

PRACTICE Reference Card

Requires TRACE32® Release 2023/09 or Later

PRACTICE, Lauterbach's own scripting language, is used to write start-up scripts, automate tests, and to customize the TRACE32® GUI. The file extension is *.cmm.

For test automation, TRACE32® also supports Python.

RUN A PRACTICE SCRIPT

In TRACE32®, choose **FILE** menu → **RUN SCRIPT**.
Or at the TRACE32® command line, type `CD.DO *`.
Or drag and drop the PRACTICE file into TRACE32®'s command line.

CREATE AND DEBUG PRACTICE SCRIPTS

PEDIT *	Open an editor to create a script
PSTEP *	Open a script for debugging
PMACRO.RESet	Delete all PRACTICE macros
PBREAK.set 5. file.cmm	Set a breakpoint in line 5 of script
PBREAK.List	View, add, edit, delete breakpoints
PLIST	Show the currently active script

RECORD TRACE32® COMMANDS

1.	LOG.OPEN "~/t32.log" ; open file t32.log
2.	; execute some TRACE32 commands
3.	; or do mouse actions here
4.	LOG.CLOSE ; close file and stop logging function

HISTORY.SAVE will just save the history of command line entries.

COMMENTS IN SCRIPT FILES

Single-line comments start with `;` or `//`.
Multi-line comments do not exist.
The command **Var.** allows just `//` as comment beginning keyword.

GET HELP FOR A COMMAND

To get the details of a specific command, type the command in the TRACE32® command line, add a **SPACE**, then press **F1**.

DEMO FILES

Choose **HELP** menu → **DEMO SCRIPTS**.

COPY CURRENT TRACE32® SETTINGS

ClipSTore <item> copies a script for the selected item to your clipboard, where **<item>** is for example:

1.	Win	Currently open windows and their positions
2.	SYStem	SYStem.state and SYStem.CONFIG
3.	Break	All breakpoints shown in Break.List window

PATH PREFIXES AND FUNCTIONS

./	Working directory	OS.PresentWorkingDirectory()
~/	User's home folder	OS.PresentHomeDirectory()
~/	System directory	OS.PresentSystemDirectory()
~/	Temporary directory	OS.PresentTemporaryDirectory()
~/	Active script location	OS.PresentPracticeDirectory()

Example:

1.	DIR ~/
2.	; example for use of path function
3.	LOCAL &psd
4.	&psd= OS.PresentSystemDirectory()
5.	D0 "&psd/hardware/s32g2/s32g2_connect_m7_0.cmm"

TRACE32® can handle forward slashes `/` on all operating systems.

LITERALS

Decimal	197.
Float	197.0 or 9.75 or 1.3e+34.
Hexadecimal	0xC5
Binary	0y0011000101
Bitmask	0y0011xx01xx
Hexmask	0x10xx
Character	'a'
Address	P:0x100
Addr. with Segment	P:0x02:0x100
Addr. with Machine	NP:1::0x02:0x100
Address Range	P:0x100--0x1ff or P:0x100++0x0ff
Boolean	TRUE() or FALSE()
String	"abc" or "escape "quotes" "
HLL Symbol	main or `main` ; backticks for symbol with special characters

OPERATOR PRECEDENCE

1.	() { }	Brackets	(highest priority)
2.	-- ++ ..	Ranges	
3.	+ - ~ !	Signs, Binary NOT, Logical NOT	
4.	<< >>	Shift operations	
5.	* / %	Multiplication, Division, Modulo	
6.	+ - +	Add, Subtract, Concatenate	
7.	== != >= <= > <	Comparisons	
8.	&	Binary AND	
9.	^	Binary XOR	
10.		Binary OR	
11.	&&	Logical AND	
12.	^^	Logical XOR	
13.		Logical OR	(lowest priority)

NOTE: No whitespaces before or after operators, since whitespaces are interpreted as separators.

WHITESPACE

PRACTICE is **WHITESPACE SENSITIVE**.

There must be at least one after every command, e.g.:

WHILE (&i>5)	is wrong (error: unknown command)
WHILE _(&i>5)	is correct

Expressions like `(5+0x20)[0x100]` must not contain any whitespaces.

DECLARE AND INITIALIZE PRACTICE MACROS (VARIABLES)

1.	PMACRO.EXPLICIT	; enforce explicit declaration
2.	GLOBAL &SessionStart	
3.	LOCAL &msg1 &started &linecount	
4.	PRIVATE &val1 &val2	
5.	&SessionStart= DATE.DATE()	
6.	&msg1="Hello World!"	; no spaces at =
7.	&started= TRUE()	
8.	&linecount=0.	; note the trailing dot
9.	&val1= Var.VALUE (flags[3])	; assign value of ; C variable
10.	&val2= Data.Byte (D:0x345f)	; assign a memory value

PRACTICE macros (variables) are simple text buffers. Macros are supported only in PRACTICE script files and are maintained on the PRACTICE stack. Check if an existing macro is initialized or not with:

IF "&var"!=""

Private macros exist inside the declaring block and are erased when the block ends. They are only visible in the declaring block and sub-blocks (IF..., WHILE..., etc.).

Local macros are like the private ones but are additionally visible inside their sub-routines, and sub-scripts.

Global macros are visible everywhere. They are not erased when the declaring file or block ends and stay alive after the script execution has finished or no script is executed.

TRACE32®-INTERNAL C-STYLE VARIABLES

1.	; create integer \i on local PRACTICE stack frame
2.	Var.NEWLOCAL int \i
3.	; create character array \myStr
4.	; on global PRACTICE stack frame
5.	Var.NEWGLOBAL char[10][128] \myStr
6.	Var.Set \i=0x42
7.	Var.Set \myStr[5]="hello"
8.	ECHO %Hex Var.VALUE (\i)
9.	ECHO Var.STRING (\myStr[5])
10.	Var.View %all \i \myStr

FUNCTIONS ON THE COMMAND LINE

1.	; show CPU selected with SYStem.CPU
2.	ECHO SYStem.CPU ()
3.	; show value of environment variable USER
4.	ECHO OS.ENV (T32ID)

IF... ELSE IF... ELSE...

```

1. IF CPUIS("A2F200")
2.   &FlashSize=0x40000
3. ELSE IF CPUIS("A2F500")
4.   &FlashSize=0x80000
5. ELSE
6.   (
7.     ; use brackets for command block
8.     ECHO %ERROR "FLASH size of CPU type is unknown"
9.     ENDDO
10.  )

```

WHILE LOOP

```

1. PRIVATE &i
2. &i=0.
3. WHILE &i<10.
4.   (
5.     ECHO "Count: " &i
6.     &i=&i+1.
7.   )

```

REPEAT WHILE LOOP (DO-WHILE LOOP)

```

1. PRIVATE &lic
2.
3. OPEN #1 "~~~/license.t32" /Read
4. RePeaT
5.   (
6.     READ #1 %LINE &lic
7.     ECHO "&lic"
8.   )
9. WHILE !FILE.EOF(LASTREAD()) ; loop if not end of file
10. CLOSE #1

```

UNCONDITIONAL REPETITION AND PAUSING SCRIPTS

```

1. ECHO "Please wait 5. sec "
2.
3. RePeaT 50. ; loop 50 times
4.   ( ; increase progress bar
5.     ECHO %CONTINUE "*" ; write * at end of
6.     ; previous line
7.     WAIT 100.ms ; pause script for 100.ms
8.   )

```

WAIT !STATE.RUN() 2.s wait until the optional condition (!STATE.RUN()) is true and maximal specified period (two seconds).

ADD A TEMPORARY BUTTON TO THE TOOLBAR

```
MENU.AddTool "Test" "TS,G" "PEDIT test.cmm"
```

For a permanent button, add the above command to your default start-up script, usually **PEDIT ~/system-settings.cmm**.

YES-NO DIALOG

```

1. PRIVATE &yes
2. DIALOG.YESNO "Do you want to close TRACE32?"
3. ENTRY &yes
4. IF &yes
5.   QUIT 0

```

The script above makes sense when the following command is added to **system-settings.cmm** or any other start-up script:

```
SETUP.QUITDO "~~~/myquit.cmm"
```

CREATE A CUSTOM DIALOG

```

1. ; embed a dialog description block in *.cmm file
2. DIALOG
3.   (
4.     HEADER "Calculator"
5.     ; POS: next element position (x,y,width,height)
6.     POS 0. 0. 25. 1.
7.     TEXT "Value:"
8.     valueField: EDIT "42." ""
9.     BUTTON "Add" "GOSUB CalcAndShow +"
10.    BUTTON "Subtract" "GOSUB CalcAndShow -"
11.    TEXT "Result:"
12.    resultField: EDIT "0." ""
13.    INIT ; INIT block called when dialog opens
14.    (
15.      ; make macro &result available in all dialog functions
16.      DIALOG.STORAGE.define &result
17.      &result=0.
18.      DIALOG.DISable resultField
19.    )
20.    ; close block called on dialog close event
21.    CLOSE
22.    (
23.      ECHO "Final result = &result"
24.      DIALOG.END
25.    )
26.    SUBROUTINE CalcAndShow
27.    (
28.      ENTRY &op
29.      &value=DIALOG.STRING(ValueField)
30.      &result=&result&op&value
31.      DIALOG.Set resultField "&result"
32.    )
33.  )
34. )

```

CUSTOM EVENT HANDLERS

```

1.   (
2.     ; set up error event handler for this block
3.     ON ERROR GOSUB loadError
4.     Data.LOAD.ELF "demo.elf" ; file might not exist
5.   )
6. SUBROUTINE loadError
7.   (
8.     ECHO %ERROR "Cannot load file"
9.     ENDDO
10.  )

```



PARAMETER EXCHANGE WITH A SUB-ROUTINE

```

1. PRIVATE &failed &addr
2. Data.PATTERN VM:0x00--0xff /RANDOM
3. GOSUB copydata "VM:0x00" "VM:0x100" "16."
4. RETURNVALUES &failed &addr
5.
6. IF &failed
7.   ECHO "Failed at &addr"
8. ENDDO
9.
10. SUBROUTINE copydata
11.   (
12.     PARAMETERS &from &to &size
13.     PRIVATE &err &eaddr
14.     Data.COPY &from++(&size-1) &to
15.     Data.ComPar &from++(&size-1) &to
16.     &err=FOUND() ; return TRUE() or FALSE()
17.     &eaddr=ADDRESS.OFFSET(TRACK.ADDRESS())
18.     RETURN "&err" "&eaddr"
19.   )

```

PARAMETER EXCHANGE WITH A SUB-SCRIPT

caller-world.cmm (file1)

```

1. PRIVATE &answer
2. DO "~~~/world.cmm" "Hello World!"
3. RETURNVALUES &answer
4. DIALOG.OK &answer
5. ENDDO

```

world.cmm (file 2)

```

1. ; implicitly declares &msg as private macro
2. ; and assigns the passed argument value to it
3. PARAMETERS &msg
4. PRIVATE &result
5. DIALOG.MESSAGE "&msg"
6. &result="Hello User!"
7. ENDDO "&result"

```

TRACE32® START-UP PARAMETERS

At the command prompt of your operating system:

```
t32m* [-c <conf> [arg]] [-s <script> [arg]]
```

Example:

```
t32marm -c config.t32 20010 -s mysetup.cmm USER
```

In your TRACE32® configuration file (config.t32) e.g.:

```

1. IC=NETASSIST
2. PORT=${1} ; use parameterized port number

```

In your start-up script (mysetup.cmm) e.g.:

```

1. LOCAL &parameter ; declare local macro
2. ENTRY &parameter ; assign parameter value
3. ECHO "Env. variable: "+OS.ENV(&parameter)

```