LAUTERBACH
DEVELOPMENT TOOLS

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

```
IF CPUIS("A2F200")
(
   &FlashSize=0x40000
)
ELSE IF CPUIS("A2F500")
(
   &FlashSize=0x80000
)
ELSE
   ECHO %ERROR "FLASH size of CPU type is unknown"
```

WHILE Loop

```
PRIVATE &i
&i=0.
WHILE  &i<10.
(
   ECHO "Count: " &i
   &i=&i+1.
)
```

Repeat While Loop (do-while loop)

```
PRIVATE &lic

OPEN #1 "~~/license.t32" /Read
RePeaT
(
   READ #1 %LINE &lic
   ECHO "&lic"
)
WHILE !FILE.EOFLASTREAD() ; loop if not end of file
CLOSE #1
```

Unconditional Repetition and Pausing Scripts

```
ECHO "Please wait 5. sec  "

RePeaT 50.              ; loop 50 times
(                       ; increase progress bar
   ECHO %CONTinue "*"   ; write * at end of
                        ; previous line
   WAIT 100.ms          ; pause script for 100.ms
)
```

`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

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

```
; embed a dialog description block in *.cmm file
DIALOG
(
   HEADER "Calculator"
  ; POS: next element position (x,y,width,height)
   POS 0. 0. 25. 1.
   TEXT "Value:"
   valueField: EDIT "42." ""
   BUTTON "Add"      "GOSUB CalcAndShow +"
   BUTTON "Subtract" "GOSUB CalcAndShow -"
   TEXT "Result:"
   resultField: EDIT "0." ""
   INIT   ; INIT block called when dialog opens
   (
      ; make macro &result available in all
      ; dialog functions
      DIALOG.STORAGE.define &result
      &result=0.
      DIALOG.DISable resultField
   )
   ; close block called on dialog close event
   CLOSE
   (
      ECHO "Final result = &result"
      DIALOG.END
   )
   SUBROUTINE CalcAndShow
   (
      ENTRY &op
      &value=DIALOG.STRing(ValueField)
      &result=&result&op&value
      DIALOG.Set resultField "&result"
   )
)
```

Custom Event Handlers

```
(
   ; set up error event handler for this block
   ON ERROR GOSUB loadError
   Data.LOAD.Elf "demo.elf" ; file might not exist
)
SUBROUTINE loadError
(
   ECHO %ERROR "Cannot load file"
   ENDDO
)
```

Parameter Exchange with a Sub-Routine

```
PRIVATE &failed &addr
Data.PATTERN VM:0x00--0xff /RANDOM
GOSUB copydata "VM:0x00" "VM:0x100" "16."
RETURNVALUES &failed &addr

IF &failed
   ECHO "Failed at &addr"
ENDDO

SUBROUTINE copydata
(
   PARAMETERS    &from &to &size
   PRIVATE       &err &eaddr
   Data.COPY     &from++(&size-1) &to
   Data.ComPare &from++(&size-1) &to
   &err=FOUND()   ; return TRUE() or FALSE()
   &eaddr=ADDRESS.OFFSET(TRACK.ADDRESS())
      RETURN "&err" "&eaddr"
)
```

Parameter Exchange with a Sub-Script

caller-world.cmm (file1)

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

world.cmm (file 2)

```
; implicitly declares &msg as private macro
; and assigns the passed argument value to it
PARAMETERS &msg
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 mysetup.cmm USER
```

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

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

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

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

LAUTERBACH
DEVELOPMENT TOOLS

PRACTICE Reference Card

Requires TRACE32 Release 2022/02 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 in scripts
`PLIST`	Show the currently active script

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 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:

`Win`	Currently open windows and their positions
`SYStem`	SYStem.state and SYStem.CONFIG
`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:

```
DIR ~/
; example for use of path function
LOCAL &psd
&psd=OS.PresentSystemDirectory()
DO &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)

```
PMACRO.EXPLICIT   ; enforce explicit declaration
GLOBAL &SessionStart
LOCAL &msg1 &started &linecount
PRIVATE &val1 &val2
&SessionStart=DATE.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(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

```
; 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"
ECHO %Hex Var.VALUE(\i)
ECHO Var.STRing(\myStr[5])
Var.View %all \i \myStr
```

Functions on the Command Line

```
; show CPU selected with SYStem.CPU
ECHO SYStem.CPU()
; show value of environment variable USER
ECHO OS.ENV(T32ID)
```