

Returns the canonical absolute path to a file or directory. The canonical absolute path does not contain any symbolic link (or junction point), nor any **.** or **..** or leading tildes (~), nor any repeated path separators.

In contrast to **OS.FILE.ABSPATH()** the given path of a directory or file must exist. Otherwise an empty string will be returned.

While **OS.FILE.REALPATH()** resolves symbolic links and junction points, it does not resolve any file shortcuts (*.lnk files) of a Windows operating system. To resolve file shortcuts on Windows use **OS.FILE.LINK()**.

Parameter Type: String.

Return Value Type: String.

OS.FILE.SIZE() File size in bytes

Syntax: **OS.FILE.SIZE(***<file>***)**

Returns the size of the file in bytes.

Parameter Type: String.

Return Value Type: Decimal value.

OS.FILE.TIME() Modification timestamp of file

Syntax: **OS.FILE.TIME(***<file>***)**

Returns the modification timestamp.

Parameter Type: String.

Return Value Type: String.

Syntax: **OS.FILE.UnixTime(<*file*>)**

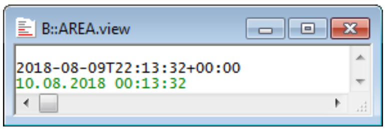
Returns the Unix timestamp of the last modification of the specified file. The Unix timestamp can be formatted to a human readable form with **DATE.MakeUnixTime()**.

Parameter Type: String.

Return Value Type: Decimal value.

Example: The backslash \ is used as a line continuation character. No white space permitted after the backslash.

```
;last modification date and time of file t32.men in ISO 8601 (UTC)
PRINT FORMAT.UnixTime("c",OS.FILE.UnixTime("~~/t32.men"),0)

;same result as PRINT OS.FILE.DATE("~~/t32.men")
PRINT FORMAT.UnixTime("d.m.Y H:i:s",\
                                 OS.FILE.UnixTime("~~/t32.men"),DATE.utcOffset())
```



```
2018-08-09T22:13:32+00:00
10.08.2018 00:13:32
```

Syntax: **OS.FIRSTFILE(*<pattern>*)**

OS.FIRSTFILE() in conjunction with **OS.NEXTFILE()** can be used to iterate over all files matching a specific *<pattern>*.

The *<pattern>* supports '*****' and '**?**' wildcards to match a specific name (e.g. 't32m*') or files of a specific type (e.g. '*.cmm'). Per default the search is perfomed in the present working directory.

Optionally the *<pattern>* can be prefixed with a directory (e.g. 'C:\t32', '/home/user') which can be used for a recursive search using the '******' syntax.

*****	matches any character 0 or more times.
?	matches any character 1 time.
******	matches directories recursively.

Example patterns:

- Match all files with extension '.cmm' in the present working directory

```
PRINT OS.FIRSTFILE("*.cmm")
; -> OS.NEXTFILE() ...
```

- Match all files '*<date>*runtest.log' with date *<year><month><day>* in c:\logs

```
PRINT OF.FIRSTFILE("c:\logs\????????runtest.log")
; -> OS.NEXTFILE()
```

- Match all files with extension '.cmm' recursively in the TRACE32 system directory (~~/)

```
PRINT OS.FIRSTFILE("~~/**/*.cmm")
; -> OS.NEXTFILE()
```

The function **OS.NEXTFILE()** returns the next file name matching the pattern. To return all file names matching the pattern, use a **WHILE** loop as shown in the example below.

Parameter Type: String.

Return Value Type: String.

Example: This script lists all TRACE32 executable file names that meet the following criteria:

- They reside in the `windows64` subfolder of the TRACE32 system directory (**~~/**)
- They start with the prefix `t32m`
- They have the suffix `.exe`

To try this script, copy it to a `test.cmm` file, and then run it in TRACE32 (See "**How to...**").

```
LOCAL &pattern &file

&pattern="~~\bin\windows64\t32m*.exe" ;define a pattern with folder path
                                      ;and file name

&file=OS.FIRSTFILE("&pattern")        ;get the first file name matching
                                      ;the pattern

OPEN #1 ~~~\list.dat /Create          ;create an output file in the
                                      ;temporary directory of TRACE32
WHILE "&file"!=""
(
   WRITE #1 "&file"                   ;write file name to output file

   &file=OS.NEXTFILE()                ;get next file name matching
)                                     ;the pattern

CLOSE #1                              ;close the output file and
TYPE ~~~\list.dat                     ;view the output file in TRACE32
```

OS.ID() User ID of TRACE32 instance

[Go to figure.]

Syntax: **OS.ID()**

Returns the user ID. Each user has a different user ID for each TRACE32 instance. The user ID is the same as in the **VERSION.ENVironment** window.

Return Value Type: String.

Example: The **OS.ID()** function is used to open the TRACE32 command history file referring to the active TRACE32 instance. The history file contains, among other things, the list of recently opened files and recently executed commands.

```
LOCAL &filename

;Concatenates the components of the history file name
&filename=OS.ID()+"store.cmm"

;The path prefix ~~~ expands to the temporary directory of TRACE32
EDIT ~~~/&filename ;Opens the history file
```

Commands that save a subset of the history file to a PRACTICE script file (*.cmm) are **HISTory.SAVE** and **STOre**. The **HISTory.type** command opens the history of the active TRACE32 instance right away. But note that double-clicking a line executes the selected command.

OS.NAME() Basic name of operating system

[build 64672 - DVD 09/2015]

Syntax: **OS.NAME()**

Returns the basic name of the operating system. A more detailed string can be obtained with the **VERSION.ENVironment(OS)** function.

Return Value Type: String.

Return Values: Windows, Linux, MacOSX, HP-UX, PowerPC

OS.NEXTFILE() Next file name matching a pattern

Syntax: **OS.NEXTFILE()**

The function **OS.FIRSTFILE(***<pattern>***)** returns the name of the first file within a specified folder. The function **OS.NEXTFILE()** returns the next file name matching the pattern.

Return Value Type: String.

Example: For an example of how to use both functions, see **OS.FIRSTFILE()**.

OS.PORTAVAILABLE.TCP() Check if TCP port is used

[build 76440 - DVD 09/2016]

Syntax: **OS.PORTAVAILABLE.TCP(***<port_number>***)**

Returns TRUE if the TCP port having the specified *<port_number>* is available at the host, i.e. the TCP port is *not* used.

Parameter Type: Decimal value. Range: **`1.`** to **`65535.`**

Return Value Type: Boolean.

OS.PORTAVAILABLE.UDP() Check if UDP port is used

[build 76440 - DVD 09/2016]

Syntax: **OS.PORTAVAILABLE.UDP(*<port_number>*)**

Returns TRUE if the UDP port having the specified *<port_number>* is available at the host, i.e. the UDP port number is *not* used.

Parameter Type: Decimal value. Range: **`1.`** to **`65535.`**

Return Value Type: Boolean.

Example: In this script, the function is used to find the next available UDP port number in the range 10000. to 10010.

```
GOSUB FindAvailableUDPPort 10000. 10010.
LOCAL &port
ENTRY &port
IF &port!=0.
  PRINT "Found available UDP Port at &port"
ELSE
  PRINT "no available UDP Port found"
ENDDO

FindAvailableUDPPort:
LOCAL &start &end &port
ENTRY &start &end
&port=&start

WHILE &port<=&end
(
  IF OS.PORTAVAILABLE.UDP(&port)==TRUE()
    RETURN &port
  &port=&port+1.
)
RETURN 0.
```

OS.PresentConfigurationFile() Name of used TRACE32 configuration file

[Go to figure.]

Syntax: **OS.PresentConfigurationFile()**
[build 89127 - DVD 02/2018]
OS.PCF()
[build 12210 - DVD 10/2008]

Returns the name of the currently used TRACE32 configuration file (default config.t32). You can access the configuration file by choosing **Help** menu > **About TRACE32**. Then click the **edit** button.

Return Value Type: String.

OS.PresentDemoDirectory() Demo directory for the current architecture

Syntax:	**OS.PresentDemoDirectory()** [build 89127 - DVD 02/2018] **OS.PDD()** [build 63341 - DVD 09/2015]

Returns path to the demo directory of the architecture, e.g.: C:\T32\demo\arm
In some cases the path can change depending on the currently selected CPU.

Return Value Type: String.

OS.PresentExecutableDirectory() Directory of current TRACE32 exe.

[Go to figure.]

Syntax:	**OS.PresentExecutableDirectory()** [build 89127 - DVD 02/2018] **OS.PED()** [build 32382 - DVD 06/2011]

Returns the directory name of the currently started TRACE32 executable, e.g.: C:\T32\bin\windows64

Return Value Type: String.

OS.PresentExecutableFile() Path and file name of current TRACE32 exe.

[Go to figure.]

Syntax:	**OS.PresentExecutableFile()** [build 89127 - DVD 02/2018] **OS.PEF()** [build 17601 - DVD 12/2009]

Returns the path and the file name of the currently started TRACE32 executable, e.g.:
C:\T32\bin\windows64\t32marm.exe

Return Value Type: String.

Remarks:

- The **OS.FILE.NAME()** function can be used to return just the file name.
- The **SOFTWARE.64BIT()** function can be used to find out if the TRACE32 software is a 64-bit executable.

OS.PresentHomeDirectory() Path of the home directory

[Go to figure.]

Syntax:	**OS.PresentHomeDirectory()**
	[build 89127 - DVD 02/2018]
	OS.PHD()
	[build 2280 - CD 07/2005]

Returns the path of the home directory.

Return Value Type: String.

OS.PresentHELPDirectory() Path of the TRACE32 online help directory

[Go to figure.]

Syntax:	**OS.PresentHELPDirectory()**
	[build 89127 - DVD 02/2018]
	OS.PHELPD()
	[build 12210 - DVD 10/2008]

Returns the path of the TRACE32 online help directory.

Return Value Type: String.

OS.PresentLicenseFile() Current TRACE32 license file

Syntax:	**OS.PresentLicenseFile()**
	[build 89127 - DVD 02/2018]
	OS.PLF()
	[build 12210 - DVD 10/2008]

Returns the name of the currently used TRACE32 license file (default: license.t32).

Return Value Type: String.

OS.PresentPracticeDirectory() Directory of currently executed script

Syntax:	**OS.PresentPracticeDirectory()**
	[build 89127 - DVD 02/2018]
	OS.PPD()
	[build 13751]

Returns the name of the directory of the currently executed PRACTICE script at the top of the PRACTICE stack. If no PRACTICE script is loaded, then this function returns an empty string. Use **PMACRO.list** to view the current PRACTICE stack.

Return Value Type: String.

OS.PresentPracticeFile() Path and file name of currently executed script

Syntax:	**OS.PresentPracticeFile()**
	[build 89127 - DVD 02/2018]
	OS.PPF()
	[build 12007 - DVD 10/2008]

Returns the path and file name of the currently executed PRACTICE script file at the top of the PRACTICE stack. If no PRACTICE script is loaded, then this function returns an empty string. Use **PMACRO.list** to view the current PRACTICE stack.

Return Value Type: String.

OS.PresentSystemDirectory() TRACE32 system directory

[Go to figure.]

Syntax:	**OS.PresentSystemDirectory()**
	[build 89127 - DVD 02/2018]
	OS.PSD()

Returns the name of the TRACE32 system directory.

Return Value Type: String.

OS.PresentTemporaryDirectory() TRACE32 temporary directory

[Go to figure.]

Syntax: **OS.PresentTemporaryDirectory()**
[build 89127 - DVD 02/2018]
OS.PTD()

Returns the name of the TRACE32 temporary directory.

Return Value Type: String.

OS.PresentWorkingDirectory() Current working directory

[Go to figure.]

Syntax: **OS.PresentWorkingDirectory()**
[build 89127 - DVD 02/2018]
OS.PWD()

Returns the name of the current working directory of TRACE32.

Return Value Type: String.

OS.RETURN() Return code of the last executed operating system command

Syntax: **OS.RETURN()**

Returns the return code of the last executed operating system commands from **OS.screen**, **OS.Window**, **OS.Hidden**, and **OS.Area**.

Return Value Type: Hex value.

Example:

```
OS.Area  myscript.bat                ;  PRACTICE script under
IF OS.RETURN()!=0                    ;  TRACE32 PowerView GUI
   GOTO  l_scripterror1

;------------------------------------------------------------------------
                                     // content of myscript.bat
perl myperlscript.pl                 // under Windows
exit %ERRORLEVEL%                    // forward the return value of
                                     // Perl call to TRACE32 PowerView GUI
```

OS.TIMER() OS timer in milliseconds

[build 06753 - DVD 01/2007]

Syntax: **OS.TIMER()**

Returns the OS timer in milliseconds. The resolution of this timer depends on the host operating system. For higher precision measurements, use **DATE.UnixTimeUS()**.

Return Value Type: Decimal value.

OS.TMPFILE() Name for a temporary file

[Examples]

Syntax: **OS.TMPFILE()**

Suggests the name for a temporary file. The function generates a *unique file name* each time the function is called.

To create the physical file with the suggested file name, you can use, for example, the **Data.SAVE.Binary** or the **OPEN** command. The newly-created file is stored in the temporary directory of TRACE32, see **OS.PresentTemporaryDirectory()**.

Return Value Type: String.

Examples

In the examples below, data is exchanged between the TRACE32 virtual memory (VM:) and a temporary file.

Example 1: The function **OS.TMPFILE()** and the command **Data.SAVE.Binary** are used to create a temporary backup file for virtual memory contents. The virtual memory contents are restored later on by loading them from the temporary backup file.

```
LOCAL &tfile1

; Get the file name for a temporary file
&tfile1=OS.TMPFILE()

Data.Set VM:0 %Byte 0 1 2 3 4 5 6 7 8    ; Write data to memory

Data.SAVE.Binary "&tfile1" VM:0--8       ; Back up memory contents

Data.Set VM:0 %Byte 0 0 0 0 0 0 0 0 0    ; Overwrite memory contents

Data.LOAD.Binary "&tfile1" VM:0--8       ; Restore memory contents

RM "&tfile1"                             ; Delete the temporary file
```

Example 2: The function **OS.TMPFILE()** and the command **OPEN** are used to create a temporary file. The data written to the temporary file is then loaded to the TRACE32 virtual memory (VM:).

```
LOCAL &tfile2

; Get the file name for a temporary file
&tfile2=OS.TMPFILE()

OPEN  #1 "&tfile2" /Create                ; Create and open the file
WRITE #1 %String "Hello TRACE32"          ; Write a string to the file
CLOSE #1

; Optional - lets you see the result on screen
Data.dump VM:0 /DIALOG

Data.LOAD.auto "&tfile2" VM:0             ; Load string to virtual memory

RM "&tfile2"                              ; Delete the temporary file
```

Syntax:	**OS.VERSION(*<version_data_type>*)**

Returns the type of the host operating system. The corresponding string is shown in the **VERSION.ENVironment** window.

Parameter and Description:

<version_data_type>	**Parameter Type**: Hex value. **0** returns the platform ID of the host operating system. **1** returns the major version. **2** returns the minor version. **3** returns the service pack number. **6** returns the product type (1:Workstation, 2:Domain-Ctrl.,3:Server) **8** returns the build number

Return Value Type: Hex value.

For detailed information about the returned hex values, see table Operating System.

NOTE:	If you only need to determine the OS name, see **OS.NAME()**.

Operating System	**OS.VERSION(0)**	**OS.VERSION(1)**	**OS.VERSION(2)**	**OS.VERSION(6)**
LINUX x86/x64 (x86 executable)	0x10	0	0	0
LINUX x64 (x64 executable)	0x11	0	0	0
LINUX on other architectures	>0x11 <0x20	0	0	0
HP-UX	0x30	0	0	0
MAC OS X (x86 executable)	0x40	0	0	0
MAC OS X (x64 executable)	0x41	0	0	0
other / unknown (should not happen)	0x00	0	0	0
Microsoft Windows	*PlatformId*	*MajorVersion*	*MinorVersion*	*ProductType*

Operating System	OS.VERSION(0)	OS.VERSION(1)	OS.VERSION(2)	OS.VERSION(6)
Windows 95 / 98	0x01	4	0 / 10	0
Windows ME	0x01	4	90	0
Windows NT 3.51	0x02	3	51	1
Windows NT 4.0	0x02	4	0	1
Windows 2000	0x02	5	0	1
Windows XP	0x02	5	1	1
Windows Server 2003	0x02	5	2	3
Windows Vista	0x02	6	0	1
Windows Server 2008 R1	0x02	6	0	3
Windows Server 2008 R2	0x02	6	1	3
Windows 7	0x02	6	1	1
Windows 8	0x02	6	2	1
Windows 8.1	0x02	6	3	1
Windows Server 2012 R1	0x02	6	2	3
Windows Server 2012 R2	0x02	6	3	3
Windows 10	0x02	10	1	1

Examples:

Example 1: PRACTICE script to detect the host operating system

```
; The logical OR operator in PRACTICE scripts is ||
IF (OS.VERSION(0)>=0x50)||(OS.VERSION(0)==0x00)
    PRINT "Unkown OS"
ELSE IF OS.VERSION(0)>=0x40
    PRINT "Mac OS X"
ELSE IF OS.VERSION(0)>=0x30
    PRINT "HP-UX"
ELSE IF OS.VERSION(0)>=0x10
    PRINT "Linux"
ELSE
    PRINT "MS Windows"
ENDDO
```

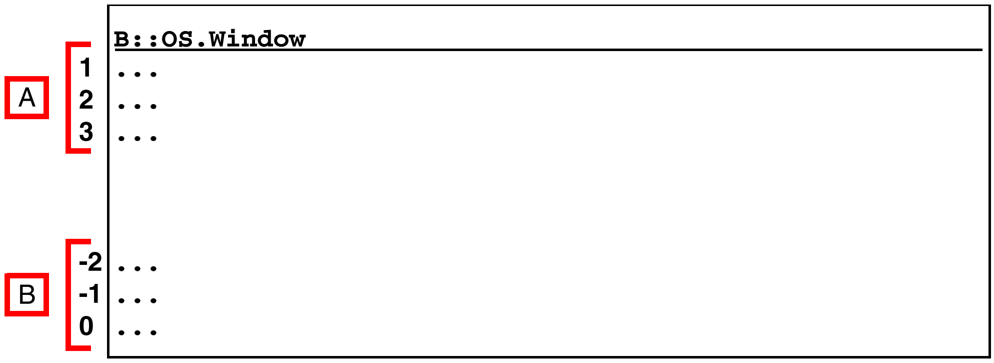
Example 2: The different *<version_data_types>* 0 to 3

```
IF (OS.VERSION(0)==0x1X)       ; For return values, see table
(                              ; Operating System above, column 2.
  PRINT "TRACE32 started on"
  PRINT "LINUX computer"
)



PRINT OS.VERSION(1) ; For return values, see table above, column 3.

PRINT OS.VERSION(2) ; For return values, see table above, column 4.

PRINT OS.VERSION(3) ; For return values, see table above, column 5.
```

Syntax: **OS.Window.LINE(WinTOP** | *<window_name>*,*<line>***)**

Returns one line from a window opened with command **OS.Window**.

This is useful to get the result of shell command executed in the host operating system, for further usage in a PRACTCIE script.



A Positive numbers for *<line>* starting with 1 identify lines from the top of the OS.Window window.

B 0 and negative numbers for *<line>* identify lines from the bottom of the OS.Window window.

Parameter and Description:

<window_name>	**Parameter Type**: String.
<line>	**Parameter Type**: Decimal value.

Return Value Type: String.

Example:

```
WinPOS ,,,,,,WINVER
OS.Window ver
ECHO OS.Window.LINE(WINVER,2)
```

PATH Functions

In This Section

See also

■ PATH	❏ PATH.NUMBER()	❏ PATH.PATH()

PATH.NUMBER()   Number of path entries

[build 99683 - DVD 09/2018]

Syntax:	**PATH.NUMBER()**

Returns the number of defined directories in the search path.
The actual defined directories from the search path are shown with the command **PATH.List**.

Return Value Type: Decimal value.

PATH.PATH()   Search path entry

[build 99683 - DVD 09/2018]

Syntax:	**PATH.PATH(***<index>***)**

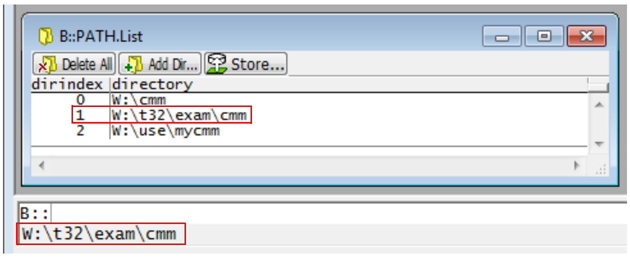
Returns a certain defined directory from the search path (see command **PATH.List**).

Parameter Type: Decimal value. The *<index>* (starting at 0) is related to the different directory names from the search path variable.

Return Value Type: String.

Example:

```
PATH.Set W:\cmm                           ; define three search directories
PATH.Set W:\t32\exam\cmm W:\use\mycmm     ; search directory order below

PRINT PATH.PATH(1.)                       ; output: W:\t32\exam\cmm
```



```
dirindex directory
       0 W:\cmm
       1 W:\t32\exam\cmm
       2 W:\use\mycmm
```

```
B::
W:\t32\exam\cmm
```

ProcessID() Process identifier of a TRACE32 PowerView instance

[build 16069 - DVD 12/2009]

Syntax:	**ProcessID()**

Returns the PID (process identifier) of the TRACE32 PowerView instance from the Windows Task Manager.

Return Value Type: Decimal value.

To display the PID column in Windows Task Manager:

1. Start the Windows Task Manager.
2. Click the **Processes** tab.
3. Choose **View** menu > **Select Columns**.
4. Select the **PID** check box.

PRACTICE Functions

In This Section

See also

- ❏ PRACTICE.ARG()
- ❏ PRACTICE.ARG.SIZE()
- ❏ PRACTICE.CALLER.FILE()
- ❏ PRACTICE.CALLER.LINE()
- ❏ PRACTICE.CoMmanD.AVAILable()
- ❏ PRACTICE.FUNCtion.AVAILable()

PRACTICE.ARG() Return value of GOSUB, DO, RETURN, and ENDDO

[build 65450 - DVD 02/2016]

Syntax: **PRACTICE.ARG(*<argument_index>*)**

Returns the value of the specified *<argument_index>*: a) *passed by* the commands **GOSUB** or **DO** or b) *returned by* the commands **RETURN** or **ENDDO**.

Parameter Type: Decimal or hex or binary value. The index number of the first argument is 0.

Return Value Type: String.

Example: A file name string is passed to a subroutine. The subroutine fetches the string using **PRACTICE.ARG()**. The illegal characters are removed and the cleaned string is returned to the caller. The caller fetches the cleaned string using **PRACTICE.ARG()**.

```
GOSUB fix_filename "/test/in*t.system"
PRINT "cleaned file name: " PRACTICE.ARG(0.)
ENDDO

fix_filename:

   PRIVATE &name
   &name=PRACTICE.ARG(0.)     ;get the file name string passed by GOSUB

                              ;create an array with 'illegal' characters
   Var.NEWLOCAL char[16] \remove_these = "[]|=/*./:;,\"\\"
   Var.NEWLOCAL int \i = 0;

   Var.WHILE \remove_these[\i]!=0
   (  ;replace all 'illegal' characters with an underscore '_'
      PRIVATE &search_str
      &search_str=""+CONVert.CHAR(Var.VALUE(\remove_these[\i++]))
      &name=STRing.Replace("&name", "&search_str", "_", 0.)
   )
   RETURN "&name"             ;return the cleaned file name to the caller
```

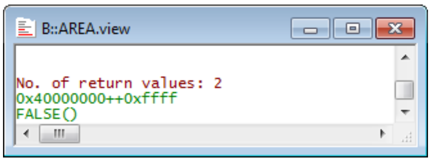
Syntax: **PRACTICE.ARG.SIZE()**

Returns the number of arguments: a) *passed by* the commands **GOSUB** or **DO** or b) *returned by* the commands **RETURN** or **ENDDO**.

Return Value Type: Decimal value.

Example: A subroutine returns a number of values to the caller. **PRACTICE.ARG.SIZE()** counts the number of values, and each value is then printed to an **AREA** window using a **WHILE** loop and **PRACTICE.ARG()**.

```
PMACRO.EXPLICIT                  ;enforce explicit macro declaration
PRIVATE &range &boolean &i
&i=0.

GOSUB AnySubroutine              ;call the subroutine

AREA.view                        ;open an AREA.view window
PRINT %COLOR.MAROON "No. of return values: " PRACTICE.ARG.SIZE()

WHILE &i<PRACTICE.ARG.SIZE()     ;print each return value to the
(                                ;AREA.view window
   PRINT %COLOR.GREEN PRACTICE.ARG(&i)
   &i=&i+1.
)

ENDDO


AnySubroutine:

   PRIVATE &my_rng &my_bool
                                 ;assign two values to PRACTICE macros:
   &my_rng="0x40000000++0xffff"  ;- any range
   &my_bool=FOUND()              ;- any boolean expression

RETURN "&my_rng" "&my_bool"      ;return the values to the caller
```



```
No. of return values: 2
0x40000000++0xffff
FALSE()
```

PRACTICE.CALLER.FILE() File name of the script/subscript caller

[build 65582 - DVD 09/2015]

Syntax: **PRACTICE.CALLER.FILE(*<index>*)**

Returns the file name of the script/subroutine caller.

Parameter and Description:

<index>	**Parameter Type**: Decimal value. Is used to walk through the call hierarchy.

Return Value Type: String. The function returns an empty string if no further caller exists.

Example: See **PRACTICE.CALLER.LINE()**.

PRACTICE.CALLER.LINE() Line number in caller script

[build 65582 - DVD 09/2015]

Syntax: **PRACTICE.CALLER.LINE(*<index>*)**

Returns the line number of a script/subroutine call in the caller script.

Parameter and Description:

<index>	**Parameter Type**: Decimal value. Is used to walk through the call hierarchy.

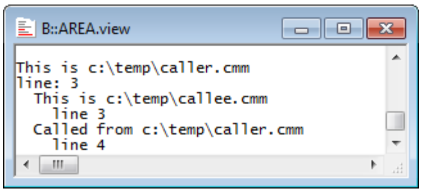
Return Value Type: Decimal value. The function returns 0. if no further caller exists.

Example:

```
// caller.cmm
PRINT "This is "+PRACTICE.CALLER.FILE(0.)
PRINT "line: "+FORMAT.Decimal(0.,PRACTICE.CALLER.LINE(0.))
DO callee.cmm // this call is in line 4
ENDDO
```

```
// callee.cmm
PRINT "  This is "+PRACTICE.CALLER.FILE(0.)
PRINT "    line "+FORMAT.Decimal(0.,PRACTICE.CALLER.LINE(0.))
PRINT "  Called from "+PRACTICE.CALLER.FILE(1.)
PRINT "    line "+FORMAT.Decimal(0.,PRACTICE.CALLER.LINE(1.))
ENDDO
```

Output:



```
This is c:\temp\caller.cmm
line: 3
  This is c:\temp\callee.cmm
    line 3
  Called from c:\temp\caller.cmm
    line 4
```

PRACTICE.CoMmanD.AVAILable()   Check if command is available

[build 64113 - DVD 09/2015]

Syntax: **PRACTICE.CoMmanD.AVAILable(***<command>***)**

Returns TRUE if a specified TRACE32/PRACTICE command is available. Returns FALSE if the passed TRACE32/PRACTICE command does not exist or is locked.

Parameter Type: String.

Return Value Type: Boolean.

Examples:

```
;returns TRUE()
PRINT PRACTICE.CoMmanD.AVAILable(sys.mode.down)

;returns FALSE(), since there is no such command "sys.mode.lunchbreak"
PRINT PRACTICE.CoMmanD.AVAILable(sys.mode.lunchbreak)
```

PRACTICE.CoMmanD.AVAILable() does not check parameters or arguments of the passed command:

```
;returns TRUE() - reason: without the period between mode and lunchbreak,
;lunchbreak is interpreted as a command argument
PRINT PRACTICE.CoMmanD.AVAILable(sys.mode lunchbreak)
```

Syntax: **PRACTICE.FUNCtion.AVAILable(***<function>***)**

Returns TRUE if a specified PRACTICE function is available. Returns FALSE if the passed function does not exist or is locked.

Parameter Type: String.

Return Value Type: Boolean.

Examples:

```
;returns TRUE()
PRINT PRACTICE.FUNCtion.AVAILable(Data.Long)

;returns FALSE()
PRINT PRACTICE.FUNCtion.AVAILable(imaginary.function)
```

PRACTICE.FUNCtion.AVAILable() does not check parameters or arguments of the passed function.

```
;returns TRUE(), even the argument is rubbish
PRINT PRACTICE.FUNCtion.AVAILable(Data.Long("rubbish"))
```

PRINTER.FILENAME() Path and file name of next print operation

[build 72057 - DVD 08/2016]

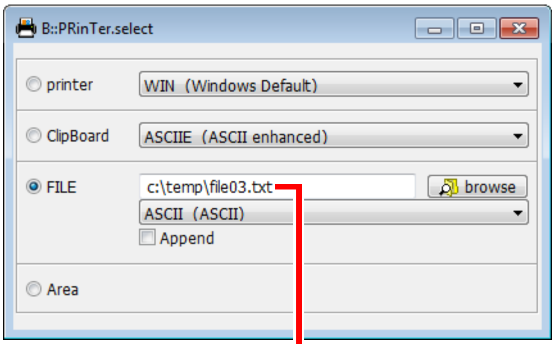
Syntax: **PRINTER.FILENAME()**

Returns the path and the initial file name set with the **PRinTer.FILE** command, e.g. c:\temp\file**01**.txt. In addition, the function returns the incremented file name for each subsequent print operation, e.g. c:\temp\file**02**.txt, c:\temp\file**03**.txt, etc.

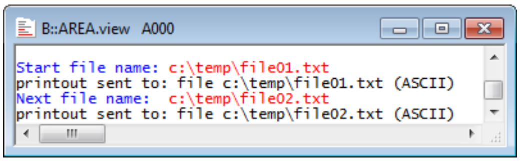
Return Value Type: String.

Example: Two **List.Mix** windows are printed to file, and each file name is returned with the **PRINTER.FILENAME()** function.

```
PRinTer.FILE ~~~\file01.txt          ; start with this file name

PRINT %COLOR.BLUE "Start file name: " %COLOR.RED PRINTER.FILENAME()

WinPrint.List.Mix func7--func17      ; print window to file01.txt

PRINT %COLOR.BLUE "Next file name:  " %COLOR.RED PRINTER.FILENAME()

WinPrint.List.Mix func18--func25     ; increment file name and print
                                     ; window to file02.txt
```


B::PRinTer.select
printer: WIN (Windows Default)
ClipBoard: ASCIIE (ASCII enhanced)
FILE: c:\temp\file03.txt
ASCII (ASCII)
Append
Area

PRINTER.FILENAME()


B::AREA.view A000

```
Start file name: c:\temp\file01.txt
printout sent to: file c:\temp\file01.txt (ASCII)
Next file name:  c:\temp\file02.txt
printout sent to: file c:\temp\file02.txt (ASCII)
```

RADIX() Current radix setting

Syntax:	**RADIX()**

Returns the current radix setting. See command **SETUP.RADIX**.

Return Value Type: Hex value.

Return Value and Description:

0x0a	Decimal mode.
0x10	Hex mode.

RANDOM() Pseudo random number

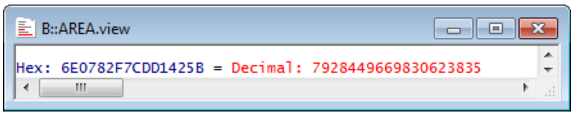
Syntax: **RANDOM()**

Returns a pseudo random number (64-bit). See also **SETUP.RANDOM** command.

Return Value Type: Hex value.

Example:

```
LOCAL &randomHex

&randomHex=RANDOM()

PRINT %COLOR.NAVY "Hex: " &randomHex
PRINT %CONTinue " = "
PRINT %CONTinue %COLOR.RED "Decimal: " CONVERT.HEXTOINT(&randomHex)
```



```
Hex: 6E0782F7CDD1425B = Decimal: 7928449669830623835
```

RANDOM.RANGE() Pseudo random number from specified range

Syntax: **RANDOM.RANGE(***<min>***,** *<max>***)**

Returns a pseudo random integer number in the range *<min>* … *<max>*. See also **SETUP.RANDOM** command.

Parameter and Description:

<min>	**Parameter Type**: Decimal value. Is a signed 64-bit integer.
<max>	**Parameter Type**: Decimal value. Is a signed 64-bit integer.

Return Value Type: Decimal value.

RANDOM.RANGE.HEX() Pseudo hex random number from specified range

Syntax:	**RANDOM.RANGE.HEX(***<min>***,** *<max>***)**

Returns a pseudo random hexadecimal number in the range *<min>* … *<max>*. This function is useful to fill registers with random values or generate random addresses. See also **SETUP.RANDOM** command.

Parameter and Description:

<min>	**Parameter Type**: Decimal value. Is an unsigned 64-bit integer.
<max>	**Parameter Type**: Decimal value. Is an unsigned 64-bit integer.

Return Value Type: Hex value.

RCL.PORT() UDP Port number of remote API interface

[build 59804 - DVD 02/2015]

Syntax: **RCL.PORT(*<index>*)**

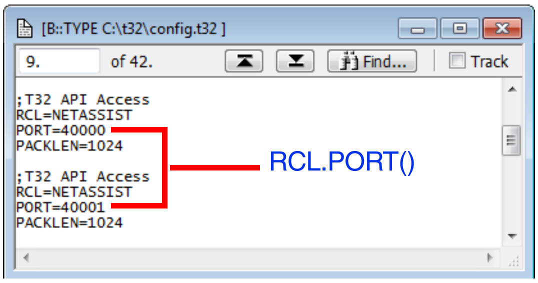
Returns the UDP port number used by the currently selected TRACE32 PowerView instance for communicating with *<index>* via the remote API interface. Returns 0 if the port number is undefined.

Parameter and Description:

<index>	**Parameter Type**: Decimal value. Stands for the client that connects to TRACE32 via the remote API. More than one client can connect to TRACE32. The port number is defined in the TRACE32 configuration file (default c:\t32\config.t32).

Return Value Type: Decimal value.

Example:



```
;T32 API Access
RCL=NETASSIST
PORT=40000
PACKLEN=1024

;T32 API Access
RCL=NETASSIST
PORT=40001
PACKLEN=1024
```

```
PRINT RCL.PORT(0.)        ;returns 40000 and
PRINT RCL.PORT(1.)        ;returns 40001 because the API setting in
                          ;the above configuration file reads:
                                  RCL=NETASSIST
                                  PORT=40000 / PORT=40001
```

RCL.TCP.NrUsedCons() Number of remote API clients connected via TCP

[build 148343 - DVD 09/2022]

Syntax: **RCL.TCP.NrUsedCons()**

Returns the number of remote API clients currently connected via TCP.

Return Value Type: Decimal value.

RCL.TCP.PORT() TCP Port number of remote API interface

[build 1222215 - DVD 09/2020]

Syntax:	**RCL.TCP.PORT()**

Returns the TCP port number used by the currently selected TRACE32 PowerView instance for communicating via the remote API interface. Returns 0 if the configuration file does not contain `RCL=NETTCP`.

SOFTWARE Functions

In This Section

See also

❏ SOFTWARE.64BIT()	❏ SOFTWARE.BUILD()	❏ SOFTWARE.BUILD.BASE()	❏ SOFTWARE.VERSION()

SOFTWARE.64BIT() Check if TRACE32 executable is 64-bit

[build 26300 - DVD 11/2010]

Syntax:	**SOFTWARE.64BIT()**

Returns TRUE if the TRACE32 software is a 64-bit executable.

Return Value Type: Boolean.

SOFTWARE.BUILD() Upper build number

[build 13875 - DVD 10/2008]

Syntax:	**SOFTWARE.BUILD()**

Returns the upper build number of TRACE32. Alias for **VERSION.BUILD()**.

Return Value Type: Decimal value.

SOFTWARE.BUILD.BASE() Lower build number

[build 15283 - DVD 10/2008]

Syntax:	**SOFTWARE.BUILD.BASE()**

Returns the lower build number of TRACE32. Alias for **VERSION.BUILD.BASE()**.

Return Value Type: Decimal value.

[build 24375 - DVD 11/2010]

Syntax:	**SOFTWARE.VERSION()**

Returns the version of the main software. For more information about the function, please refer to its alias **VERSION.SOFTWARE()**.

Return Value Type: String.

STRing Functions

In This Section

See also

❏ STRing.CHAR()	❏ STRing.ComPare()	❏ STRing.COUNT()	❏ STRing.CUT()
❏ STRing.ESCapeQuotes()	❏ STRing.FIND()	❏ STRing.LENgth()	❏ STRing.LoWeR()
❏ STRing.MID()	❏ STRing.Replace()	❏ STRing.SCAN()	❏ STRing.SCANAndExtract()
❏ STRing.SCANBack()	❏ STRing.SPLIT()	❏ STRing.TOKEN()	❏ STRing.TRIM()
❏ STRing.UPpeR()			

STRing.CHAR() Extract a character

Syntax: **STRing.CHAR("***<string>***",***<index>***)**

Extracts a character at the given index position of the string.

Return Value Type: Hex value. The function returns 0xFFFFFFFFFFFFFFFF (= -1) if the index exceeds the string length.

Examples:

```
PRINT STRing.CHAR("abcdef",2)           ; result: 0x63 (=='c')

PRINT STRing.CHAR("abcdef",10.)        ; result: 0xFFFFFFFFFFFFFFFF (==-1)

IF STRing.CHAR("abcdef",10.)==0FFFFFFFFFFFFFFFF
   DIALOG.OK "<index> exceeds the string length."

IF STRing.CHAR("abcdef",10.)==-1
   DIALOG.OK "<index> exceeds the string length."
```

STRing.ComPare() Check if string matches pattern

[build 53898 - DVD 09/2014]

Syntax: **STRing.ComPare("*<string>*","*<pattern>*")**

Returns TRUE if the string matches the wildcard pattern containing '*' and '?'.

Parameter and Description:

<string>	**Parameter Type**: String.
<pattern>	**Parameter Type**: String.

Return Value Type: Boolean.

Example:

```
;returns TRUE
PRINT STRing.ComPare(STRing.LoWeR("JohnPaulGeorgeRingo"),"*paul*")
```

STRing.COUNT() Substring occurrences

[build 66950 - DVD 02/2016]

Syntax: **STRing.COUNT("*<string>*","*<substring>*")**

Counts the number of occurrences of *<substring>* in *<string>*.

Parameter and Description:

<string>	**Parameter Type**: String. The original string.
<substring>	**Parameter Type**: String. The string we search for in *<string>*.

Return Value Type: Decimal value.

Example:

```
PRINT STRing.COUNT("Hello World","l")       ; returns 3
PRINT STRing.COUNT("Hello World","ll")      ; returns 1
```

STRing.CUT() Cut string from left or right

Syntax: **STRing.CUT("*<string>*",*<length>*)**

Cuts off the start or end of a string. Positive values cut off the start, negative values cut off the end of the string.

Parameter and Description:

<string>	**Parameter Type**: String. String to be modified.
<length>	**Parameter Type**: Hex or decimal value.

Return Value Type: String.

Examples:

```
PRINT STRing.CUT("abcdef",1)                        ; result "bcdef"
PRINT STRing.CUT("abcdef",-1)                       ; result "abcde"

&abc="test"
&def=STRing.CUT("&abc",2)                           ; result &def="st"
PRINT "&def"
```

STRing.ESCapeQuotes() Double quote character inside string

[build 94943- DVD 09/2018]

Syntax: **STRing.ESCapeQuotes("*<string>*")**

This function takes a string as parameter and doubles all `"` characters inside it.

Parameter Type: String.

Return Value Type: String.

As a use case for this function, here is an example PRACTICE script:

```
LOCAL &mycmd &mystr
&mystr="  starts with space, has "" double quotes "", ends with space  "
&mycmd="PRINT ""mystr:>"+STRing.ESCapeQuotes("&mystr")+"<"""
&mycmd
```

This script will produce the following output in the AREA window:

```
mystr:>  starts with space, has " double quotes ", ends with space  <
```

STRing.FIND() Check if search characters are found within string

Syntax: **STRing.FIND("*<string1>*","*<string2>*")**

Checks if *<string1>* and *<string2>* share at least 1 character.

Parameter Type: String.

Return Value Type: Boolean.

Example 1:

```
PRINT STRing.FIND("auto","sample")          ; result TRUE
PRINT STRing.FIND("abc","xyz")              ; result FALSE

IF STRing.FIND("auto","sample")==TRUE()
(
   DIALOG.OK "The two string share at least one character."
)
```

Example 2: In this script, the **STRing.FIND()** function is used to check if a file name contains characters from the blacklist.

```
LOCAL &blackList
&blackList="\/:*?""<>| -"

IF STRing.FIND("hello world.dat","&blackList")==TRUE()
(
  PRINT %ERROR "File name contains illegal characters!"
  ENDDO
)
```

See also: **STRing.SCAN()**.

STRing.LENgth() Length of string

[build 27143 - DVD 06/2011]

Syntax: **STRing.LENgth("*<string>*")**

Returns the length of the string.

Return Value Type: Decimal value.

Example:

```
STRing.LENgth("abcDEF")                       ; result 6
```

STRing.LoWeR()   String to lowercase

[build 27143 - DVD 06/2011]

Syntax: **STRing.LoWeR("*<string>*")**

Returns the string converted into lowercase.

Parameter Type: String.

Return Value Type: String.

Example:

```
STRing.LoWeR("abcDEF")                        ; result "abcdef"
```

STRing.MID()   Extract part of string

Syntax: **STRing.MID("*<string>*",*<start_at>*,*<length>*)**

Extracts a part of a string. The string starting at position *<start_at>* with the *<length>* is extracted. The first element has the offset 0.

Parameter and Description:

<string>	**Parameter Type**: String. String to be modified.
<start_at>	**Parameter Type**: Hex or decimal value.
<length>	**Parameter Type**: Hex or decimal value.

Return Value Type: String.

Examples:

```
STRing.MID("abcdef",2.,2.)                    ; result "cd"
STRing.MID("abcdef",2.,100.)                  ; result "cdef"
STRing.MID("abcdef",10.,100.)                 ; result ""
```

Syntax: **STRing.Replace("***<source_string>***","***<search_string>***","***<replace_string>***",** *<no_replaces>***)**

Scans the *<source_string>* for the occurrence of *<search_string>* and replaces these string parts by *<replace_string>*. An empty *<replace_string>* will result in a string cutting. The function returns a modified string.

Parameter and Description:

*<*string*>*	**Parameter Type**: String.
<no_replaces>	**Parameter Type**: Hex or decimal value. Defines the number of replacements:
	0 Replace all occurrences of *<search_string>*.
	1..n Number of replacements from string begin.
	-1..-n Number of replacements from string end.

Return Value Type: String.

Examples:

```
  &unix_path=STRing.Replace(OS.PWD(),"\","/",0.)

  &my_path="~~\testdir"
  &unix_path=STRing.Replace(&my_path,"\","/",0.)
; no expansion of "~~" will be done

  PRINT   STRing.Replace("abcdefgabcdefgabcdefgabcdefg","cd","123",0.)
; result:                 ab123efgab123efgab123efgab123efg

  PRINT   STRing.Replace("abcdefgabcdefgabcdefgabcdefg","cd","",0.)
; result:                 abefgabefgabefgabefgabefg

  PRINT   STRing.Replace("abcdefgabcdefgabcdefgabcdefg","acd","12",0.)
; result:                 abcdefgabcdefgabcdefgabcdefg

  PRINT   STRing.Replace("abcdefgabcdefgabcdefgabcdefg","cd","123",2.)
; result:                 ab123efgab123efgabcdefgabcdefg

  PRINT   STRing.Replace("abcdefgabcdefgabcdefgabcdefg","cd","123",-2.)
; result:                 abcdefgabcdefgab123efgab123efg

  PRINT   STRing.Replace("aaaaaaaaaa","aaa","123",2.)
; result:                 123123aaaa

  PRINT   STRing.Replace("aaaaaaaaaa","aaa","123",-1.)
; result:                 aaaaaaa123
```

STRing.SCAN()

Syntax: **STRing.SCAN("***<source_string>***","***<search_string>***",***<start_at>***)**

Scans the *<source_string>* for the first occurrence of *<search_string>*.
The search begins at the offset *<start_at>*. The first string element has index 0.
The function returns the offset of the found string or -1 if the string was not found.

Parameter and Description:

<source_string>, *<search_string>*	**Parameter Type**: String.
<start_at>	**Parameter Type**: Hex or decimal value.

Return Value Type: Hex value.

Examples:

```
PRINT STRing.SCAN("abcdefabcde","cd",0)          ;result 2
PRINT STRing.SCAN("abcdefabcde","cd",3)          ;result 8
PRINT STRing.SCAN("abcdefabcde","xy",0)          ;result -1
```

Syntax: **STRing.SCANAndExtract("***<string>***","***<key>***","***<default_value>***")**

Scans the *<string>* for the first occurrence of *<key>* and extracts a subsequent value. Use to parse key-value pairs of a configuration string in the form of "*<key><value>*".

This function is space sensitive. *<key>* must be preceded by a space character, unless it is at the start of the string. *<value>* ends at the first space character found or at the end of *<string>*. *<value>* can include whitespace characters if put in quotes.

The function returns the value found after the *<key>*. If *<key>* was not found *<default_value>* is returned.

Parameter and Description:

<string>, *<key>*, *<default_value>*	**Parameter Type**: String.

Return Value Type: String.

Examples:

```
LOCAL &parameters &device_id &node &name

ENTRY %LINE &parameters
; e.g. &parameters = DEVICEID=7 NODE= NAME="ETH 1"

&device_id=STRing.SCANAndExtract("&parameters","DEVICEID=","0")
; &device_id = 7

&id=STRing.SCANAndExtract("&parameters","ID=","-1")
; &id = -1, as there is no key "ID=" in &parameters

&node=STRing.SCANAndExtract("&parameters","NODE=","DEFAULT")
; &node is empty

&name=STRing.SCANAndExtract("&parameters","NAME=","anonymous")
; &name = "ETH 1"
```

Syntax:	**STRing.SCANBack("***<source_string>***","***<search_string>***",***<start_at>***)**

Scans the *<source_string>* for the first occurrence of *<search_string>*.
The search begins backwards at the offset *<start_at>*. The first string element has index 0.
The function returns the offset of the found string or -1 if the string was not found.

Parameter and Description:

<source_string>, *<search_string>*	**Parameter Type**: String.
<start_at>	**Parameter Type**: Hex or decimal value.

Return Value Type: Hex value.

Examples:

```
PRINT STRing.SCANBack("abcdefabcdcde","cd",0)                 ;result -1
PRINT STRing.SCANBack("abcdefabcde","cd",3)                   ;result  2
PRINT STRing.SCANBack("abcdefabcde","xy",10.)                 ;result -1

PRINT STRing.SCANBack("aaaaa","aaa",4)                        ;result  2
PRINT STRing.SCANBack("aaaaa","aaa",3)                        ;result  1
PRINT STRing.SCANBack("aaaaa","aaa",1)                        ;result -1

PRINT STRing.SCANBack("aaaaa","aaa",STRing.LENgth("aaaaa")-1) ;result  2
```

[build 60314 - DVD 02/2015]

Syntax:	**STRing.SPLIT("***<string>***","***<separator>***",***<index>[,<options>]**)**

Splits a string in parts at the given *separator* and returns the resulting element at the specified index. If a negative index is used, the elements are counted backwards (e.g -1: last element, -2: second last...).

As *separator* the parameter *<separator>* is interpreted as a list of characters (`DELIMITER=char`) or as a string (default: `DELIMITER=STRING`). Per default there is no special handling for quotes. With option `QUOTEDSTRINGS=ON` tokenization is disabled between matching quotes. Using an escape character (`ESCAPECHAR='<c>'`) *quotes* as well as a *separator* can be escaped.

Parameter and Description:

<string>	String to split. **Parameter Type**: String.
<separator>	Separator string / Sequence of character separators respectively depending on DELIMITER= option. **Parameter Type**: String.
<index>	Index of item. **Parameter Type**: Hex or decimal value.

<options>	Available from build 143582 - DVD 02/2022
	Optional: Key-Value string
	`QUOTEDSTRINGS=off|ON`
	`DELIMITER=string|CHAR`
	`ESCAPECHAR='<c>'`
	Parameter Type: String.

Options and Description:

QUOTEDSTRINGS=ON	A token is not generated between matching quotes. E.g. `key="value with spaces"` with separator *<space>* generates one token.
DELIMITER=string	String *<separator>* is used as *separator.*
DELIMITER=CHAR	A *separator* is any character part of *<separator>*.
ESCAPECHAR='<c>'	Escape *separator*'s and *quotes* using `<c>`. With `QUOTEDSTRINGS=ON` `<c>` is not removed from the final token within a quoted string. The escape char itself can be escaped using `<c><c>`.

Return Value Type: String.

Examples:

```
PRINT STRing.SPLIT("Hello TRACE32!"," ",0)          ; result "Hello"
PRINT STRing.SPLIT("C:\T32\demo\arm","\",2)         ; result "demo"
PRINT STRing.SPLIT("C:\T32\demo\arm","\",-1)        ; result "arm"
PRINT STRing.SPLIT("C:\T32\demo\arm","demo",0)      ; result "C:\T32\"
```

```
; negative indexes
PRINT STRing.SPLIT("a||b","|",0.)                   ; result "a"
PRINT STRing.SPLIT("a||b","|",1.)                   ; result ""
PRINT STRing.SPLIT("a||b","|",2.)                   ; result "b"
PRINT STRing.SPLIT("a||b","|",-1.)                  ; result "b"
PRINT STRing.SPLIT("a||b","|",-2.)                  ; result ""
PRINT STRing.SPLIT("a||b","|",-3.)                  ; result "a"
; negative indexes with no match
PRINT STRing.SPLIT("abcd","|",0.)                   ; result "abcd"
PRINT STRing.SPLIT("abcd","|",1.)                   ; result ""
PRINT STRing.SPLIT("abcd","|",-1.)                  ; result "abcd"
; leading/trailing separator
PRINT STRing.SPLIT("|b||","|",0.)                   ; result ""
PRINT STRing.SPLIT("|b||","|",1.)                   ; result "b"
PRINT STRing.SPLIT("|b||","|",-3.)                  ; result "b"
```

```
; Demonstrate QUOTEDSTRINGS=ON
PRIVATE &i &str
&i=0.
&str="DRINK=""Lemonade,Apple juice,Tonic Water"" TIP=5$ PAID=TRUE"
RePeaT 3.
(
  PRIVATE &sToken
  &sToken=STRing.SPLIT("&str"," ",&i,QUOTEDSTRINGS=ON)
  PRINTF "; Index %2u - %s" &i “&sToken”
  &i=&i+1.
)
; Result:
; Token 0 - DRINK="Lemonade,Apple juice,Tonic Water"
; Token 1 - TIP=5$
; Token 2 - PAID=TRUE
; Demonstrate ESCAPECHAR='<c>'
PRIVATE &i &str
&i=0.
&str="COMMAND=""awk -f \""/scripts 2022/test.awk\"" output.txt"" "
&str="&(str) KEY\ WITH\ SPACES=VALUE\ WITH\ SPACES "
&str="&(str) BACKSLASH=\\ "
RePeaT 5.
(
  PRIVATE &sToken
  &sToken=STRing.SPLIT("&str"," ",&i,QUOTEDSTRINGS=ON ESCAPECHAR='\')
  PRINTF "; Index %2u - %s" &i "&sToken"
  &i=&i+1.
)
; Result:
; Index 0 - COMMAND="awk -f \"/scripts 2022/test.awk\" output.txt"
; Index 1 -
; Index 2 - KEY WITH SPACES=VALUE WITH SPACES
; Index 3 -
; Index 4 - BACKSLASH=\
; Empty indexes 1&3 are caused by multiple <space>'s in &str
```

See also **STRing.TOKEN()**.

```
; Demonstrate DELIMITER=CHAR
SUBROUTINE splitPath
(
  PRIVATE &i &sPath
  PARAMETERS &sPath
  &i=0.
  RePeaT 4.
  (
    PRIVATE &sToken
    &sToken=STRing.SPLIT("&sPath","/\",&i,DELIMITER=CHAR)
    PRINTF "; Index %2u - %s" &i "&sToken"
    &i=&i+1.
  )
)
GOSUB splitPath "..\..\demo\arm"
; Result:
; Index 0 - ..
; Index 1 - ..
; Index 2 - demo
; Index 3 - arm
GOSUB splitPath "../../demo/arm"
; Result:
; Index 0 - ..
; Index 1 - ..
; Index 2 - demo
; Index 3 - arm
```

STRing.TOKEN()

Syntax:	**STRing.TOKEN("***<string>***","***<delimiter>***",***<index>*[,*<options>*]**)**

STRing.TOKEN is aligned to C/C++ strtok(). The functions extracts tokens from *<string>* which are sequences of contiguous characters separated by *separator*'s. To extract a token the function scans *<string>* till the first location not a *separator*. Sequences of consecutive *separator*'s between/after a token are ignored.

As *separator* the parameter *<delimiter>* is interpreted as a list of characters (default: `DELIMITER=char`) or as a string (`DELIMITER=STRING`). Per default there is no special handling for quotes. With option `QUOTEDSTRINGS=ON` tokenization is disabled between matching quotes. Using an escape character (`ESCAPECHAR='<c>'`) *quotes* as well as a *separator* can be escaped.

If a negative index is used, the tokens are counted backwards (e.g -1: last element, -2: second last...).

Parameter and Description:

<string>	String to tokeninze. **Parameter Type**: String.
<delimiter>	Sequence of delimiters / Delimiter string respectively depending on DELIMITER= option. **Parameter Type**: String.
<index>	Index of token. **Parameter Type**: Hex or decimal value.

<options>	Optional: Key-Value string
	`QUOTEDSTRINGS=off│ON`
	`DELIMITER=char│STRING`
	`ESCAPECHAR='<c>'`
	Parameter Type: String.

Options and Description:

QUOTEDSTRINGS=ON	A token is not generated between matching quotes. E.g. `key="value with spaces"` with separator *<space>* generates one token.
DELIMITER=char	A *separator* is any character part of *<delimiter>*, like C/C++ strtok().
DELIMITER=STRING	String *<delimiter>* is used as *separator*.
ESCAPECHAR='<c>'	Escape *separator*'s and *quotes* using `<c>`. With `QUOTEDSTRINGS=ON` `<c>` is not removed from the final token within a quoted string. The escape char itself can be escaped using `<c><c>`.

Return Value Type: String.

Examples:

```
PRIVATE &i &str
&i=0.
&str="String separated with one or multiple spaces, colons or \
      semicolons ;."
RePeaT 11.
(
  PRINTF "; Token %2u - %s" &i STRing.TOKEN("&str"," ,;",&i)
  &i=&i+1.
)
; Result:
; Token 0 - String
; Token 1 - separated
; Token 2 - with
; Token 3 - one
; Token 4 - or
; Token 5 - multiple
; Token 6 - spaces
; Token 7 - colons
; Token 8 - or
; Token 9 - semicolons
; Token 10 - .
```

```
; negative indexes
PRINT STRing.TOKEN("a||b","|",0.)                     ; result "a"
PRINT STRing.TOKEN("a||b","|",1.)                     ; result "b"
PRINT STRing.TOKEN("a||b","|",2.)                     ; result ""
PRINT STRing.TOKEN("a||b","|",-1.)                    ; result "b"
PRINT STRing.TOKEN("a||b","|",-2.)                    ; result "a"
PRINT STRing.TOKEN("a||b","|",-3.)                    ; result ""
; negative indexes with one token
PRINT STRing.TOKEN("abcd","|",0.)                     ; result "abcd"
PRINT STRing.TOKEN("abcd","|",1.)                     ; result ""
PRINT STRing.TOKEN("abcd","|",-1.)                    ; result ""
; leading/trailing separator’s
PRINT STRing.TOKEN("|b||","|",0.)                     ; result "b"
PRINT STRing.TOKEN("|b||","|",1.)                     ; result ""
PRINT STRing.TOKEN("|b||","|",-1.)                    ; result ""
```

```
; Demonstrate QUOTEDSTRINGS=ON
PRIVATE &i &str
&i=0.
&str="DRINK=""Lemonade,Apple juice,Tonic Water"" TIP=5$ PAID=TRUE"
RePeaT 3.
(
  PRIVATE &sToken
  &sToken=STRing.TOKEN("&str"," ",&i,QUOTEDSTRINGS=ON)
  PRINTF "; Token %2u - %s" &i "&sToken"
  &i=&i+1.
)
; Result:
; Token 0 - DRINK="Lemonade,Apple juice,Tonic Water"
; Token 1 - TIP=5$
; Token 2 - PAID=TRUE
```

```
; Demonstrate ESCAPECHAR='<c>'
PRIVATE &i &str
&i=0.
&str="COMMAND=""awk -f \""/scripts 2022/test.awk\"" output.txt"" "
&str="&(str) KEY\ WITH\ SPACES=VALUE\ WITH\ SPACES "
&str="&(str) BACKSLASH=\\ "
RePeaT 3.
(
  PRIVATE &sToken
  &sToken=STRing.TOKEN("&str"," ",&i,QUOTEDSTRINGS=ON ESCAPECHAR='\')
  PRINTF "; Token %2u - %s" &i "&sToken"
  &i=&i+1.
)
; Result:
; Token 0 - COMMAND="awk -f \"/scripts 2022/test.awk\" output.txt"
; Token 1 - KEY WITH SPACES=VALUE WITH SPACES
; Token 2 - BACKSLASH=\
```

STRing.TRIM() String without leading and trailing whitespaces

[build 34324 - DVD 02/2012]

Syntax: **STRing.TRIM("*<string>*")**

Returns a string without leading and trailing whitespaces.

Parameter Type: String.

Return Value Type: String.

Examples:

```
; Remove whitespaces.
PRINT STRing.TRIM("    abcDEF   ")                  ; result "abcDEF"

; Remove whitespaces and convert to upper case.
PRINT STRing.TRIM(STRing.UPpeR("     abcDEF    ")) ; result "ABCDEF"
```

STRing.UPpeR()   String to uppercase

[build 27143 - DVD 06/2011]

Syntax: **STRing.UPpeR("***<string>***")**

Returns the string converted to uppercase.

Parameter Type: String.

Return Value Type: String.

Example:

```
PRINT STRing.UPpeR("abcDEF")                ;result ABCDEF
```

TCF Functions (TRACE32 as TCF Agent)

For information about how to configure and use TRACE32 as TCF agent, refer to **“TRACE32 as TCF Agent”** (app_tcf_setup.pdf).

In This Section

See also

❏ TCF.DISCOVERY()   ❏ TCF.PORT()

TCF.PORT() Port number of TCF interface

[build 71558 - DVD 02/2016]

Syntax:	**TCF.PORT()**

Returns the port number used by the currently selected TRACE32 PowerView instance for communication via the TCF interface. Returns 0 if the port number is undefined.

Return Value Type: Decimal value.

TCF.DISCOVERY() Check if TCF discovery is enabled

[build 71558 - DVD 02/2016]

Syntax:	**TCF.DISCOVERY()**

Returns TRUE if the TCF discovery is enabled in TRACE32.

Return Value Type: Boolean.

TEST.TIMEISVALID() Check if time value is valid

[build 78580 - DVD 02/2017]

Syntax: **TEST.TIMEISVALID(*<time>*)**

Returns TRUE if the given time value is correct. Returns FALSE otherwise.

This function is not intended for syntax checking of strings containing time values.

Parameter Type: Time value.

Return Value Type: Boolean.

Example:

```
IF TEST.TIMEISVALID(Trace.RECORD.TIME(-12.))==FALSE()
(
  PRINT %WARNING "timestamp of record -12. isn’t valid"
)
```

TIMEOUT() Check if command was fully executed

[build 45047 - DVD 08/2013]

Syntax: **TIMEOUT()**

Returns TRUE if a previous command terminated due to a timeout. The commands which can be terminated by a timeout are:

WAIT [*<condition>*] [*<period>*] [build 94995 - DVD 09/2015]	Waits until a condition is true or a period has elapsed. The screen is not updated while waiting.
TIMEOUT *<period> <command>* [build 45047 - DVD 08/2013]	Specifies a timeout for a TRACE32 command.
SCREEN.WAIT [*<condition>* | *<period>*] [build 94995 - DVD 09/2015]	Updates the screen while waiting.

Return Value Type: Boolean.

Example 1:

```
Go.direct main          ;set temporary breakpoint on main()
                        ;function and start CPU

WAIT !STATE.RUN() 5.s   ;wait 5.s for CPU to stop

IF TIMEOUT()==TRUE()
   ECHO %ERROR "CPU does not stop."
```

Example 2:

```
;your start-up script

TIMEOUT 500.ms Data.Copy D:0--0x3ffffff VM:0 /Byte /Verify

IF TIMEOUT()==TRUE()
(
  PRINT %WARNING "'Data.Copy D:0--0x3ffffff VM:0' canceled after 50.ms"
)
```

Example 3:

```
TERM.TRIGGER #1 "[U-BOOT]"
SCREEN.WAIT TERM.TRIGGERED(#1) 15.s
IF TIMEOUT()
    STOP %ERROR "Failed to reach prompt"
```

TITLE()

Caption of the TRACE32 main window

[build 24341 - DVD 11/2010]

Syntax:	**TITLE()**

Returns the caption of the TRACE32 main window. The caption can be modified with the **TITLE** command.

Return Value Type: String.

Example:

```
TITLE "TRACE32 for CPU0"                ; Define the caption of the
                                        ; TRACE32 main window
 …
IF TITLE()=="TRACE32 for CPU0"          ; Check if the actual caption
                                        ; matches a certain debugger
                                        ; scenario
```

TRUE Function

TRUE()

Boolean expression

[build 36180 - DVD 02/2012]

Syntax: **TRUE()**

Always returns the boolean value TRUE. It can be used for increasing the readability of PRACTICE scripts when initializing PRACTICE macros. The counterpart is **FALSE()**.

Return Value Type: Boolean.

Example:

```
&s_error_occurred=TRUE()              ; instead of
&s_error_occurred=(0==0)
```

WARNINGS Function

WARNINGS()

Check if warning occurred during command execution

Syntax: **WARNINGS()**

Returns TRUE if a warning occurred during command execution (only applicable in PRACTICE script files *.cmm).

Return Value Type: Boolean.

WINdow Functions

NOTE:	•	Window names are case-sensitive.
	•	Page names are case-sensitive.

In This Section

See also

■ Win	❏ WINdow.COMMAND()	❏ WINdow.EXIST()	❏ WINdow.LIST()
❏ WINdow.POSition()	❏ WINPAGE.CURRENT()	❏ WINPAGE.EXIST()	❏ WINPAGE.LIST()

WINdow.COMMAND() Command string displayed in window

[build 69465 - DVD 02/2016]

Syntax: **WINdow.COMMAND(WinTOP** | *<window_name>***)**

Returns the TRACE32 command which was used to open a window. The device prompt, e.g. `B::`, is included in the return value.

Parameter and Description:

WinTOP (or **TOP** as an alias)	Returns the TRACE32 command of the active window.
<window_name>	**Parameter Type**: String. Window names are case-sensitive. They are created with the **WinPOS** command.

Return Value Type: String. The string is empty if the specified window name does not exist.

Example:

```
WinPOS ,,,,,, abc               ;assign the name 'abc' to the next window
List.auto                       ;open the window

PRINT WINdow.COMMAND(abc)       ;prints "B::List.auto" to the message AREA
```

WINdow.EXIST() Check if window name exists

Syntax: **WINdow.EXIST(***<window_name>***)**
WINDOW.NAME() (deprecated)

Returns TRUE if a window with the specified name exists.

Parameter Type: String. Window names are case-sensitive. They are created with the **WinPOS** command or the **DIALOG** NAME element.

Return Value Type: Boolean.

Example:

```
WinPOS ,,,,,,abc               ;create a window with a user-defined name
Register.view
 …
IF WINdow.EXIST(abc)==TRUE()   ;check if this window still exists
```

WINdow.LIST() Generate a comma-separated list of window names

[build 146569 - DVD 08/2022]

Syntax: **WINdow.LIST(**[<page_name>]**)**

Returns a comma-separated list of window names of all open windows either on all window pages, on a specific window page or the currently selected window page. If no open window exist the returned string is empty.

Parameter and Description:

no parameter	**Parameter Type**: none. If no parameter is given the function returns a list of all window names of all opened windows on all window pages.
<page_name>	**Parameter Type**: String. Name of a window page created with the **WinPAGE.Create** command. Page names are case-sensitive. The returned comma-separated list contains all window names of all opened windows for the given window page

Return Value Type: String.

NOTE:	The list may contain duplicate window names because the window name could be a user-defined value that is not a unique identifier for a window. See command **WinPOS** how to set the name of a window. The order of the window names in the list can vary between calls to the function and depends on the overlapping order (Z-order) of the windows on the screen.

Examples:

```
PRINT WINdow.LIST()        ; no parameter will print a list of all window
                           ; names on all window pages
PRINT WINdow.LIST("P000") ; print a list of all window names on page P000
```

WINdow.POSition() Information on window position and dimension

[build 64413 - DVD 09/2015]

Syntax:	**WINdow.POSition(WinTOP** | *<window_name>*,*<position_item_name>***)**
<position_item_name>:	**LEFT** | **UP** | **HSIZE** | **VSIZE** | **HSCALE** | **VSCALE**

Returns the current value of the specified position type if a window with the specified name exists.

Parameter and Description:

WinTOP (or **TOP** as an alias)	Returns the values of the active window.
<window_name>	**Parameter Type**: String. Window names are case-sensitive. They are created with the **WinPOS** command.
<position_item_name>	For a description of **LEFT** to **VSCALE**, refer to the parameters of the **WinPOS** command.

Return Value Type: Float.

Examples:

```
WinPOS 10.25 20.50 80. 25. 15. 2. abc ; create a window with defined
List.auto                             ; name, position and size

PRINT WINdow.POSition(abc,left)       ; ==> 10.25
PRINT WINdow.POSition(abc,up)         ; ==> 20.50
PRINT WINdow.POSition(abc,hsize)      ; ==> 80.0
PRINT WINdow.POSition(abc,vsize)      ; ==> 25.0
PRINT WINdow.POSition(abc,hscale)     ; ==> 15.0
PRINT WINdow.POSition(abc,vscale)     ; ==>  2.0
```

WINPAGE.CURRENT() Get name of currently selected window page

[build 146627 - DVD 08/2022]

Syntax: **WINPAGE.CURRENT()**

Returns the name of the currently selected window page. Use the command **WinPAGE.select** to change the currently selected window page or the command **WinPAGE.Create** to create a new named window page.

Return Value Type: String.

Example:

```
LOCAL &page &list
&page=WINPAGE.CURRENT()
&list=WINdow.LIST(&page)
PRINT "Windows on current page &page: &list"
```

WINPAGE.EXIST() Check if window page exists

Syntax: **WINPAGE.EXIST(*<page_name>*)**

Returns TRUE if a window page with the specified name exists.

Parameter Type: String. Name of a window page created with the **WinPAGE.Create** command. Page names are case-sensitive.

Return Value Type: Boolean.

Example:

```
WinPAGE.Create  Analyzer                ; create separate window page
Trace.List                              ; with defined name
 …
IF WINPAGE.EXIST(Analyzer)==TRUE()      ; check if this window page still
                                        ; exists
```

WINPAGE.LIST() Generate comma-separated list of page names

[build 146654 - DVD 08/2022]

Syntax: **WINPAGE.LIST()**

Returns a comma-separated list containing the names of all existing window pages.

Return Value Type: String.

A new named window page is created with the command **WinPAGE.Create**. An interactive window displaying all window pages and their included windows is shown with the command **WinPAGE.List**.

Example:

```
LOCAL &pagelist                       ; define macro
WinPAGE.Create "MYPAGE" /NoSELect     ; create window page named MYPAGE
&pagelist=WINPAGE.LIST()              ; get list of all page names
PRINT "list:&pagelist"                ; shows "list:P000,MYPAGE"
```