

## PRACTICE Reference Card

PRACTICE, the Lauterbach script language, is used for automating tests, configuring the TRACE32 GUI and your debug environment. The file extension is \*.cmm.

### Run a PRACTICE Script

In TRACE32, choose **File** menu → **Run Script**.  
Or at the TRACE32 command line, type `DO file.cmm`.  
Or drag and drop the PRACTICE file into the command line.

### Create and Debug PRACTICE Scripts

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

### Path Prefixes

./	Current working directory	OS.PWD ()
~/	Home directory of user	OS.PHD ()
~/	System directory of TRACE32	OS.PSD ()
~/	Temporary directory of TRACE32	OS.PTD ()
~/	Directory of the current script	OS.PPD ()

TRACE32 can handle forward slashes / on all operating systems.

### 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 and Icon Library

Choose **Help** menu → **Demo Scripts**.

Choose **Misc** menu → **Tools** → **Display internal icon library**.

```
DO "~/demo/menu/internal_icons.cmm"
```

### Copy Current TRACE32 Settings

ClipSToRe <item> copies a script for the selected item to your clipboard, where <item> is one of the following:

Win	Currently open windows and their positions
SYStem	SYSTEM.state and SYSTEM.CONFIG
Break	All breakpoints shown in Break.List window
MAP	Memory mapping shown in MAP.List
Analyzer	Trace configuration of your PowerTrace

### Record TRACE32 Commands

```
LOG.OPEN "~/~/t32.log" ;Open file t32.log
```

;Execute some TRACE32 commands or do mouse actions here

```
LOG.CLOSE ;Close file and terminate logging function
```

Or save the current command history with `HISTory.SAVE`

### Literals

Decimal	197.
Float	197.0 or 9.75
Hexadecimal	0xC5
Binary	0y0011000101
Bitmask	0y0011xx01xx
Hexmask	0x10XX
Boolean	TRUE () or FALSE () or &i<20.
String	"abc" or "escape ""quotes"" "
Character	'a'
Address	P:0x100
Addr. with Segment	P:0x02:0x100
Address Range	P:0x100--0x1ff P:0x100+0x0fxf
HLL Symbol	`main` // These `` are backticks.

### Operator Precedence

1.	( ) { }	Parentheses, curly braces (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

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

### Declare and Initialize PRACTICE Macros (Variables)

```
PMACRO.EXPLICIT //Enforce explicit macro declaration
GLOBAL &SessionStart
LOCAL &msg1 &started &linecount
PRIVATE &val1 &val2

&SessionStart=CLOCK.DATE ()
&msg1="Hello World!" //No spaces at =
&started=TRUE ()
&linecount=0. //Note the trailing dot
&val1=Var.VALUE (flags+3) //Assign value of C variable
&val2=Data.Byte (flags+3) //Assign a memory value
```

PRACTICE macros (variables) are simple text buffers. Macros are supported only in PRACTICE script files. Check if an existing macro is initialized or not with: `IF "&var"!=""`

**Local macros** exist inside the declaring block and are erased when the block ends. They are visible inside their blocks, sub-blocks, sub-routines, and sub-scripts.

**Private macros** are like the local ones but are only visible in the declaring block and sub-blocks.

**Global macros** are visible everywhere. They are not erased when the declaring file or block ends.

### TRACE32-Internal Variables (C-Style Variables)

```
//Create integer \i on local PRACTICE stack frame
Var.NEWLOCAL int \i
//Create character array \myStr on global PRACTICE stack frame
Var.NEWGLOBAL char[10][128] \myStr

Var.Set \i=0x42
Var.Set \myStr[5]="hello"

PRINT %Hex Var.VALUE (\i)
PRINT Var.STRING (\myStr[5])

Var.View %all \i \myStr
```

### Comments in Script Files

Single-line comments start with ; or //  
In multi-line comments, comment out each line.  
You cannot start a comment with ; in the same line as a command starting with V. a Var.

### Whitespace

PRACTICE is **whitespace sensitive**. There must be at least one space 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.

## IF... ELSE IF... ELSE...

```
SYStem.DETEct IDCode           //Get JTAG ID
IF IDCODE(0.)==0x049220DD      //Excalibur chip
(
  SYStem.CPU EPXA
)
ELSE IF IDCODE(0.)==0x4BA00477 //MK60 chip
(
  SYStem.CPU MK60DN512VMD10
)
ELSE
(
  SYStem.CPU ARM7TDMI
)
```

## While Loop

```
PRIVatE &i
&i=0.

WHILE &i<10.           ;Loop while &i is smaller 10
(
  PRINt "Count: " &i
  &i=&i+1.
)
```

## Repeat While Loop (do-while loop)

```
PRIVatE &lic
OPEN #1 "~/license.t32" /Read

RePeaT
(
  READ #1 %LINE &lic
  PRINt "&lic"
)
WHILE !FILE.EOFLASTREAD() ;Loop if not end of file
CLOSE #1
```

## Unconditional Repetition and Pausing Scripts

```
PRINt "Please wait 5. sec "RePeaT 50.           ;Loop 50 times
(
  PRINt %CONTInue "*" ;Increase progress bar
  WAIT 100.ms         ;Write * at end of previous line
                     ;Pause script for 100.ms
)
```

WAIT 1.s can be used to wait for e.g. a target bootloader.

## Functions on the Command Line

```
PRINt SYStem.CPU() ;Show CPU selected with SYStem.CPU
PRINt OS.ENV(USER) ;Show value of environment variable
                     ;USER
```

## Parameter Exchange with a Sub-Routine

```
PRIVatE &result &addr

Data.PATtERN VM:0x00--0xff /RANDOM
GOSUB copydata "VM:0x00" "VM:0x100" "16."
RETURNVALUES &result &addr

IF !&result
  PRINt "Failed at &addr"
ENDDO

copydata:           ;Labels must start in the first column!
  PARAMETERs &from &to &size
  PRIVatE &err &addr

  Data.COpy &from++(&size-1) &to
  Data.CoMpare &from++(&size-1) &to
  &err=!FOUND() ;Returns TRUE() or FALSE()

  &addr=ADDRESS.OFFSET(TRACK.ADDRESS())
  RETURN "&err" "&addr"
```

## Parameter Exchange with a Sub-Script

caller-world.cmm (file 1)

```
PRIVatE &answer

DO "~~~/world.cmm" "Hello World!"
RETURNVALUES &answer
DIALOG.OK "&answer"

ENDDO
```

world.cmm (file 2)

```
PARAMETERs &msg // Creates private macro with param.value
PRIVatE &result

DIALOG.MESSAGe "&msg"
&result="Hello User!"

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 t32.cmm USER
```

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

```
IC=NETASSIST
PORT=${1} ;Use parameterized port number
```

In your start-up script (t32.cmm) e.g.:

```
LOCAL &parameter ;Declare local macro
ENTRY &parameter ;Assign parameter value to macro
PRINt "Env. variable: "+OS.ENV(&parameter)
```

## Custom Event Handlers

```
ON.ERROR.GOSUB myHandler //Set up error event handler

Data.LOAd.Elf "demo.elf" //File might not exist
ON.ERROR.inherit //Disable error handler
ENDDO

myHandler: PRINt %ERROR " Can't load file"
           RETURN
```

## 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 `C:\T32\t32.cmm`.

## Yes-No Dialog

```
PRIVatE &yes

DIALOG.YESNO "Do you want to close TRACE32?"
ENTRY &yes

IF &yes
  QUIT 0
```

This script makes sense when the following command is added to `t32.cmm` or any other start-up script:

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

## Create a Custom Dialog

```
DIALOG ;Embed a dialog description block in a *.cmm file
(
  ;In dialog blocks, comments must be placed in separate lines
  HEADER "Basic Dialog Structure"
  ICON ":chip"
  ;x, y, width, and height of next dialog element
  POS 2. 0.5 12. 1.
  TEXT "Hello World!"
  textbox: EDIT "dummy text" ""

  POS 30. 4.5 8. 1.25
  DEFBUTTON "OK" "CONTInue"

  ;Handle [X] button and ESC key
  ;CLOSE must be inside the dialog block
  CLOSE "GOTO closeDialog"
)

DIALOG.Set textbox "Hello User!"
DIALOG.Disable textbox ;Gray out text box
STOP ;Wait for the user's response to the dialog

PRINt "Clicked OK!"
PRINt "Message: " DIALOG.STRING(textbox)
closeDialog: ;Labels must start in the first column!
DIALOG.END
```