DATE ................................................................................................................ ......................... 32
  Display date and time 32

DEL ................................................................................................................. .......................... 33
  Delete file 33

DIALOG .............................................................................................................. ...................... 34
  Custom dialogs 34
  Dialog Definition Programming Commands 34
    BAR ................................................. Progress bar 35
    BOX ............................................ Define a decorative border 36
    BUTTON .................................... Raised button with an icon and text 37
    CHECKBOX .................................. Define a checkbox 39
    CHOOSEBOX ................................. Define a choose box 40
    CLOSE ....................................... Catch window close 42
    COMBOBOX ................................. Define a combo box 42
    DEFBUTTON ................................ Define the default button 43
    DEFCOMBOBOX ............................. Define a default combo box 43
    DEFEDIT .................................... Define a default edit control 43
    DEFHOTCOMBOBOX ..................... Define a default hot combo box 43
    DEFHOTEDIT ............................... Define a default hot edit control 43
    DEFMEDIT ................................ Define a default multiline edit control 44
    DLISTBOX .................................. Define a draggable list box 44
    DYNAMIC .................................. Dynamic, single-line area 45
    DYNCOMBOBOX ............................ Define a dynamic combo box 46
    DYNDEFCOMBOBOX ...................... Define a default dynamic combo box 46
    DYNDEFHOTCOMBOBOX ................ Define a dynamic default hot combo box 46
    DYNHOTCOMBOBOX ...................... Define a dynamic hot combo box 46
    DYNLTEXT ................................ Dynamic single-line text area in bold and large font size 47
    DYNPULLDOWN ............................. Define a dynamic pull-down list 48
    DYNTTEXT ................................ Dynamic, single-line text area in regular font size 50
    EDIT ......................................... Define an edit control 51
    HEADER .................................... Define window header 52
    HELP ........................................ Define a help icon 53
    HOTEDIT .................................... Define a hot edit control 54
    HOTCOMBOBOX ............................ Define a hot combo box 55
    ICON ........................................ New icon in top left corner of dialog 55
    INFOTEXT ................................ Define a multiline info text box on a dialog 56
    LINE ........................................ Define a decorative horizontal line 58
    LISTBOX .................................... Define a list box 59
    LTEXT ....................................... Static, single-line text area in bold and large font size 60
    LEDIT ........................................ Define an edit control in bold and large font 61
    MEDIT ....................................... Define a multiline edit control 61
    MLISTBOX ................................ Define a multiline list box 61
    NAME ....................................... Internal dialog name 62
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POS</td>
<td>Define position and size</td>
</tr>
<tr>
<td>POSX</td>
<td>Define position and size on the x-axis</td>
</tr>
<tr>
<td>POSY</td>
<td>Define position and size on the y-axis</td>
</tr>
<tr>
<td>PULLDOWN</td>
<td>Define a static pull-down list</td>
</tr>
<tr>
<td>SPACE</td>
<td>Apply previous height to next dialog element</td>
</tr>
<tr>
<td>STATIC</td>
<td>Place an icon in a dialog</td>
</tr>
<tr>
<td>TEXT</td>
<td>Static, single-line text area in regular font size</td>
</tr>
<tr>
<td>TEXTBUTTON</td>
<td>Flat button with text only</td>
</tr>
<tr>
<td>TREEBUTTON</td>
<td>Implements a +/- toggle button</td>
</tr>
<tr>
<td>UPDATE</td>
<td>Executes commands periodically</td>
</tr>
<tr>
<td>VLINE</td>
<td>Decorative vertical line</td>
</tr>
<tr>
<td>DIALOG.AREA</td>
<td>Adds an output area to a custom dialog</td>
</tr>
<tr>
<td>DIALOG.DIR</td>
<td>Display a folder picker dialog</td>
</tr>
<tr>
<td>DIALOG.Disable</td>
<td>Disable dialog elements</td>
</tr>
<tr>
<td>DIALOG.Enable</td>
<td>Enable dialog elements</td>
</tr>
<tr>
<td>DIALOG.END</td>
<td>Close the dialog window</td>
</tr>
<tr>
<td>DIALOG.EXecute</td>
<td>Execute a dialog button</td>
</tr>
<tr>
<td>DIALOG.File</td>
<td>Pass file name from OS file dialog to PRACTICE script</td>
</tr>
<tr>
<td>DIALOG.File.open</td>
<td>Display an OS file-open dialog</td>
</tr>
<tr>
<td>DIALOG.File.SAVE</td>
<td>Display an OS file-save dialog</td>
</tr>
<tr>
<td>DIALOG.File.SELECT</td>
<td>Display an OS file-select dialog</td>
</tr>
<tr>
<td>DIALOG.MESSAGE</td>
<td>Create dialog box with an information icon</td>
</tr>
<tr>
<td>DIALOG.OK</td>
<td>Create dialog box with an exclamation mark</td>
</tr>
<tr>
<td>DIALOG.Program</td>
<td>Interactive programming</td>
</tr>
<tr>
<td>DIALOG.ReProgram</td>
<td>Dialog programming</td>
</tr>
<tr>
<td>DIALOG.SESelect</td>
<td>Programmatically focus on this dialog</td>
</tr>
<tr>
<td>DIALOG.Set</td>
<td>Modify the value of a dialog element</td>
</tr>
<tr>
<td>DIALOG.SetDIR</td>
<td>Browse for folder</td>
</tr>
<tr>
<td>DIALOG.SetFile</td>
<td>Pass file name from OS file dialog to custom dialog</td>
</tr>
<tr>
<td>DIALOG.SetFile.open</td>
<td>OS file-open dialog &gt; file name &gt; EDIT element</td>
</tr>
<tr>
<td>DIALOG.SetFile.SAVE</td>
<td>OS file-save dialog &gt; file name &gt; EDIT element</td>
</tr>
<tr>
<td>DIALOG.SetFile.SELECT</td>
<td>OS file-select dialog &gt; file name &gt; EDIT element</td>
</tr>
<tr>
<td>DIALOG.view</td>
<td>Show dialog window</td>
</tr>
<tr>
<td>DIALOG.YESNO</td>
<td>Create dialog box with YES and NO buttons</td>
</tr>
<tr>
<td>DIR</td>
<td>List subdirectories and files</td>
</tr>
<tr>
<td>DUMP</td>
<td>Binary file dump</td>
</tr>
<tr>
<td>EDIT</td>
<td>TRACE32 editor</td>
</tr>
<tr>
<td>EDIT.CLOSE</td>
<td>Close a text file</td>
</tr>
<tr>
<td>EDIT.EXtern</td>
<td>Use specified external ASCII editor to edit file</td>
</tr>
</tbody>
</table>
HISTORY  ............................................................................................................. ........................ 131
HISTORY Command history of last executed commands 131
HISTORY.eXecute Execute command history 131
HISTORY.SAVE Store command history log 132
HISTORY.Set History settings 133
HISTORY.SIZE Command history and file history 134
HISTORY.SIZE.cmd Define log size of command history 134
HISTORY.SIZE.FILE Define number of recently used files in 'File' menu 135
HISTORY.type Display command history log of last executed commands 135

IFCONFIG  ............................................................................................................. ..................... 136
IFCONFIG Ethernet or USB communication 136
IFCONFIG.PROfile Display operation profiles 136
IFCONFIG.TEST Test interface function and speed 138
IFCONFIG.state Interface configuration 139

INTERCOM  ............................................................................................................. ................... 141
INTERCOM Data exchange between different TRACE32 systems 141
INTERCOM.Evaluate Evaluate function via INTERCOM 142
INTERCOM.execute Execute command via INTERCOM 143
INTERCOM.executeNoWait Execute command via INTERCOM 143
INTERCOM.PING Test INTERCOM system 144
INTERCOM.PipeCLOSE Close named pipe 144
INTERCOM.PipeOPEN Open named pipe 145
INTERCOM.PipeREAD Read from named pipe 145
INTERCOM.PipeWRITE Write to named pipe 145
INTERCOM.WAIT Wait for remote INTERCOM system 146

LICENSE  ............................................................................................................. ...................... 147
LICENSE Manage TRACE32 licenses 147
LICENSE.List Display all license information 147
LICENSE.REQuest Request a license 148
LICENSE.state Display the currently used maintenance contract 149
LICENSE.UPDATE Update the maintenance contract 150

LOG  ................................................................................................................. .......................... 151
LOG Log TRACE32 commands 151
LOG.CLOSE Close command log 151
LOG.OFF Switch off command log 152
LOG.ON Switch on command log 152
LOG.OPEN Open command log file 153
LOG.toAREA Log commands by writing them to an AREA window 154
LOG.type Display command log 158

LS  .................................................................................................................. ............................ 158
LS Display directory 158

©1989-2018 Lauterbach GmbH
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>MENU</td>
<td>Customize the user interface TRACE32 PowerView</td>
<td>159</td>
</tr>
<tr>
<td>MENU.AddMenu</td>
<td>Add one standard menu item</td>
<td>159</td>
</tr>
<tr>
<td>MENU.AddTool</td>
<td>Add a button to the main toolbar</td>
<td>160</td>
</tr>
<tr>
<td>MENU.Delete</td>
<td>Delete nested menu</td>
<td>161</td>
</tr>
<tr>
<td>MENU.Program</td>
<td>Interactive programming</td>
<td>161</td>
</tr>
<tr>
<td>MENU.ReProgram</td>
<td>Menu programming</td>
<td>162</td>
</tr>
<tr>
<td>MENU.RESet</td>
<td>Default configuration</td>
<td>164</td>
</tr>
<tr>
<td>Programming Commands</td>
<td>Add definition to existing menu</td>
<td>165</td>
</tr>
<tr>
<td>ADD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDHERE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AFTER</td>
<td>Place a new menu option or separator after the named menu option</td>
<td>166</td>
</tr>
<tr>
<td>BEFORE</td>
<td>Place a new menu option or separator before the named menu option</td>
<td>166</td>
</tr>
<tr>
<td>BUTTONS</td>
<td>Add user-defined local buttons to a window</td>
<td>167</td>
</tr>
<tr>
<td>DEFAULT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DELETE</td>
<td>Delete a certain item</td>
<td>168</td>
</tr>
<tr>
<td>ELSE</td>
<td>Conditional compile</td>
<td>168</td>
</tr>
<tr>
<td>ENABLE</td>
<td>Conditional enable</td>
<td>169</td>
</tr>
<tr>
<td>HELP</td>
<td>Define a help item</td>
<td>170</td>
</tr>
<tr>
<td>IF</td>
<td>Conditional compile</td>
<td>170</td>
</tr>
<tr>
<td>MENU</td>
<td>Menu definition</td>
<td>171</td>
</tr>
<tr>
<td>MENUITEM</td>
<td>Item definition</td>
<td>173</td>
</tr>
<tr>
<td>POPUP</td>
<td>Popup definition</td>
<td>173</td>
</tr>
<tr>
<td>REPLACE</td>
<td>Replace the following item</td>
<td>174</td>
</tr>
<tr>
<td>SEPARATOR</td>
<td>Separator definition</td>
<td>174</td>
</tr>
<tr>
<td>TEAROFF</td>
<td>Define tearoff menu</td>
<td>174</td>
</tr>
<tr>
<td>TOOLBAR</td>
<td>Toolbar definition</td>
<td>175</td>
</tr>
<tr>
<td>TOOLITEM</td>
<td>Item definition</td>
<td>175</td>
</tr>
<tr>
<td>WIDTH</td>
<td>Increase/decrease button width</td>
<td>177</td>
</tr>
<tr>
<td>MKDIR</td>
<td>Create new directory</td>
<td>178</td>
</tr>
<tr>
<td>MV</td>
<td>Rename file</td>
<td>179</td>
</tr>
<tr>
<td>OS</td>
<td>Execute host commands</td>
<td>180</td>
</tr>
<tr>
<td>OS.Area</td>
<td>Re-route host command output to AREA window</td>
<td>181</td>
</tr>
<tr>
<td>OS.Command</td>
<td>Execute a host command</td>
<td>182</td>
</tr>
<tr>
<td>OS.Hidden</td>
<td>Execute a host command in silent mode</td>
<td>184</td>
</tr>
<tr>
<td>OS.screen</td>
<td>Call up the shell or execute host command</td>
<td>185</td>
</tr>
<tr>
<td>OS.Window</td>
<td>Re-route host command output to the OS.Window</td>
<td>187</td>
</tr>
<tr>
<td>PACK</td>
<td>Compress files (with LZW algorithm)</td>
<td>188</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>PATCH</td>
<td>Binary file patching</td>
<td></td>
</tr>
<tr>
<td>PATH</td>
<td>Define searchpath</td>
<td></td>
</tr>
<tr>
<td>PRinTer</td>
<td>Print and export window contents</td>
<td></td>
</tr>
<tr>
<td>PRinTer.Area</td>
<td>Re-route printer output to AREA window in specified format</td>
<td></td>
</tr>
<tr>
<td>PRinTer.ClipBoard</td>
<td>Re-route printer output to clipboard in specified format</td>
<td></td>
</tr>
<tr>
<td>PRinTer.CLOSE</td>
<td>Close file after multiple printer outputs</td>
<td></td>
</tr>
<tr>
<td>PRinTer.EXPORT</td>
<td>Export formatted printer output to file</td>
<td></td>
</tr>
<tr>
<td>PRinTer.FILE</td>
<td>Re-route printer output to a file in specified file format</td>
<td></td>
</tr>
<tr>
<td>PRinTer.FileType</td>
<td>Select file format</td>
<td></td>
</tr>
<tr>
<td>PRinTer.HardCopy</td>
<td>Make a hardcopy of the screen</td>
<td></td>
</tr>
<tr>
<td>PRinTer.OFFSET</td>
<td>Specify print-out borders</td>
<td></td>
</tr>
<tr>
<td>PRinTer.OPEN</td>
<td>Re-route multiple printer outputs to the same file</td>
<td></td>
</tr>
<tr>
<td>PRinTer.PRINT</td>
<td>Print to opened printer file</td>
<td></td>
</tr>
<tr>
<td>PRinTer.select</td>
<td>Select printer</td>
<td></td>
</tr>
<tr>
<td>PRinTer.SIZE</td>
<td>Specify print-out size</td>
<td></td>
</tr>
<tr>
<td>PWD</td>
<td>Change directory</td>
<td></td>
</tr>
<tr>
<td>QUIT</td>
<td>Return to operating system</td>
<td></td>
</tr>
<tr>
<td>REN</td>
<td>Rename file</td>
<td></td>
</tr>
<tr>
<td>RM, RMDIR</td>
<td>Delete file, Remove directory</td>
<td></td>
</tr>
<tr>
<td>SCreenShot</td>
<td>Save a screenshot of a window to a file</td>
<td></td>
</tr>
<tr>
<td>SETUP</td>
<td>Setup commands (part 1 of 2)</td>
<td></td>
</tr>
<tr>
<td>SETUP.ASCIITEXT</td>
<td>Configure ASCII text display</td>
<td></td>
</tr>
<tr>
<td>SETUP.BAKfile</td>
<td>Set backup file mode</td>
<td></td>
</tr>
<tr>
<td>SETUP.CClear</td>
<td>Erase caches</td>
<td></td>
</tr>
<tr>
<td>SETUP.COLOR</td>
<td>Change colors</td>
<td></td>
</tr>
<tr>
<td>SETUP.DEVNAME</td>
<td>Set logical device name</td>
<td></td>
</tr>
<tr>
<td>SETUP.DRV</td>
<td>Send command to driver</td>
<td></td>
</tr>
<tr>
<td>SETUP.EDITEXT</td>
<td>Define an external editor</td>
<td></td>
</tr>
<tr>
<td>SETUP.EXTension</td>
<td>Set default file name extensions</td>
<td></td>
</tr>
<tr>
<td>SETUP.HOLDDIR</td>
<td>Configure working directory</td>
<td></td>
</tr>
<tr>
<td>SETUP.ICONS</td>
<td>Display icons in popup menus</td>
<td></td>
</tr>
</tbody>
</table>
IDE Reference Guide

**SETUP.INTERCOMACKTIMEOUT** Sets the intercom acknowledge timeout 226

**SETUP.MEMory** Display free and used SCU memory 226

**SETUP.PDFViewer** Context-sensitive help via your favorite PDF viewer 227

**SETUP.PDFViewer.EXEcutable** Path and executable of your PDF viewer 228

**SETUP.PDFViewer.OPEN** Open a PDF of the help system 228

**SETUP.PDFViewer.PRinT** Print PDF via HELP window 228

**SETUP.PDFViewer.RESet** Reset the settings in SETUP.PDFViewer dialog 229

**SETUP.PDFViewer.TEMPorary** Help configuration for demo purposes 230

**SETUP.PDFViewer.TEMPorary.EXEcutable** PDF viewer for demo purposes 230

**SETUP.PDFViewer.TEMPorary.OPEN** Open a PDF of the help system 230

**SETUP.PDFViewer.TEMPorary.PRinT** Print PDF via HELP window 230

**SETUP.PDFViewer.TEMPorary.RESet** Reset demo-help configuration 231

**SETUP.QUITDO** Define quit PRACTICE script file 232

**SETUP.RADIX** Radix mode 233

**SETUP.RANDOM** Set seed for RANDOM() function 234

**SETUP.ReDraw** Update whole screen 234

**SETUP.RESTARTDO** Define restart PRACTICE script file 234

**SETUP.SOUND** Set sound generator mode 235

**SETUP.STATistic** Cache and memory usage 235

**SETUP.STOre** Configure output of the STOre commands 236

**SETUP.TabSize** Configure tab width 237

**SETUP.TIMEFORM** Select scientific time format 238

**SETUP.UpdateRATE** Update rate for windows 239

**SETUP.WARNSTOP** Configure PRACTICE stops 240

**SETUP.XSLTSTYLESHEET** Reference to XSLT stylesheet for XML files 241

**SILENT** Suppress informational messages in AREA window 242

**SOFTKEYS** Toggle the buttons on the softkey bar 243

**STATUSBAR** Toggle state line 244

**STOre** Store a setting 245

**STOre** Store a setting 245

**SUBTITLE** Define a window subtitle for AMP debugging 246

**TAR** Pack files into an archive 248

**TIMEOUT** Specify time-out for TRACE32 command 250

**TITLE** Define a screen title for TRACE32 PowerView GUI 251

©1989-2018 Lauterbach GmbH
TOOLBAR ............................................................................................................. 252
 TOOLBAR  Toggle toolbar 252

TYPE ....................................................................................................................... 253
 TYPE  Display text file 253

UNARchive ............................................................................................................ 254
 UNARchive  Linux and Microsoft libraries 254
 UNARchive.extract  Extract files from Linux library and Microsoft library 254
 UNARchive.Show  Extract files from library and list them in window 255
 UNARchive.Table  Display table of contents of library 255

UNPACK ............................................................................................................. 256
 UNPACK  Expand files (with LZW algorithm) 256

UNZIP ................................................................................................................. 257
 UNZIP  Expand GZIP archive file (with DEFLATE algorithm) 257

VERSION ............................................................................................................. 258
 VERSION  TRACE32 version information 258
 VERSION.ENVironment  Display environment settings 258
 VERSION.HARDWARE  Display hardware versions 259
 VERSION.SOFTWARE  Display software versions 259
 VERSION.view  Display window with version info 260

WELCOME ........................................................................................................... 261
 WELCOME  Welcome to TRACE32 261
 WELCOME.CONFIG  Configure search paths for PRACTICE demo scripts 261
 WELCOME.CONFIG.ADDDIR  Add a new script search path 262
 WELCOME.CONFIG.FILTER  Set the script search path 262
 WELCOME.CONFIG.ReMoveDIR  Remove a script search path 262
 WELCOME.CONFIG.RESet  Reset the script search configuration 262
 WELCOME.CONFIG.state  Open the welcome config window 263
 WELCOME.SCRIPTS  Open the script search window 264
 WELCOME.STARTUP  Open the welcome window if not disabled 264
 WELCOME.view  Open the welcome window 265

Window .............................................................................................................. 266
 Win  Window handling (size, position, font size, etc.) 266
 WinBack  Generate background window 267
 WinCLEAR  Erase windows 267
 WinDEFaultSIZE  Apply a user-defined default size to windows 269
 WinDuplicate  Duplicate window 270
 WinExt  Generate external window 270
 WinFIND  Search for text in window 271
 WinFreeze  Generate frozen window 273
 WinLarge  Generate window with large font 274
 WinMid  Generate window with regular font 274
WinOverlay  Pile up windows on top of each other 275
WinPAGE  Window pages 276
WinPAGE.Create  Create and select page 277
WinPAGE.Delete  Delete page 277
WinPAGE.List  Display an overview of all pages and their windows 278
WinPAGE.REName  Rename page 279
WinPAGE.RESet  Reset window system 279
WinPAGE.select  Select page 279
WinPAN  Specify a window cut-out 280
WinPOS  Define window dimensions and window name 281
WinPrint  Print window 284
WinPRT  Hardcopy of window 284
WinResist  Generate a resistant window 285
WinRESIZE  New size for window 286
WinSmall  Generate window with small font 287
WinTABS  Specify size of re-sizable columns 287
WinTOP  Bring window to top 288
WinTrans  Generate transparent window 289
ZERO  ................................................................................................................ ........................ 290
ZERO.offset  Set time reference 290
ZERO.RESet  Reset to original value 290
ZIP  ................................................................................................................. ............................ 291
ZIP  Compress files to GZIP archive (with DEFLATE algorithm) 291
Appendix A - Help Filters  ........................................................................................... ............. 292
History

15-Nov-17 You can display the online help for last executed commands by right-clicking in the HISTORY.type window and selecting Help.

26-Oct-17 New command TAR.

25-Sep-17 New example and screenshots for the SUBTITLE command.

07-Aug-17 New command group SETUP.STOre with example and screenshot.

27-Jun-17 Added description, example, and screenshot for the new SILENT command.

10-May-17 New dialog control LEDIT.

10-Apr-17 Added description, screenshots, and example for the new command PRinTer.Area.

13-Dec-16 Added description, screenshot, and example for the new commands AREA.List and AREA.Delete.
This command works for the following configurations:

- TRACE32-ICD with PODBUS Ethernet Controller
- TRACE32-ICE
- TRACE32-FIRE with PODBUS Ethernet Controller

This command stops the host driver program. All TRACE32 settings are maintained. After restarting the driver program the user can continue to work, without having to reboot the system. This command, for example, is useful when running compilers on the host system. As opposed to the OS command, no host memory is occupied. In ETHERNET environments, this command can be used to log out from one workstation and continue the work on another workstation. The new workstation must have the same environment (mounted file systems, window system, etc.). When the ABORT command is executed from PRACTICE, the command line arguments of the host driver are passed to the PRACTICE parameters. They can be fetched by the ENTRY command.

```plaintext
E::ABORT ; stop driver on host
... C:>dir ; run command on DOS level
... C:>T32 ; restart host driver
E::Data.dump ; next command on TRACE32 level
```

See also
- QUIT
- SETUP.RESTARTDO
- ‘Program End’ in ‘IDE User's Guide’
Message areas are the IN/OUT windows for error texts or print commands. They work like a standard scrolling terminal. All asynchronous error messages, which appear in the message line, are written to the default message area (named A000), which can be displayed in the AREA.view A000 window. The name of an AREA window is case sensitive, i.e. A000 and a000 are not the same!

If several error messages appear in rapid succession, they can be redisplayed by using the AREA.view command (short form: AREA).

PRACTICE messages can be sent to an AREA window with the PRINT command. Interactive keyboard input on an AREA window can be made with the ENTER command.

Multiple AREA windows may be opened and selected by name. This allows very complex display configurations.

How to save the whole content of a long AREA window? Use the AREA.SAVE command or take a look at this example:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WinPOS ././././. myAreaWin</td>
<td>define a window name for an AREA window</td>
</tr>
<tr>
<td>AREA.view A000</td>
<td>and display the default message area A000</td>
</tr>
<tr>
<td>PRinTer.EXPORT.ASCIIIE C:\area.txt</td>
<td>define file format and name</td>
</tr>
<tr>
<td>WinPAN 0 -999. myAreaWin</td>
<td>scroll back to the first line of the area window (for windows with fewer than 1000. lines)</td>
</tr>
<tr>
<td>WinPRT myAreaWin /ALL</td>
<td>/ALL prints all lines from the visible top of the window to the end</td>
</tr>
</tbody>
</table>

See also
- AREA.CLEAR
- AREA.Close
- AREA.List
- AREA.Select
- AREA.Create
- AREA.Replace
- AREA.Save
- AREA.open
- AREA.View
- LOG.toAREA
- SILENT
AREA.CLEAR

Clear area

Format: `AREA.CLEAR [<area_name>]`

Clears the contents from an AREA window. The `<area_name>` is case sensitive! Alternatively, right-click the AREA window you want, and then select Clear from the popup menu.

Without an `<area_name>`, the default area A000 will be cleared.

See also
- `AREA.CLOSE`
- `AREA.RESet`
- `AREA.Delete`
- `AREA.OPEN`
- `AREA.Create`
- `AREA.Select`
- `AREA.OPEN`

AREA.CLOSE

Close output file

Format: `AREA.CLOSE [<area_name>]`

The output to a file is stopped and the file is closed.

Without an AREA name all AREA output files will be closed.

See also
- `AREA.CLOSE`
- `AREA.RESet`
- `AREA.Delete`
- `AREA.OPEN`
- `AREA.Create`
- `AREA.Select`
- `Area Functions’ in ‘IDE Functions’`
- ‘Message Windows’ in ‘IDE User’s Guide’
AREA.Create
Create or modify message area

Format:  

```
AREA.Create [<area_name> [<columns>] [<lines>]]
```

Creates a new message area or modifies the number of columns and lines of an existing one.

```
<area_name>  The AREA name must not contain the following characters:
    * / \ ' " ; , &
    The AREA name is case sensitive.
```

If you omit the name for the new message area, TRACE32 will use a unique name in the form Axxx, where x will be replace by a decimal digit.

You may create up to 19 additional message areas.

```
AREA.Create A000 60. 100. ; change number of columns and lines of
                      ; the default area
AREA.RESet          ; init area system
AREA.Create XMESSAGE 20. 20. ; create new area named "XMESSAGE"
AREA.view XMESSAGE   ; open window for area "XMESSAGE"
AREA.Select XMESSAGE ; select area for PRINT and ENTER
PRINT "Test"        ; print string constant
AREA.Select A000    ; select standard area
```

See also

- AREA.CLOSE
- AREA
- AREA.Select
- PRINT

▲ ‘Message Windows’ in ‘IDE User’s Guide’
▲ ‘Release Information’ in ‘Release History’
▲ ‘I/O Commands’ in ‘Training Script Language PRACTICE’
Delete message area

Format: 

\texttt{AREA.Delete <area\_name>}

Deletes the specified message area, which has previously been created with \texttt{AREA.Create}, and closes the associated \texttt{AREA} window. You cannot delete the default message area \texttt{A000}.

- If there is no message area of the given name, then \texttt{AREA.Delete} will not show any error.
- If there are multiple \texttt{AREA} windows for the same message area name, then the message area with the specified name will be deleted and all window copies will be closed. For an example, see below.

The following example is for demo purposes only. To try this script, simply copy it to a \texttt{test.cmm} file, and then step through it in TRACE32 (See “How to...”).

```
AREA.Create ephone  ;create the message areas 'ephone'
AREA.Create testlog ;and 'testlog'

AREA.view ephone   ;display the AREA windows for the
AREA.view testlog  ;message areas 'ephone' and 'testlog'

Area.view testlog  ;open two window copies for 'testlog'
Area.View testlog  ;by typing the AREA.view command in
                   ;a different camel casing

AREA.Delete testlog ;delete the message area 'testlog and
                     ;close all three associated AREA windows
```

See also

\begin{itemize}
\item \texttt{AREA}
\item \texttt{AREA.CLEAR}
\item ‘Release Information’ in ‘Release History’
\end{itemize}
AREA.List

Display a detailed list of all message areas

Opens the **AREA.List** window, displaying all **AREA** window names, i.e. the default name **A000** and all user-defined names. To add user-defined names to the list, use the **AREA.Create** command.

![AREA.List Window](image)

A Yellow and bold indicate the active **AREA** window.

Right-click the name of an **AREA** window to open the **Message Area** popup menu:

- **View** brings a window with this window name to the front.
- **Select** highlights a row in yellow and bold to indicate the active **AREA** window. Information can now be printed to this **AREA** window, e.g. with the commands **PRINT** and **ENTER**. Additionally, the same information can be streamed to a file with the **Stream to file** option.
- **Delete** removes the selected message area and closes the associated **AREA** window. If there are multiple windows for the same message area name, then all window copies will be closed as well. For an example, see **AREA.Delete**.
- **Stream to file** displays the **AREA.OPEN** window, where you can create or browse for a streaming file. You can open a streaming file for each **AREA** window, but streaming is possible to only one file at a time, i.e. to the file of the active **AREA** window.
- **Close stream to file** closes the associated streaming file.

Double-clicking an entry selects and opens this **AREA** window.
Example:

WinExt AREA.List ; overview of existing AREA windows
AREA.Create ephone ; create the AREA window names 'ephone'
AREA.Create testlog ; and 'testlog'
AREA.view testlog ; open the AREA window named 'testlog'
AREA.SELECT testlog ; and select it for screen output
AREA.OPEN testlog \testlog.txt ; additionally stream the screen output
to the file 'testlog.txt'

See also

■ AREA
■ AREA.OPEN
▲ 'Release Information' in 'Release History'
The outputs to the AREA window are saved in a file. The file can be closed with the AREA.CLOSE command.

- **<area_name>**
  - Specify a user-defined <area_name>. Area names are created with the AREA.Create command.
  - If the <area_name> is omitted, then AREA.OPEN refers to the default message AREA window named A000.

- **<filename>**
  - If the file with the specified <filename> already exists, the file will be overwritten by default (same effect as option /Create).

- **<option>**
  - The options are only available if you specify an <area_name>, else the message line displays an error message.

- **Append**
  - Appends the output to an existing file (if the file does not exist, a new file will be created).

- **NoFileCache**
  - Disables the file buffer cache and writes each line to the file immediately. This can be useful to get a complete log file of the AREA window output even if TRACE32 is killed by the operation system.

```
AREA.OPEN A000 protocol.lst ; area will be saved in 'protocol.lst'
DO test
...
AREA.CLOSE A000 ; all messages will be saved

AREA.OPEN A000 ~~~\file.txt /Append
```

**See also**

- AREA
- AREA.CLOSE
- AREA.Create
- AREA.List
- AREA.SAVE
- AREA.Select

▲ 'Message Windows' in 'IDE User's Guide'
▲ 'Release Information' in 'Release History'
AREA.RESet

Format:  

AREA.RESet

All additionally created areas are removed from the area system, and the message AREA A000 is set to the default size (one page). All print outputs and error messages are routed to this AREA window.

AREA.RESet closes all open AREA windows, which have been created with AREA.Create. However, the window displaying the default message area A000 is not closed by AREA.RESet.

See also

■ AREA ■ AREA.CLEAR ■ AREA.CLOSE

▲ ‘Message Windows’ in ‘IDE User’s Guide’
▲ ‘I/O Commands’ in ‘Training Script Language PRACTICE’

AREA.SAVE

Save AREA window contents to file

Format:  

AREA.SAVE [<area_name>] <filename>

Saves the complete and current contents of the specified AREA window to file. Alternatively, right-click in the AREA window, and then select Save from the popup menu.

<area_name> Specify the name of the AREA window you want to save. If <area_name> is omitted, then the contents of the default AREA window A000 are saved.

<filename> Path and file name. Alternatively, use an asterisk if you want to open a dialog-save window. If the file with the specified name already exists, the file will be overwritten.

AREA.SAVE ~~~\areawin.txt ;save the contents of the default AREA window A000

The path prefix ~~~ expands to the temporary directory of TRACE32.

See also

■ AREA ■ AREA.OPEN ■ PRINT

▲ ‘Message Windows’ in ‘IDE User’s Guide’
Selects an output area for the **PRINT** command, when running under PRACTICE. Internal system and error messages are not affected by this command, they are always displayed in the **AREA A000**.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AREA.RESet</td>
<td>; init area system</td>
</tr>
<tr>
<td>AREA.Create XMESSAGE 20. 20.</td>
<td>; create new area named &quot;XMESSAGE&quot;</td>
</tr>
<tr>
<td>AREA.view XMESSAGE</td>
<td>; display window for area &quot;XMESSAGE&quot;</td>
</tr>
<tr>
<td>AREA.Select XMESSAGE</td>
<td>; select area for PRINT and ENTER</td>
</tr>
<tr>
<td>PRINT &quot;Test&quot;</td>
<td>; print string constant</td>
</tr>
<tr>
<td>AREA.Select A000</td>
<td>; select standard area</td>
</tr>
</tbody>
</table>

See also
- **AREA**
- **AREA.CLOSE**
- **AREA.Create**
- **AREA.OPEN**
- **PRINT**

▲ ‘Message Windows’ in ‘IDE User’s Guide’
▲ ‘I/O Commands’ in ‘Training Script Language PRACTICE’
This command displays a message area in an **AREA.view** window. If no argument is used, the default message area **A000** will be displayed in the **AREA.view** window.

```plaintext
; initialize the area system
AREA.RESet

; display the default message area A000 in an AREA window
AREA.view

; create a new, user-defined message area named 'XMESSAGE'
AREA.Create XMESSAGE 20.20.

; display the new message area 'XMESSAGE' in a second AREA window
AREA.view XMESSAGE

; select the message area 'XMESSAGE' for a PRINT operation
AREA.Select XMESSAGE

; print a string constant to the message area 'XMESSAGE' (see AREA win.)
Print "Test"

; select the default message area A000
AREA.Select A000
PRINT "Name of this message area: " AREA.SELECTed()
```

See also

- **AREA**
- **PRINT**
- 'Message Windows' in 'IDE User's Guide'
- 'I/O Commands' in 'Training Script Language PRACTICE'
### AutoSTOre

**Save and restore settings (history, GUI, etc.) automatically**

**Format:**

```
AutoSTOre <filename> [<item> …] [/<option>]
```

**<item>:**

- ALL | HISTory | Win | WinPAGE
- <device_specific_settings>

**<option>:**

- NoDate

Restores settings from the previous TRACE32 session and stores specified settings automatically at the end of a TRACE32 session.

When **AutoSTOre** is executed, the following happens:

- **AutoSTOre** calls the PRACTICE script specified by `<filename>`. The script gets executed as if it was executed by the **DO** command.
- **AutoSTOre** registers the specified items to be stored when the TRACE32 session ends. The settings will be stored to the PRACTICE script specified by `<filename>`.

The **AutoSTOre** command should be used only once per TRACE32 session. Usually it is used within the PRACTICE script that is automatically executed during the start of TRACE32 (e.g. `~/t32.cmm`)

Alternatively, you can save settings manually with the **STOre** command and restore them with the **DO** command. Therefore you might want to use **SETUP.QUITDO** to execute **STOre** on the end of a TRACE32 session.

The **AutoSTOre** command is available also in other systems, like emulators or analyzers, with more system specific options.

<table>
<thead>
<tr>
<th><code>&lt;filename&gt;</code> or ,</th>
<th>User-defined path and file name. If a comma is used instead, TRACE32 saves the file in the temporary directory of TRACE32. See Example. The auto-generated file name consists of the return value of the <strong>OS.ID()</strong> function and the string <code>store.cmm</code>.</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;item&gt;</code>, <code>&lt;option&gt;</code>, and <code>&lt;device_specific_settings&gt;</code></td>
<td>For a detailed description of <code>&lt;item&gt;</code>, <code>&lt;option&gt;</code>, and <code>&lt;device_specific_settings&gt;</code>, refer to the <strong>STOre</strong> command.</td>
</tr>
<tr>
<td>HELP</td>
<td>Store the help settings and the help bookmarks.</td>
</tr>
<tr>
<td>HISTory</td>
<td>Store the command history.</td>
</tr>
<tr>
<td>PBREAK</td>
<td>Store the breakpoints created for PRACTICE scripts (<code>*.cmm</code>).</td>
</tr>
<tr>
<td>Win</td>
<td>Store the entire window configuration (all pages).</td>
</tr>
<tr>
<td>-----</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>WinPAGE</td>
<td>Store the current window page.</td>
</tr>
<tr>
<td>…</td>
<td>All other keywords refer to the commands of the same name.</td>
</tr>
</tbody>
</table>

**Example:** Restore settings saved by **AutoSTOre** in the previous TRACE32 session and register the saving of the following items when TRACE32 gets closed: Command history (**HISTORY**), the address and trace bookmarks (**BOOKMARK**) and the help bookmarks (**HELP**).

```plaintext
AutoSTOre, HISTORY BOOKMARK HELP
```

**See also**
- ClipSTOre
- STOre
- 'Window System' in 'IDE User's Guide'
Bitmap editor for user-defined icons

Format: **BITMAPEDIT** [<filename>]

Allows you to edit bitmaps embedded in the following TRACE32 file types: PRACTICE (*.cmm), menu (*.men), or dialog (*.dlg) files. Bitmaps can be included in three different formats and two variants. The bitmap editor can only be used to modify bitmaps. The placeholder [] for the bitmap must be created beforehand with a text editor in the TRACE32 file. For step-by-step procedures, see “Icons” (ide_user.pdf).

The bitmaps can be placed in one string or into multiple lines. The multiple line format is only suitable for TOOLITEM commands in menu definition files. The string format can be placed in toolbar buttons, dialog buttons, window buttons and menu items. The brackets can contain either a reference to a predefined bitmap (which cannot be edited with the bitmap editor) or the data for a colored bitmap. The bitmap can have three different formats:

- **NATIVE**: In the plain format each character corresponds to one pixel in the bitmap. The character defines the color of the pixel.
- **RLE**: The compressed format adds a simple run-length compression to this format to save space. Both formats (plain and compressed) can also be edited with a regular text editor.
- **SIGNATURE**: The signature format provides the best compression, but the bitmap can only be edited by this bitmap editor.

The bitmaps can be placed in one string or into multiple lines. The multiple line format is only suitable for TOOLITEM commands in menu definition files. The string format can be placed in toolbar buttons, dialog buttons, window buttons and menu items. The brackets can contain either a reference to a predefined bitmap (which cannot be edited with the bitmap editor) or the data for a colored bitmap. The bitmap can have three different formats:

- **NATIVE**: In the plain format each character corresponds to one pixel in the bitmap. The character defines the color of the pixel.
- **RLE**: The compressed format adds a simple run-length compression to this format to save space. Both formats (plain and compressed) can also be edited with a regular text editor.
- **SIGNATURE**: The signature format provides the best compression, but the bitmap can only be edited by this bitmap editor.

```plaintext
MENUITEM "[ ]New Menu" ... ; The square brackets will later contain the bitmap.
MENUTITEM "[ ]Second New Menu"

MENU.ReProgram
{
  ADD TOOLBAR
  {
    TOOLITEM "newbutton" "cmd"
    [ ] ; The square brackets will later contain the bitmap.
  }
}
...
ENDDO

B::BITMAPEDIT addmybutton.cmm ; Opens a window for defining and modifying the bitmap.
```
Changes or displays the current working directory. On MSDOS/Windows environments the drive may be selected too. When used as a command prefix, the directory is changed to the path used in the command line (implicit change).

**Examples:**

```
ChDir \t32 ; change directory

ChDir a: ; change drive

ChDir a:\t32 ; change drive and directory

ChDir.DO c:\sample\x ; change to c:\sample and execute the
                       ; file 'x'

ChDir.DO * ; use the file browser to choose a
             ; new directory
             ; and execute a PRACTICE script there
```

**See also**

- DO
- MKDIR
- PWD
- OS.DIR()
- 'File and Folder Operations’ in 'IDE User’s Guide'
- 'Release Information’ in 'Release History'
Store settings to clipboard

- **ClipSTOre** [\%<format>] [<item> …]

  - `<format>`: sYmbol | NosYmbol
  - `<item>`: HISTory | Win | WinPAGE | …
    <device_specific_settings>

Stores settings to the clipboard. Press **Ctrl+V** to paste the clipboard contents into a file, e.g. a PRACTICE script file (*.cmm). The **ClipSTOre** command is available also in other systems, like emulators or analyzers, with more system specific options.

- `<item>`, `<format>`: For a detailed description of `<item>` and `<format>`, refer to the **STOre** command.

  - **HELP**: Store the help settings and the help bookmarks.
  - **HISTory**: Store the command history.
  - **PBREAK**: Store the breakpoints created for PRACTICE scripts (*.cmm).
  - **Win**: Store the entire window configuration (all pages).
  - **WinPAGE**: Store the current window page.
  - …: All other keywords refer to the commands of the same name.

**Example 1**: Copies the current settings of the **SYStem.state** window to the clipboard.

```
ClipSTOre SYStem
```

To re-use the settings in a script, continue with the steps suggested in “**How to Run Demo Scripts Copied from the PDF Manuals**” in PRACTICE Script Language User's Guide, page 19 (practice_user.pdf).

**Example 2**: Copies the current settings of the **SYStem.state** window and the command history to the clipboard.

```
ClipSTOre SYStem HISTory
```

**See also**
- **AutoSTOre**
- **PEDIT**
- **STOre**

▲ 'Window System' in 'IDE User's Guide'

©1989-2018 Lauterbach GmbH
Controls the position and size of the TRACE32 main window if TRACE32 is configured to work in MWI window mode (Multiple Window Interface). Use the optional \texttt{<colorindex>} parameter to set the toolbar and/or MWI background color to one of the available eight colors that can be assigned to cores and windows for multicore debugging.

In MWI window mode, the TRACE32 windows and dialog boxes float freely outside the TRACE32 main window.

- For more information about the user interface, see \textit{“Graphical User Interface”} (ide_user.pdf).
- For an overview of the eight colors for cores, open the \texttt{SETUP.COLOR} window.

\begin{verbatim}
Format: \texttt{CmdPOS <left> <up> <hsize> <vsize> [\texttt{<item>}} [\texttt{<colormode>}]}

\texttt{<item>}: Normal | Iconic | Maximized

\texttt{<colormode>}: Auto | DEFault | \texttt{<colorindex>}
\end{verbatim}

\texttt{<left>}: x-coordinate as a floating point or integer or percentage value.

\texttt{<up>}: y-coordinate as a floating point or integer or percentage value.

\texttt{<hsize>}: Horizontal main window size in cursor width or percentage (only valid for Normal)

\texttt{<vsize>}: Vertical main window size in cursor height or percentage (only valid for Normal)

\texttt{Normal}: The TRACE32 main window is positioned at the given x- and y-coordinate with the chosen horizontal and vertical size.

\texttt{Iconic}: The TRACE32 main window is minimized and an icon is shown on the taskbar. Position and size values can be set but will have no effect.
### Maximized

The TRACE32 main window is maximized and fills the whole desktop. Position and size values can be set but will have no effect.

### Auto

Automatically select background color for the toolbar and MWI background according to the current \texttt{CORE} variable within the configuration file (config.t32). If \texttt{CORE} is not set, then the default coloring is used.

### DEFault

Set default colors for toolbar and MWI background.

\texttt{<colorindex>}

Integer number between 0 and 7 to select a fixed background color for toolbar and MWI background.

#### Examples:

<table>
<thead>
<tr>
<th>CmdPOS 10. 10. 70. 30. Normal</th>
<th>; Shows the TRACE32 main window; including the work area</th>
</tr>
</thead>
<tbody>
<tr>
<td>CmdPOS , , , 0. , Normal</td>
<td>; Hides the work area but shows the; menubar, toolbar and commandbar</td>
</tr>
<tr>
<td>CmdPOS , , , Iconic</td>
<td>; Minimized TRACE32 to an icon on; the taskbar</td>
</tr>
<tr>
<td>CmdPOS , , , Auto</td>
<td>; CORE dependent toolbar color</td>
</tr>
</tbody>
</table>

#### See also

- FramePOS
- SETUP.COLOR
- CORE.SHOWACTIVE
- ‘Screen Display’ in ‘IDE User’s Guide’
- ‘Commands’ in ‘IDE User’s Guide’
- ‘Software Installation’ in ‘TRACE32 Installation Guide’
ComPare

ComPare

Compare files

Format: ComPare <filename1> <filename2> [Case]

ComPare compares two files on a byte-by-byte level. The ComPare command stops at the first difference. The different bytes are displayed, together with the position counted in bytes, in lines and columns. The result will be found in the FOUND() function. By comparing test results to reference files, complex system tests will become very simple.

Case

- Observe case sensitivity, i.e. upper and lower case characters are not the same.

ComPare mcc.c mcc.bak

The first difference is displayed in the message line and in the AREA window.

.. PRINT "Comparing files..."
OPEN #1 C:\testfiles\test.log /Append
COMPARE &file_name flash.dump
IF FOUND()
  WRITE #1 "the files are different"
ELSE
  WRITE #1 "the files are identical"
// &verifyResult=FOUND()
// WRITE #1 "&verifyResult"
CLOSE #1
..

See also
- FIND
- TYPE
- FOUND()
- TRACK.COLUMN()

- 'File and Folder Operations’ in 'IDE User's Guide'

©1989-2018 Lauterbach GmbH
COPY

Copy files

Format:  

COPY <source> <destination>

Duplicates one file. No query will be made if the destination file already exists.

COPY ~/per68302.t32 per68302.per
COPY text1.txt text1.old
DATE

For architectures that do not have the CLOCK command group, CLOCK is an alias for DATE.

DATE

Display date and time

Format: DATE

Opens a window with the current system time and date. Useful for documentary purposes in screenshots.

![DATE window example]

The date and time values are returned by the functions DATE.DATE() and DATE.TIME().

```
DATE
PRINT DATE.DATE() " " DATE.TIME(); display date and time in DATE window
PRINT DATE.DATE() " " DATE.TIME(); print date and time to message line
```

See also

- CLOCK
- CONVERT.TIMESTAMP
- CONVERT.TIME
- DATE.DATE()
- DATE.TIME()

▲ 'DATE Functions' in 'IDE Functions'
Delete file

Format: \texttt{DEL \textless filename\textgreater}

This command removes one file. Wildcard characters within the file name will open the browser for selecting one file.

\texttt{DEL "c:/t32/test.bak"}

See also
- RM
- 'File and Folder Operations' in 'IDE User's Guide'
The **DIALOG** command group and its dialog elements, such as buttons and edit boxes, are used to create and display custom dialog boxes. They are normally used to increase the flexibility of PRACTICE script files by providing user selectable actions or requesting information from the user, e.g. actual firmware file name for the flash process.

**NOTE:** Examples of dialog definitions reside in the directories:
- ~/demo/practice/dialogs
- ~/demo/analyzer/trigger

For information about dialog syntax, file types, built-in icons, return values, and PRACTICE macros inside dialog definitions, see “**Dialog Programming**” (ide_user.pdf).

For reference information, screenshots, and source code examples of the various dialog elements, see “**Dialog Definition Programming Commands**” in this manual.

**Dialog Definition Programming Commands**

The syntax of a definition file is line oriented. Blanks and empty lines can be inserted to structure the script. Comment lines start with a semicolon.

Commands which define a dialog element can have a label in front of the command. This label can be used to access the value of the dialog element.

The initial position and size of a custom dialog box on the screen can be governed with **WinPOS**.

The position and size of buttons, drop-down lists, etc. on a custom dialog box can be governed with **POS**, **POSX**, and **POSY**.

See also
- DIALOG.AREA
- DIALOG.END
- DIALOG.OK
- DIALOG.Set
- DIALOG.YESNO
- DIALOG.AREA
- DIALOG.DIR
- DIALOG.EXecute
- DIALOG.Program
- DIALOG.SetDIR
- END
- DIALOG.Disable
- DIALOG.File
- DIALOG.ReProgram
- DIALOG.SetFile
- DIALOG.Enable
- DIALOG.MESSAGE
- DIALOG.Select
- DIALOG.view

▲ 'DIALOG Functions' in 'IDE Functions’
▲ 'Release Information’ in 'Release History’
BAR

Defines a progress bar. The length of the progress bar is governed by the `<width>` of POS. The length of the blue indicator is measured in percentage (%) and can be modified using `DIALOG.Set`.

A more complex demo script is included in your TRACE32 installation. To access the script, run this command:

```
B::CD.PEDIT ~/demo/practice/dialogs/dialog_update.cmm
```
Define a decorative border

Defines a box around other items. It has no effect on input in the window. Position and size are governed by POS.

Source code for the above example:

```plaintext
DIALOG.view
{
    HEADER "BOX Demo"

    POS 1. 1. 12. 5.
    BOX "Options:"

    POS 2. 2. 10. 1.
    Option1: CHECKBOX "Option1" ""
    Option2: CHECKBOX "Option2" ""
    Option3: CHECKBOX "Option3" ""

    POS 20. 6. 5.
    DEFBUTTON "OK" "CONTinue"
}
STOP
DIALOG.END
ENDDO
```
BUTTON

Raised button with an icon and text

Format:  

BUTTON "<text>" [<command>]

Defines a raised button that can display an icon and text. The button can execute a command when clicked. If the command string is omitted, the next line must begin with an open bracket to include a PRACTICE script.

Examples

- The position and size of buttons, drop-down lists, etc. on a custom dialog box can be governed with POS, POSX, and POSY. POS is used in the following two examples.

- The third example uses POSY to place the toggle button to the right of the label “Toggle On/Off”.

Example 1:

```
DIALOG.view
  (  
    HEADER "BUTTON Demo 1"

    POS 1. 1. 10.,
    LAB: EDIT "" ""

    POS 12. 1. 10.,
    BUTTON "[:edit]Browse..."
      (  
        DIALOG.SetFile LAB ~/demo/practice/dialogs/*.cmm
      )

    POS 30. 3. 5.,
    DEFBUTTON "OK" "CONTinue"
  )
STOP
DIALOG.END
ENDDO
```
Example 2:

```plaintext
wr.DIALOG.view
{
    HEADER "Button Demo 2"

    ;  x  y  w  height
    POS 23.1.1.3.
    VLINE ""
    ;  height
    POS 25.1.10.1.
    BUTTON "[:edit]Browse..."
    {
        ; your code, see also DIALOG.SetFile
    }
    ;  height
    POS 25.3.10.,
    DEFBUTTON "OK" "CONTinue"
}
STOP
DIALOG.END
ENDDO
```

**NOTE:** Regarding source code and screenshot of example 2:
- **VLINE** has a height of 3. units. To prevent the **BUTTON** from receiving the same height, we need to set the height of **POS** to 1.
- The comma for **POS** and **DEFBUTTON** means that the value of the previous **POS** argument is used, i.e. height=1.

Example 3 shows how to implement a toggle button using the two keywords "ON" and "OFF" in the **DIALOG.Set** command.

```plaintext
DIALOG
{
    HEADER "BUTTON Demo 3"
    POS 1.0.
    TEXT "Toggle On/Off"
    POSX 5.10.1.
    btn: BUTTON "[:colorlime]On,[:colorred]Off"
    {
        LOCAL &tmp
        ENTRY &tmp
        IF "&tmp"=="ON"
            DIALOG.Set btn "OFF"
        ELSE
            DIALOG.Set btn "ON"
        }
    }
STOP
DIALOG.END
ENDDO
```
CHECKBOX "<text>" [<command>]

Defines a check box item. A check box can have two states: ON or OFF. The <command> is executed when the check box state is changed. If the command string is omitted the next line must begin with an open bracket to include a PRACTICE script. The ON or OFF state is passed as parameter to this script. Here, selecting the check box formats 16 as a hex value; clearing the check box formats the hex value as 16 again.

Source code for the above example:

```c
DIALOG.view
{
  HEADER "CHECKBOX demo"
  
  POS 1. 1. 5.
  cbHEX: CHECKBOX "HEX" "GOTO cbStatus"
    
    POS 8. 1. 10.
    VAL: EDIT "16" ""
    POS 29. 3. 5.
    DEFBUTTON "OK" "CONTinue"
}

; Opens the dialog with the checkbox selected
DIALOG.Set cbHEX ; Omit line to start with the checkbox cleared
DIALOG.Disable VAL ; Make the EDIT text box read-only.

; Respond to the status of the checkbox
cbStatus:
  IF DIALOG.BOOLEAN(cbHEX)
    DIALOG.Set VAL FORMAT.HEX(8,16.)
  ELSE
    DIALOG.Set VAL FORMAT.DECIMAL(8,16.)
  STOP
DIALOG.END
ENDDO
```

A more complex demo script is included in your TRACE32 installation. To access the script, run this command:

```bash
B::CD.PSTEP ~/demo/practice/dialogs/dialog_checkbox.cmm
```
Choose Box

Define a choose box item (radio button type). Normally a choose box is an element of a set/group of buttons, from which only one button can be active at any time.

**Format:**

```
<label> CHOOSEBOX "<text>" [<command>]
```

```
<label>: <groupname>.<subname>:
```

The differentiation which choose box item belongs to which group will be done only by the group name independent of the definition order of all choose box items.

The optional command is executed when the choose box is activated. If the command string is omitted, the next line must begin with an open bracket to include a PRACTICE script.

For the source code of this screenshot, refer to the script on the next page.

```
<groupname> A
```

```
<groupname> mdo
```

```
<groupname> flashsize
```

Another demo script is included in your TRACE32 installation. To access the script, run this command:

```
B::CD.PSTEP ~/demo/practice/dialogs/dialog_choosebox.cmm
```
LOCAL &count &mdo_type &flashsize_selection  
&count=""  
&mdo_type=""  
DIALOG.view  
  (  
      HEADER "CHOOSEBOX Demo"  
      POS 1.0.28.  
      LINE "Type selection:"  
      A.C: CHOOSEBOX "Number" ""  
      A.T: CHOOSEBOX "Letter" ""  
      LINE "MDO selection:"  
      mdo.aaaa: CHOOSEBOX "MD04"  
        (  
           &mdo_type="MD04"  
           PRINT "MDO type 04 selected"  
        )  
      mdo.cccc: CHOOSEBOX "MD16"  
        (  
           &mdo_type="MD16"  
           PRINT "MDO type 16 selected"  
        )  
      LINE "Flash size selection:"  
      flashsize.1: CHOOSEBOX "512KB"  
        (&flashsize_selection="512kb""")  
      flashsize.2: CHOOSEBOX "16MB"  
        (&flashsize_selection="16mb""")  
      flashsize.3: CHOOSEBOX "256MB"  
        (&flashsize_selection="256mb""")  
      POS 24.10.5.  
      DEFBUTTON "OK" "CONTinue"  
  )  
STOP  
  AREA  
  AREA.CLEAR  
  ;----------- check result of choosebox group "A" -----------  
  IF DIALOG.BOOLEAN(A.C)  
    &count=1.  
  ELSE IF DIALOG.BOOLEAN(A.T)  
    &count=0.  
  ELSE  
    PRINT ":- no Type selected"  
  ;----------- check result of choosebox group "mdo" -----------  
  IF "&mdo_type==""  
    PRINT ":- no MDO type selected"  
  ELSE  
    PRINT ":- MDO type selected: &mdo_type"  
    IF POWERNEXUS()  
      SYStem.Option.NEXUS &mdo_type  
    )  
  ;----------- check result of choosebox group "flashsize" -----------  
  PRINT ":&flashsize_selection"  
DIALOG.END  
ENDDO
CLOSE

Format:  **CLOSE [<command>]**

Executes a command when the user tries to close the dialog window. If the command string is omitted, the next line must begin with an open bracket to include a PRACTICE script. The dialog window is NOT closed when this command is present. Closing the window with the **DIALOG.END** command is still possible.

```
LOCAL &label

DIALOG.view
  (POS 1. 1. 10.
   LAB:   EDIT """
   POS 1. 3. 5.
   DEFBUTTON "OK" "JUMPTO okclose"
   CLOSE "JUMPTO winclose"
  )
STOP

okclose:
  &label=DIALOG.STRing(LAB)
winclose:
  DIALOG.END
ENDDO
```

COMBOBOX

Format:  **COMBOBOX "<list_items>" [<command>]**

Defines a combobox item. A combobox provides a list of pre-defined items like a **PULLDOWN**, but additionally lets the user enter a value/string which is not pre-defined. Set the current list item using the **DIALOG.Set** command. Use **DIALOG.STRing()** to retrieve the active list item.

```
COMBOBOX
```

The demo script for the above example is included in your TRACE32 installation. To access the script, run this command:

```
B::PSTEP ~/demo/practice/dialogs/dialog_combobox.cmm
```
**DEFBUTTON**  
Define the default button

Format:  
```
DEFBUTTON "<text>" [<command>]
```

Defines a **BUTTON** item which has the input focus when the dialog is opened. Only one element of a dialog can have the default input focus.

**DEFCOMBOBOX**  
Define a default combo box

Format:  
```
DEFCOMBOBOX "<list_items>" [<command>]
```

Defines a **COMBOBOX** control which has the input focus when the dialog is opened. Only one element of a dialog can have the default input focus.

**DEFEDIT**  
Define a default edit control

Format:  
```
DEFEDIT "<initial_text>" [<command>]
```

Defines an **EDIT** control which has the input focus when the dialog is opened. Only one element of a dialog can have the default input focus.

**DEFHOTCOMBOBOX**  
Define a default hot combo box

Format:  
```
DEFHOTCOMBOBOX "<list_items>" [<command>]
```

Defines a **HOTCOMBOBOX** control which has the input focus when the dialog is opened. Only one element of a dialog can have the default input focus.

**DEFHOTEDIT**  
Define a default hot edit control

Format:  
```
DEFHOTEDIT "<initial_text>" [<command>]
```

Defines a **HOTEDIT** control which has the input focus when the dialog is opened. Only one element of a dialog can have the default input focus.
DEFMEDIT

Define a default multiline edit control

Format: \texttt{DEFMEDIT}"\langle\texttt{initial\_text}\rangle\" [\langle\texttt{command}\rangle]

Defines an \texttt{MEDIT} control which has the input focus when the dialog is opened. Only one element of a dialog can have the default input focus.

DLISTBOX

Define a draggable list box

Format: \texttt{DLISTBOX}"\langle\texttt{list\_items}\rangle\" [\langle\texttt{command}\rangle]

Defines a \texttt{LISTBOX} control where the list items can be rearranged by drag and drop.

\texttt{DLISTBOX}

Click and drag a list item.

\texttt{DIALOG.STRING2()} returns the current sequence of list items.

\texttt{DIALOG.STRING()} returns the selected list item.

The demo script for the above dialog is included in your TRACE32 installation. To access the script, run this command:
\texttt{B::PSTEP ~~/demo/practice/dialogs/dialog_dlistbox.cmm}
### DYNAMIC

**Format:**

```
DYNAMIC "<initial_text>"
```

Defines a single-line area that can be dynamically modified using `DIALOG.Set` while the dialog is open.

![Dynamic Demo](image)

```
LOCAL &switch
&switch=0

DIALOG.view
(
    HEADER "DYNAMIC Demo"

    POS 20. 1.
    myIcon: DYNAMIC ":[stop]"

    POS 1. 1.
    DEFBUTTON "Toggle icon"
    
    IF &switch==0
    (
        DIALOG.Set myIcon "[:colorlime]"
        &switch=1
    )
    ELSE
    (
        DIALOG.Set myIcon "[:stop]"
        &switch=0
    )
)
STOP
DIALOG.END
```

An advanced demo script is included in your TRACE32 installation. To access the script, run this command:

```
B:CD.PSTEP ~/demo/practice/event_controlled_program/dialog_dynamic.cmm
```
DYNCOMBOBOX

Define a dynamic combo box

Format:  DYNCOMBOBOX "<list_items>" [<command>]

tbd.

DYNDEFCOMBOBOX

Define a default dynamic combo box

Format:  DYNDEFCOMBOBOX "<list_items>" [<command>]

Defines a DYNCOMBOBOX control which has the input focus when the dialog is opened. Only one element of a dialog can have the default input focus.

DYNDEFHOTCOMBOBOX

Define a dynamic default hot combo box

Format:  DYNDEFHOTCOMBOBOX "<list_items>" [<command>]

Defines a DYNHOTCOMBOBOX control which has the input focus when the dialog is opened. Only one element of a dialog can have the default input focus.

DYNHOTCOMBOBOX

Define a dynamic hot combo box

Format:  DYNHOTCOMBOBOX "<list_items>" [<command>]

tbd.
Defines a single-line text area in bold and large font size. This text area can be dynamically modified using `DIALOG.Set` while the dialog is open. This is useful, for example, if you want to toggle the display of text you want to emphasize.

Source code for the above example:

```lauterbach
DIALOG.view
{
    HEADER "DYNLTEXT Demo"
    POS 9. 1. 22.
    myMsg: DYNLTEXT ""
    POS 21. 3. 9.
    StartBTN: DEFBUTTON "Start"
    (  
        DIALOG.Set myMsg "Test started"
        DIALOG.Disable StartBTN
        DIALOG.Enable StopBTN
    )
    POS 1. , ,
    StopBTN: BUTTON "Stop"
    (  
        DIALOG.Set myMsg "Test stopped"
        DIALOG.Enable StartBTN
        DIALOG.Disable StopBTN
    )
}
STOP
DIALOG.END
ENDDO
```
DYNPULLDOWN

Format: \texttt{DYNPULLDOWN}\ "\textit{<list\_items>}" [\textit{<command>}]}

Defines a pull-down list that can be dynamically modified using \texttt{DIALOG.Set} while the dialog is open.

\textit{<list\_items>}

The different list items of a pull-down list are defined in the first argument, separated by commas. The selected item is passed as a parameter to the script. Retrieve the currently selected list item with the \texttt{DIALOG.STRing()} function.

\textit{<command>}

The command is executed when a list item is selected. If the command string is omitted, the next line must begin with an open bracket to include a \texttt{PRACTICE} script. For an example, see \texttt{PULLDOWN}.

In the example below, the dialog opens with an empty pull-down list. Clicking the first button loads the list items into the pull-down list. Result: “b” appears as the first item in the pull-down list. In addition, the second button is now activated.

Clicking the second button replaces the previous list items with new ones. Result: “7” appears as the first list item.

The list item that is displayed first (here, “b” and then “7”) is in both cases defined by \texttt{DIALOG.Set}, and not by \texttt{DYNPULLDOWN}. To reproduce this example, see source code below.
The `<list_items>` can be controlled by external data sources (e.g. register contents, etc.) and displayed on screen as members of a dynamic pull-down list using `DIALOG.Set`. However, this is only possible if the string to be displayed really is and remains a member of `<list_items>`. If it is not the case (due to misspelling, other name, etc.), a blank space will be displayed instead. There is no error message.
DYNTAXT
Dynamic, single-line text area in regular font size

Format:

```
DYNTAXT "<initial_text>"
```

Defines a dynamic, single-line text area in regular font size. This text area can be dynamically modified using DIALOG.Set while the dialog is open. This is useful, for example, for text that needs to be refreshed while the dialog is open.

Source code for the above example:

```
DIALOG.view
{
    HEADER "DYNTAXT Demo"

    POS 1. 1. 22.
    myMsg: DYNTAXT "Click Start."

    POS 21. 3. 9.
    StopBTN: BUTTON "Stop"

    DIALOG.Set myMsg "Click Start."
    DIALOG.Enable StartBTN
    DIALOG.Disable StopBTN

    POS 1. , ,
    StartBTN: DEFBUTTON "Start"

    DIALOG.Set myMsg "Click Stop."
    DIALOG.Disable StartBTN
    DIALOG.Enable StopBTN

    DIALOG.Disable StopBTN

    STOP
    DIALOG.END
ENDDO
```
Defines an **EDIT** control. The `<command>` is executed only after the text has been modified and the **EDIT** control has been left. If the command string is omitted, the next line must begin with an open bracket to include a PRACTICE script. The string of the **EDIT** control is passed as a parameter to the script.

**Format:**

```
EDIT "<initial_text>" [<command>]
```

A more complex demo script is included in your TRACE32 installation. To access the script, run this command:

```
B::CD.PSTEP ~~/demo/practice/dialogs/dialog_edit.cmm
```

The **HOTEDIT** control executes the `<command>` for each character while you are typing.
Defines the header line of a dialog. You can also customize the icon in the top left corner using `ICON`.

**Example 1**: Dialog with a static header line.

```plaintext
DIALOG.view
  (
    HEADER "HEADER Demo"
    POS 30. 3. 5.
    DEFBUTTON "OK" "CONTinue"
  )
STOP
DIALOG.END
ENDDO
```

**Example 2**: To implement a variable header line, remember to use the ampersand character `&` as shown below.

```plaintext
LOCAL &header_text ;declare local PRACTICE macro
&header_text="HEADER Demo 2" ;assign parameter value to macro
DIALOG.view
  (& ;note that the ampersand (&) character is required here.
    HEADER "&header_text"
    POS 30. 3. 5.
    DEFBUTTON "OK" "CONTinue"
  )
STOP
DIALOG.END
ENDDO
```
Define a help icon

Format: HELP <name>

tbd.
HOTEDIT

Define a hot edit control

Defines an EDIT control. While a normal EDIT control executes \texttt{<command>} when the control loses input focus, HOTEDIT executes \texttt{<command>} whenever the text in the control changes.

In the following example, HOTEDIT is used to validate user input. Without input, the OK button and the icon are grayed out. If the input is valid, the OK button is activated and the icon turns green. If the input is invalid (e.g. a disallowed character), a red icon prompts users to correct their inputs before they can proceed.

Two disallowed characters: blank and ?

```
DIALOG.view
{
    HEADER "HOTEDIT Demo"
    POS 1. 0. 29. 1.
    TEXT "Enter string:"

    myHEDT: HOTEDIT ""
    (;; for each keystroke execute \texttt{<command>}:
      PRIVATE &Input
      &Input=DIALOG.STRing(myHEDT)
      DIALOG.Disable btnOK
      ;; check the input for the following disallowed characters
      IF STRING.FIND("&Input", "/:*?<>|äöü",")==TRUE()
          DIALOG.Set myIcon "[:colorred]"
      ELSE IF "&Input"=="
          DIALOG.Set myIcon "[:colorgrey]"
      ELSE
          ;; enable the OK button if the input is valid
          DIALOG.Set myIcon "[:colorlime]"
          DIALOG.Enable btnOK
      )
    myIcon: DYNAMIC "[:colorgrey]"
    btnOK: DEFBUTTON "OK" "CONTinue"
}
DIALOG.Disable btnOK ;; disable the OK button
STOP ;; wait for the user’s response to the dialog
&retVal=DIALOG.STRing(myHEDT); get the string and then
DIALOG.END ;; close the dialog
DIALOG.OK "Result: &retVal" ;; display the string
```

A more complex demo script is included in your TRACE32 installation. To access the script, run this command: \texttt{B::CD.PSTEP ~/demo/practice/dialogs/dialog_hotedit.cmm}
**HOTCOMBOBOX**

Define a COMBOBOX control. While a normal COMBOBOX control executes `<command>` when the control loses input focus, HOTCOMBOBOX executes `<command>` whenever the control’s text or selection changes.

**Format:**

\[
\text{HOTCOMBOBOX "<list_items>" [<command>]}
\]

**ICON**

New icon in top left corner of dialog

**Format:**

\[
\text{ICON "<built_in_icon_name>" | "<user_defined_icon>"}
\]

Replaces the default icon in the top left corner of a dialog with a different icon. To display icons from the TRACE32 icon library in a dialog, observe the rules shown in [A] and [B]:

A  To show an icon in the header, use ICON.

B  To show icons below the header, use STATIC or DYNAMIC.

Source code for the above example:

```plaintext
DIALOG.view
  (;
    (A) icon in header: omit brackets
    ICON "[:achartnest]"
    HEADER "ICON demo"

    (B) icon below header: include brackets
    POS 27. 1. 2.
    STATIC "[:ddraw]"

    POS 30. 3. 5.
    DEFBUTTON "OK" "CONTinue"
  )
STOP
DIALOG.END
ENDDO
```

For more information about icons, type at the TRACE32 command line: `Help.Index "icons"`
INFOTEXT

Define a multiline info text box on a dialog

INFOTEXT "<msg_text>" [<background>] [<border_style>] [<font>] [<scrollbar>] [<padding>]

<back ground>:
- GRay
- WWhite
- LightGray
- DarkGray
- STicker

<border_style>:
- NoBorder
- Simple
- SUnken
- RAised

<font>:
- Variable1
- Fixed1
- Fixed2
- Fixed3
- Fixed4

<scrollbar>:
- HScroll

<padding>:
- 0...
- 7

Defines a multiline info text box for messages you want to display on a dialog. Unlike DIALOG.AREA, an INFOTEXT can be placed anywhere on the dialog. The display of an INFOTEXT box can be formatted with the options listed above.

The message text is write-protected and cannot be directly edited by users. However, the message text can be dynamically modified using DIALOG.Set while the dialog is open. This is useful, for example, if you want to provide embedded user assistance on a dialog.

Your message text automatically adjusts to the width of the INFOTEXT box.

<table>
<thead>
<tr>
<th>&lt;msg_text&gt;</th>
<th>Max. length 2048 characters.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default settings</td>
<td>If you omit all formatting options, then INFOTEXT is formatted with GRay, NoBorder, Variable1, and 0 by default.</td>
</tr>
<tr>
<td>HScroll</td>
<td>If HScroll is included, the INFOTEXT box displays a horizontal scrollbar, and the automatic word wrap is turned off.</td>
</tr>
<tr>
<td></td>
<td>If HScroll is omitted, the horizontal scrollbar is hidden, and the automatic word wrap is turned on.</td>
</tr>
<tr>
<td></td>
<td>Your message text automatically adjusts to the width of the INFOTEXT box.</td>
</tr>
</tbody>
</table>
All formatting options

An demo script is included in your TRACE32 installation. The script provides an interactive demo of all formatting options. To view the formatting effects, click the radio options in the demo dialog.

To access the demo script, run this command:
B::CD.PSTEP ~/demo/practice/dialogs/dialog_infotext.cmm

Source code example for the above screenshot:

```
LOCAL &addTxt ;declare local macro

&addTxt="<Your information for script users>"+CONV.CHAR(10.)
&addTxt="&addTxt"+"1. ..."+CONV.CHAR(10.) ;adds a line feed
&addTxt="&addTxt"+"2. ..."+CONV.CHAR(10.)
&addTxt="&addTxt"+"3. ..."+CONV.CHAR(10.)
&addTxt="&addTxt"+"4. ...

DIALOG.view
(&
    ; '+' allows you to pass the local macro to a
    ; dialog block that is embedded in a *.cmm file
    HEADER "INFOTEXT Demo"

    ;   x    y     width  height
    POS  0.5  0.25   2.   1.
    STATIC "[:stop]"

    POSX 1.           27.
    LTEXT "Pre-conditions for ... :"

    POSY 0.5          ,       4.25
    myLabel: INFOTEXT "&addTxt" STicker SIMple Variable1 7.
)

STOP
DIALOG.END
```
LINE

Format: \texttt{LINE "<text>"}

Defines an decorative line. It has no effect on input in the window.

Source code for the above example:

```
DIALOG.view
  (  
    HEADER "LINE Demo"

    \texttt{POS 1. 1. 20.  
      LINE "Options:"}

    \texttt{POS 2. 2.25 10. 1.  
      Option1: CHECKBOX "Option 1" ""  
      Option2: CHECKBOX "Option 2" ""}

    \texttt{POS 29. 5. 5.  
      DEFBUTTON "OK" "CONTinue"
    )
  )
STOP
DIALOG.END
ENDDO
```
LISTBOX "<list_items>" [<command>]

Defines a listbox control. The control allows to select one of the items in the list. Set the current selection using the DIALOG.Set command. Retrieve the current selection with the DIALOG.STRING() function.

The demo script for the above example is included in your TRACE32 installation. To access the script, run this command:

B:~CD.PSTEP ~/demo/practice/dialogs/dialog_listbox.cmm
LTEXT

Static, single-line text area in bold and large font size. This is useful, for example, if you want to format text as a heading or alert users to important things.

Source code for the above example:

```lauterbach
DIALOG.view
  (  
    HEADER "LTEXT Demo"
    
    POS 1. 1. 2. 
    STATIC "[:stop]"

    POS 4. 1. 29. 
    LTEXT "Before you proceed:"

    POS 4. 2.25
    TEXT  "1. ..." 
    TEXT  "2. ..."

    POS 29. 5. 5. 
    DEFBUTTON "OK" "CONTinue"
  )
STOP
DIALOG.END
ENDDO
```
**LEDIT**  Define an edit control in bold and large font

Format: `LEDIT "<initial_text> [<command>]`

Defines an edit control in which the user input is formatted in bold and large font. For an illustration of LEDIT and EDIT, see EDIT.

**MEDIT**  Define a multiline edit control

Format: `MEDIT "<initial_text> [<command>]`

Defines a multiline edit control. Compared to the normal EDIT control, MEDIT is capable of holding multiple lines of text. Set the edit text using DIALOG.Set. Retrieve the current text with the DIALOG.STRING() function.

![Example of EDIT and MEDIT controls](image)

DIALOG.Set  MEDIT

DIALOG.STRING()

The demo script for the above example is included in your TRACE32 installation. To access the script, run this command:

```
B::CD.PSTEP ~/demo/practice/dialogs/dialog_edit.cmm
```

**MLISTBOX**  Define a multiline list box

Format: `MLISTBOX "<list_items>" [command]`

Defines a multiline LISTBOX control. The control allows to select one or more items at the same time of the items in the list. Set the selected items using DIALOG.Set. Retrieve the current selection with the DIALOG.STRING() function. The selected items are transferred in a comma-separated string.

A complex demo script is included in your TRACE32 installation. To access the script, run this command:

```
B::CD.PSTEP ~/demo/practice/dialogs/dialog_listbox.cmm
```
Internal dialog name

Format:  

```
NAME "<text>"
```

Defines an internal name for a dialog. The internal name is not displayed on the GUI. Internal names can be used to manipulate dialogs programmatically. For example, you can programmatically check and respond to the status of a dialog (open or close). The dialog name can also be used bring a particular dialog to the front when it is hidden behind a lot of other open dialogs and windows.

In the example below, the **Toggle 2nd dialog** button opens and closes the small dialog based on the return value of the `WINDOW.NAME()` function. If you comment or leave out the line `DIALOG.SELECT myDlg2` then the large dialog is closed.

![Example dialog](image)

Source code for the above example:

```plaintext
DIALOG.view
{
  NAME "myDlg"
  HEADER "NAME Demo"

  POS 1. 4. 29.
  DEFBUTTON "Toggle 2nd dialog"
  {
    IF WINDOW.NAME("myDlg2") == FALSE()
      GOSUB NextDialog
    ELSE
      DIALOG.SELECT myDlg2
      DIALOG.END
    ENDIF
  }
  STOP
DIALOG.END
ENDDO

;------------------------------------
NextDialog:
DIALOG
{  
  NAME "myDlg2"
  HEADER "NAME Demo 2"
  }
STOP
DIALOG.END
ENDDO
```
Define position and size

Format:  

POS <x> <y> <width> <height>

Defines the size and position of the next dialog element in units. Buttons in normal dialog windows have a width of 9. units and a height of 1. unit. Without POS, the vertical position of a dialog element is advanced by 1. unit, and the default size is 9. x 1. units.

NOTE:  

POS has no effect on the size of the dialog or window itself. POS determines only the size and position of the next dialog element, e.g. a BUTTON or an EDIT control.

1 POS unit is not equal to 1 WinPOS unit.

<table>
<thead>
<tr>
<th>&lt;x&gt;</th>
<th>Max. &lt;x&gt; is 16383.5 units.</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;y&gt;</td>
<td>Max. &lt;y&gt; is 8191.75 units.</td>
</tr>
<tr>
<td>&lt;width&gt;</td>
<td>Max. &lt;width&gt; of an element is 16383.5 units.</td>
</tr>
<tr>
<td>&lt;height&gt;</td>
<td>Max. &lt;height&gt; of an element is 8191.75 units.</td>
</tr>
<tr>
<td>,</td>
<td>Value of the previous POS argument is used.</td>
</tr>
<tr>
<td>&lt;no_argument&gt;</td>
<td>Value of the previous POS argument is used, starting from right to left. In this example, the &lt;height&gt; and &lt;width&gt; of the previous POS are used for the unspecified &lt;height&gt; and &lt;width&gt;:</td>
</tr>
</tbody>
</table>

;     <x>  <y>  <width> <height>
POS    3.   7.

The horizontal size and position can be selected in half units: (0.0 - 0.5 - 1.0 - 1.5 - 2.0 - ...). The vertical size and position can be selected in half and quarter units: (0.0 - 0.25 - 0.5 - 0.75 - ...).

Example: The following example is for demo purposes only. It illustrates how POS can be used to determine the positions and sizes of several BUTTON dialog elements in a very large custom dialog.
By executing an optional WinPOS command before the dialog block, you can limit the initial size of very large custom dialogs; scrollbars are added automatically.

x and y, height and width are POS units.

The initial width of this dialog is 67. WinPOS units.

The initial height of this dialog is 8. WinPOS units.

To try this script, simply copy it to a test.cmm file, and then run it in TRACE32 (See “How to…”).

```cmm
WinPOS , , 67. 8. ;limit the initial size of this large custom dialog
DIALOG.view
{
    HEADER "POS Demo"
    ;No POS command => default width is 9. and default height is 1. unit
    BUTTON "[:t32]" "PRINT "This is a demo." "
    ;No POS command => next element is advanced by 1. unit on the y-axis
    BUTTON "[:config]" ""
    ;     <x>     <y>       <btn_width>   <btn_height>
    POS 25.      3.           ,             2.             
    BUTTON "[:colors]" ""
    ;     <x>     <y>       <btn_width>   <no_argument>
    POS 500.    100.          10.          
    BUTTON "OK" "CONTinue"
}
```
**POSX**

**Define position and size on the x-axis**

Format: 

```
POSX <inc> <width> <height>
```

Defines the position and size (width and height) of one dialog element or a block of dialog elements on the x-axis relative to the absolute position of the previous POS command. For parameter descriptions, see POS.

```
DIALOG.view
 {
   HEADER "POSX and POSY Demo"
   ; <x> <y> <w> <h>
   POS 4. 1. 24. 1.
   myBox1: EDIT "A1" ""
   myBox2: EDIT "A2" ""
   POSY 0.5 , ,
   myBox3: EDIT "A3" ""
   POSX 3. 4. 1.
   POSY -2.5
   myBox4: EDIT "B1" ""
   ; <no_arguments>
   POSY 1.5
   myBox5: EDIT "B3" ""
   POSX 1. 6. 1.
   SPACE
   DEFINETEXT "OK" "CONTinue"
 }  
STOP
DIALOG.MESSAGE DIALOG.STRING(myBox1) ;get value of EDIT box by label
DIALOG.END
```

By modifying just the two <x> and <y> values of POS in the source code below, you can move the entire block up/down, left/right.

---

**POSY**

**Define position and size on the y-axis**

Format: 

```
POSY <inc> <width> <height>
```

Defines the position and size (width and height) of one dialog element or a block of dialog elements on the y-axis relative to the absolute position of the previous POS command. For parameter descriptions, see POS.
PULLDOWN

Defines a static pull-down list.

Format: PULLDOWN "<list_items>" [<command>]

A pull-down list can have different list items. The list items are defined in the first argument, separated by commas. The selected item is passed as parameter to the PRACTICE script. You can retrieve the currently selected item with the DIALOG.STRing() function.

The command is executed when a list item is selected. If the command string is omitted the next line must begin with an open bracket to include a PRACTICE script.

Source code for the above example:

```c
DIALOG.view
{
   HEADER "PULLDOWN Demo"

   POS 1. 1. 12.
   BASE:   EDIT "" ""

   POS 14. 1. 5.
   UNIT:   PULLDOWN "ks,ms,us"
   {
      IF DIALOG.STRing(UNIT)=="ks"
         DIALOG.Set BASE "1000. s"
      IF DIALOG.STRing(UNIT)=="ms"
         DIALOG.Set BASE "1/1000. s"
      IF DIALOG.STRing(UNIT)=="us"
         DIALOG.Set BASE "1/1000000. s"
   }

   POS 30. 3. 5.
   DBUTTON "OK" "CONTinue"
}
STOP
DIALOG.END
ENDDO
```

A more complex demo script is included in your TRACE32 installation. To access the script, run this command:

```
B::CD.PSTEP ~/demo/practice/dialogs/dialog_pulldown.cmm
```

©1989-2018 Lauterbach GmbH
**SPACE**  
 Applies the `<height>` of the previous POS, POSX, or POSY command to the next dialog element.

**STATIC**  
 Place an icon in a dialog

Format:  

```
STATIC "<built_in_icon_name>" | "<user_defined_icon>"
```

Defines a static, single-line area. **STATIC** is typically used to place an icon in a dialog. See also **ICON**.

It is recommended that you use **TEXT** if you want to display text next to the icon. Assigning icon and text directly to **STATIC** is possible, too. But this approach makes it difficult to position the element.

Source code for the above example:

```c
DIALOG.view
(   HEADER "STATIC Demo"
 ; x y width height
  POS 2.0 0.5 2. ,
  STATIC "[:aprochart]"
  STATIC "[:aprofile]"
  STATIC "[:pperf]"

  STATIC "[:profile]"
  POS 6 , 6.
  TEXT "Profile"

  POS 30.4 5.
  DEFBUTTON "OK" "CONTinue"
)
STOP
DIALOG.END
ENDDO
```

For more information about icons, type at the TRACE32 command line: **Help.Index "icons"**
Static, single-line text area in regular font size

Format:  \texttt{TEXT "<text>"}

Defines a static, single-line text area in regular font size. \texttt{TEXT} can be used to display a user-defined name for a control, here for an \texttt{EDIT} text box.

In addition, you can use \texttt{TEXT} to specify the initial width for any dialog. Simply combine \texttt{TEXT} and \texttt{POS} to create an empty line, see source code below.

Source code for the above example:

```
DIALOG.view
{
  HEADER "TEXT Demo"
  ; define width of dialog by printing an empty text: width is 29. units
  ;  x  y  w   h
  POS 0. 0. 29. 1.
  TEXT ""

  POS 1. 1.
  TEXT "any string:"

  POS 1. 2. 10.
  myLabel: EDIT "" ""

  POS 1. 4. 5.
  DEFBUTTON "OK" "CONTinue"
}
STOP
DIALOG.END
ENDDO
```

Alternatively, you can set the initial dialog width by moving, for example, the \texttt{OK} button to the right of the dialog as far as required. For information about the maximum values of width and height, see \texttt{POS}.

```
POS 30. 3. 5.
DEFBUTTON "OK" "CONTinue"
```
TEXTBUTTON

Format:  TEXTBUTTON "<text>" [<command>]

Defines a flat button with text only. The result is comparable to a clickable area where the borders are not visible. The button can execute a command when clicked. If the command string is omitted, the next line must begin with an open bracket to include a PRACTICE script.

Source code for the above example:

```c
DIALOG.view
{
    HEADER "TEXTBUTTON Demo"

    POS 1. 1. 10.
    myLabel: EDIT "" ""

    POS 12. 1. 10.
    TEXTBUTTON "Browse..."
    (  
        DIALOG.SetFile myLabel  ~~/demo/practice/dialogs/*.cmm
    )

    POS 30. 3. 5.
    DEFBUTTON "OK" "CONTinue"
}

STOP
DIALOG.END
ENDDO
```
Implements a +/- toggle button on a dialog. Clicking the button toggles between a [+] icon and a [-] icon. The +/- toggle button can execute a command when clicked. If the command string is omitted, the next line must begin with an open bracket to include a PRACTICE script. You can increase the clickable area, by using TREEBUTTON together with TEXTBUTTON.

In the example below, clicking the +/- toggle button expands and collapses the lower part of a dialog: This dialog part could, for example, be used for (a) making advanced options available or (b) a brief description of a script or (c) quick access to the source code of the script or (d) quick access to the location of the script.

Source code for the above example:

```plaintext
LOCAL &expand
&expand=0

DIALOG.view
{
    NAME "myDemoDlg"
    HEADER "TREEBUTTON demo"
    POS 1. 1. 1.

treeBTN: TREEBUTTON ""
    (    
        IF &expand==0
        (        
            DIALOG.Set treeBTN "ON"
            &expand=1
            WinRESIZE 35. 3. myDemoDlg
        )
        ELSE
        (        
            DIALOG.Set treeBTN "OFF"
            &expand=0
            WinRESIZE 35. 2. myDemoDlg
        )
    )
    POS 1. 2. 25.
    DYNTXT "CPU Family License: "+LICENSE.FAMILY(LICENSE.getINDEX())
}
WinRESIZE 35. 2. myDemoDlg ;Initial dialog size, collapsed
STOP
DIALOG.END
ENDDO
```
UPDATE

Executes commands periodically. The default update interval is one second. The \texttt{\textless update\_interval\textgreater} cannot be interrupted. It is recommended that you comment out the \texttt{UPDATE} line before debugging such a PRACTICE script.

In this example, the \texttt{DIALOG.Set} command is parametrized with the \texttt{DATE.Time()} function to implement a timer on a dialog.

Source code for the above example:

```c
DIALOG.view
{
    NAME "myDlg"
    HEADER "UPDATE Demo"

    ; Defines the position of the next GUI control.
    ; x  y  w   h
    POS 0. 0. 29.

    ; This GUI control is a text box that can be updated dynamically,
    ; i.e. while the dialog is open.
    ; Display the current time in this text box.
    ; Assign the label myTimer to the dynamic text box.
    myTimer: DYNTEXT Clock.Time()

    ; Loop to update the text box labeled myTimer.
    ; The text box is updated as long as the dialog is open.
    UPDATE "DIALOG.Set myTimer Clock.Time()" 1.0s
}
STOP
DIALOG.END
ENDDO
```

A more complex demo script is included in your TRACE32 installation. To access the script, run this command:

```bash
B::CD.PEDIT ~/demo/practice/dialogs/dialog_update.cmm
```

Remember that the \texttt{\textless update\_interval\textgreater} of \texttt{UPDATE} cannot be interrupted.

An alternative to \texttt{UPDATE} is \texttt{ON TIME}. To access the demo script, run this command:

```bash
B::CD.PSTEP ~/demo/practice/event\_controlled\_program/dialog_onetime.cmm
```

\texttt{ON TIME} can be interrupted.
Define a decorative vertical line. It has no effect on input in the window.

Source code for the above example:

```bash
DIALOG.view
(  
  HEADER "VLINe Demo"

  POS 1. 1. 10. 1.
  TEXT "any string 1:"
  myLabelA:  EDIT "" ""

  ; x y w height
  POS 12. 1. , 3.
  VLINE ""

  POS 14.25 1. 10. 1.
  TEXT "any string 2:"
  myLabelB:  EDIT "" ""

  POS 30. 4. 5. ,
  DEFBUTTON "OK" "CONTinue"
)
STOP
DIALOG.END
ENDDO
```
DIALOG.AREA

Adds an output area to a custom dialog

Format: DIALOG.AREA [<area_name> [<file>]]

Adds a named output AREA at the bottom of custom dialogs.

- **Example 1**: The named AREA is created by a *.cmm file. The DIALOG.AREA command in the *.cmm file calls the *.dlg file containing the actual dialog definition.

- **Example 2**: The entire DIALOG.AREA block is embedded in the same *.cmm file, where the named AREA is created.

---

Example 1 - Source code for the above screenshots (*.cmm and *.dlg file)

```cmm
;*.cmm file:
copy and paste this block in a *.cmm file which calls the *.dlg file:
AREA.Create myMsg ;create a named area that is invisible
AREA.Select myMsg ;select this area for output
DIALOG.AREA myMsg ~/mytest.dlg ;call the *.dlg file
```

```cmm
;*.dlg file:
copy and paste this block in the *.dlg file called by the *.cmm file:
HEADER "DIALOG.AREA Demo"

StartBTN: DEFBUTTON "Start"
{
    PRINT "Started at: "+CLOCK.TIME()
    ;...<your_code>
}
;move button 6 units on the x axis
POSX 6.

StopBTN: BUTTON "Stop"
{
    PRINT " Stopped at: "+CLOCK.TIME()
    ;...<your_code>
}

CLOSE
( ;select default AREA A000 for output again and close the dialog
AREA.Select A000
DIALOG.END
)
```
Example 2 - a single *.cmm file: A complex demo script is included in your TRACE32 installation. To access the script, run this command:

B::CD.PSTEP ~/.demo/practice/dialogs/dialog_area.cmm

See also

- DIALOG
- DIALOG.view

▲ ‘Dialog Programming’ in ‘IDE User's Guide’

**DIALOG.DIR**

Display a folder picker dialog

Format: `DIALOG.DIR <directory_name>`

Creates a dialog box to choose a directory name. The directory name must contain wildcard characters. The directory must exist. The result of the selection is returned like the result value of a subroutine.

```
LOCAL &directoryname
DIALOG.DIR *
ENTRY %LINE &directoryname ;%LINE is recommended since the
;return value may contain spaces
;<your_code>
```

In case of spaces in the selected directory name or its path ENTRY %LINE &directoryname needs to be used.

See also

- DIALOG
- DIALOG.SetDIR
- DIALOG.view

▲ ‘Dialog Programming’ in ‘IDE User's Guide’
DIALOG.Disable

Disables dialog elements. Disabled elements are shaded out and cannot be executed.

Format:  

DIALOG.Disable <label>

<label>  User-defined label identifying a dialog element.

Source code for the above example:

```
DIALOG.view
{
    HEADER "Enable/Disable Demo"
    
    POS 1. 1. 22.
    myMsg:    DYNTEXT "Click Start."
    
    POS 21. 3. 9.
    StopBTN: BUTTON "Stop"
    {
        DIALOG.Set myMsg "Click Start."
        DIALOG.Enable StartBTN
        DIALOG.Disable StopBTN
    }
    
    POS 1. , ,
    StartBTN: DEFBUTTON "Start"
    {
        DIALOG.Set myMsg "Click Stop."
        DIALOG.Disable StartBTN
        DIALOG.Enable StopBTN
    }
}
DIALOG.Disable StopBTN
STOP
DIALOG.END
ENDDO
```

See also

- DIALOG
- DIALOG.Enable
- DIALOG.Set
- DIALOG.view
- 'Dialog Programming' in 'IDE User's Guide'
DIALOG.Enable

Enable dialog elements

Format:  DIALOG.Enable  <label>

Enables dialog elements. Disabled elements are shaded out and cannot be executed. For an example with screenshot and source code, see **DIALOG.Disable**.

See also
- DIALOG
- DIALOG.Disable
- DIALOG.Set
- DIALOG.view
- 'Dialog Programming' in 'IDE User's Guide'

DIALOG.END

Close the dialog window

Format:  DIALOG.END

Closes the currently active dialog window.

See also
- DIALOG
- DIALOG.view
- 'Dialog Programming' in 'IDE User's Guide'

DIALOG.EXecute

Execute a dialog button

Format:  DIALOG.EXecute  <label>

Executes the command of a button. This can be useful when the commands one button should be included in the sequence executed by another button.

See also
- DIALOG
- DIALOG.view
- 'Dialog Programming' in 'IDE User's Guide'
Using the **DIALOG.File** command group, you can incorporate three different types of OS file dialogs in your PRACTICE scripts (*.cmm). This allows users of your script to pick a file via a dialog.

The execution of a script stops when a file dialog is called and waits for the user input. After users have opened, saved, or selected the file they want, the file name is passed to the PRACTICE script and script execution continues right away.

The table below provides an overview of the differences between the three dialog types.

<table>
<thead>
<tr>
<th>Dialog Type:</th>
<th>File open</th>
<th>File save</th>
<th>File select</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DIALOG.File.open</td>
<td>DIALOG.File.SAVE</td>
<td>DIALOG.File.SELECT</td>
</tr>
<tr>
<td>Default button</td>
<td>Open</td>
<td>Save</td>
<td>OK</td>
</tr>
<tr>
<td>Existing file was chosen</td>
<td>accept</td>
<td>ask user if file should be replaced</td>
<td>accept</td>
</tr>
<tr>
<td>Non-existing file was chosen</td>
<td>reject (file must exist)</td>
<td>accept</td>
<td>accept</td>
</tr>
<tr>
<td>Command Examples</td>
<td>DIALOG.File.open Data.Load.Elf *</td>
<td>DIALOG.File.SAVE STOre * WIN</td>
<td>DIALOG.File.SELECT Trace.SAVE ➤ Browse...</td>
</tr>
</tbody>
</table>

**NOTE:** If you want the user input to be passed to your own custom dialogs, then use the commands of the **DIALOG.SetFile** command group.

**See also**
- DIALOG.File.open
- DIALOG.File.SAVE
- DIALOG.File.SELECT
- DIALOG.SetFile
- DIALOG.view

▲ 'Dialog Programming' in 'IDE User's Guide'
DIALOG.File.open

Display an OS file-open dialog

| Format: | DIALOG.File.open <filename> |

Creates a dialog box for choosing a file name. The file name usually contains a wildcard character. The file selection is returned like the return value of a subroutine.

- Assumes read access to the file.
- The file chosen by the user always exists. (The file-open dialog will refuse to close if the user enters the name of a non-existing file.)

To try this script, copy it to a test.cmm file, and then run it in TRACE32 (See “How to…”).

```c
PRIVATE &filename &string

WinPOS ,,,,,,,, "Open my text file" ;window title of file-open dialog
DIALOG.File.open "*.txt"
ENTRY %LINE &filename ;%LINE is recommended since the ;return value may contain spaces
IF "&filename"!="" ;if the user has not clicked Cancel
(
  OPEN  #1 "&filename" /Read
  READ  #1 %LINE &string
  CLOSE #1
  PRINT "The files first line says: &string"
)
```

In case of spaces in the selected file name or its path ENTRY %LINE &filename needs to be used.

**NOTE:** For TRACE32 PowerView older than 2016/03 just write DIALOG.File instead of DIALOG.File.open. For those older versions you must use a wildcard in the file name.

**See also**

- DIALOG.File
- DIALOG.SetFile.open
### DIALOG.File.SAVE

Display an OS file-save dialog

<table>
<thead>
<tr>
<th>Format:</th>
<th>DIALOG.File.SAVE &lt;filename&gt;</th>
</tr>
</thead>
</table>

Creates a dialog box for choosing a file name. The file name usually contains a wildcard character. The file selection is returned like the return value of a subroutine.

- Assumes write access to the file.
- The file chosen by the user does not need to exist.
- The dialog box will show a warning if the user selects an existing file.

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

```plaintext
PRIVATE &filename

WinPOS ,, ,, ,, ,, ,, "Save my text file" ; window title of file-save dialog
DIALOG.File.SAVE "~~~/*.txt"
ENTRY %LINE &filename ;%LINE is recommended because the ; return value may contain spaces

IF "&filename"!="" ; if the user has not clicked Cancel
(
  OPEN  #1 "&filename" /Create
  WRITE #1 "Hello World"
  CLOSE #1
)
```

In case of spaces in the selected file name or its path `ENTRY %LINE &filename` needs to be used.

**NOTE:**

For PowerView older than 2016/03 use `DIALOG.FileW` instead of `DIALOG.File.SAVE`. For those older versions you must use a wildcard in the filename.

**See also**

- DIALOG.File
- DIALOG.SetFile.SAVE

©1989-2018 Lauterbach GmbH
DIALOG.File.SELECT  

Display an OS file-select dialog  

Format:  

```
DIALOG.File.SELECT <filename>
```

Creates a dialog box for choosing a file name. The file name usually contains a wildcard character. The file selection is returned like the return value of a subroutine.

- Assumes proper access rights to the file.
- The file chosen by the user does not need to exist.
- Use DIALOG.File.SELECT if you do not intend to open the file or write to it immediately.

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

```
PRIVATE &filename

WinPOS ,,,,,,, "Check Read Permission" ;window title of file select
DIALOG.File.SELECT "*.elf" ;dialog
ENTRY %LINE &filename ;%LINE is recommended since the
;return value may contain spaces
IF OS.FILE.ACCESS(&filename,"cw")
    PRINT "You may open '&filename'"
ELSE
    PRINT %ERROR "Sorry, you may not open '&filename'"
```

In case of spaces in the selected filename or its path, ENTRY %LINE &file name needs to be used.

See also

- DIALOG.File
- DIALOG.SetFile.SELECT
DIALOG.MESSAGE  Create dialog box with an information icon

Format:  

DIALOG.MESSAGE <message>

Creates a standard dialog box with an information icon and an OK button.

![Dialog box with information icon and OK button](image)

; your code here

l_error:
    DIALOG.MESSAGE "FLASH programming error occurred"
l_end:
ENDDO

For information about line breaks and the line continuation character, see DIALOG.OK.

See also

- DIALOG
- DIALOG.OK
- DIALOG.view
- DIALOG.YESNO
- FORMAT.Decimal()

▲ 'Dialog Programming' in IDE User's Guide
▲ 'Release Information' in Release History
Create dialog box with an exclamation mark icon and an **OK** button.

To create a line break in the message of a dialog box, use for example:

```
DIALOG.OK "Hello"+CONV.CHAR(0x0D)+"World!"
```

A backslash \ is used as a line continuation character. It allows you to continue with the message text in the next line of the script file. Only the first line may be indented, the other lines must start in the first column.

```
DIALOG.OK "Please switch ON the TRACE32 debugger first"+CONV.CHAR(0x0d)+\  "and then switch ON the target board."

DIALOG.OK "Please switch ON the hardware \  in this sequence:" \  "1. Switch ON the TRACE32 debugger." \  "2. Switch ON the target board."
```

As the above example shows, the line continuation character \ and the empty-space line break can be combined, too.

**See also**
- **DIALOG**
- **DIALOG.MESSAGE**
- **DIALOG.view**
- **DIALOG.YESNO**
- 'Dialog Programming' in 'IDE User's Guide'
- 'I/O Commands' in 'Training Script Language PRACTICE'
This command offers an editor with on-line syntax check to create a dialog definition file. The input is guided by softkeys. The syntax for the dialog definition file is described below. The command is only used to create new dialogs. The programmed dialog can be viewed with the DIALOG.view command. The default extension for `<filename>` is `.dlg`.

**SPACE** `<line>` Define space

**STATIC** `<line>` Defines a static, single-line area. STATIC is typically used to place an icon on a dialog.

**TEXT** Defines a text field. A text field is no input element.

**VLINE** Define a decorative vertical line. It has no effect on input in the window.

See also

- DIALOG
- DIALOG.ReProgram
- DIALOG.view

▲ 'Dialog Programming' in 'IDE User's Guide'
▲ 'Release Information' in 'Release History'
**DIALOG.ReProgram**

Dialog programming

Format:  

```plaintext
DIALOG.ReProgram [<filename>]
```

Without parameter the default file name in the actual working directory is used (t32.dlg). Without parameter in a PRACTICE script, the definition is embedded in the block following the command. With parameter the corresponding file is compiled. The file should not have any errors, when using this command.

```plaintext
DIALOG.ReProgram mydialog.dlg   ; opens dialog window
;<your_code>...
```

tbd.

See also

- DIALOG
- DIALOG.Program
- DIALOG.view

▲ 'Dialog Programming' in 'IDE User's Guide'
▲ 'Release Information' in 'Release History'

**DIALOG.Select**

Programmatically focus on this dialog

Format:  

```plaintext
DIALOG.Select [<name>]
```

Places the programmatic focus on the named dialog. For an example, see NAME. To bring the dialog to the front from a user's point of view, use WinTOP.

See also

- DIALOG
- DIALOG.view

▲ 'Dialog Programming' in 'IDE User's Guide'
DIALOG.Set  Modify the value of a dialog element

<table>
<thead>
<tr>
<th>Format:</th>
<th>DIALOG.Set &lt;label&gt; &lt;value&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;label&gt;</td>
<td>User-defined label identifying a dialog element.</td>
</tr>
<tr>
<td>&lt;value&gt;</td>
<td>The value you want to dynamically assign to the dialog element.</td>
</tr>
</tbody>
</table>

Type:
- **Boolean**, e.g. TRUE(), FALSE(), `<logical_expressions>`
- **String**, e.g. "Lauterbach GmbH", function return values, or empty string "".

**Example 1**: Here, selecting the check box formats 16 as a hex value; clearing the check box formats the hex value as 16 again.

To run, simply copy and paste the entire example into the TRACE32 command line:

```plaintext
DIALOG
  (  
    HEADER "DIALOG.Set demo"
    
    POS 1. 1. 5.
    HEX: CHECKBOX "HEX"
    (  
      IF DIALOG.BOOLEAN(HEX)
        
        DIALOG.Set VAL FORMAT.HEX(8,16.)
      ELSE
        
        DIALOG.Set VAL FORMAT.DECIMAL(8,16.)
      
    )  
    
    POS 8. 1. 10.
    VAL: EDIT "16" ""
    
    POS 29. 3. 5.
    DEFBUTTON "OK" "CONTinue"
  )  
DIALOG.Disable VAL
STOP
DIALOG.END
ENDDO
```
Example 2 shows how you can set and toggle the state of CHECKBOX and CHOOSEBOX using DIALOG.Set:

```plaintext
DIALOG.view
   (HEADER "DIALOG.Set demo"
      POS 0.5 0.5 27.
      CHECK: CHECKBOX "Checkbox Example"
      CHOOSE.1: CHOOSEBOX "First Choosebox"
      CHOOSE.2: CHOOSEBOX "Second Choosebox"
   )

; e.g. assign a state to a boolean element, e.g. a CHECKBOX
DIALOG.Set CHECK TRUE(); or FALSE()
DIALOG.Set CHECK "ON"; or "OFF"

; e.g. toggle the state of a boolean element, e.g. a CHECKBOX
DIALOG.Set CHECK

; e.g. using the result value of a boolean expression
DIALOG.Set CHECK VERSION.BUILD()>75234.

; e.g. select a CHOOSEBOX
DIALOG.SET CHOOSE.2 ; now "Second Choosebox" is selected
DIALOG.SET CHOOSE.1 ; now "First Choosebox" is selected and
                     ; "Second Choosebox" is de-selected

Example 3: To run, simply copy and paste the PRACTICE script example into the TRACE32 command line.

```plaintext
DIALOG.view
   (HEADER "DIALOG.Set demo"
      POS 0.5 0.5 27.
      myVAL: EDIT "Example String"
      btnA: BUTTON "Modify A" "GOTO StringA"
      btnB: BUTTON "Modify B" "GOTO StringB"
   )
STOP

StringA: ; e.g. assign a string
DIALOG.Set myVAL "New Example String"
STOP

StringB: ; e.g. using the result value of a boolean expression
DIALOG.Set myVAL "TRACE32 Build "+FORMAT.DECIMAL(0.,VERSION.BUILD())
STOP
ENDDO
```

See also
- DIALOG
- DIALOG.Disable
- DIALOG.Enable
- DIALOG.view

▲ 'Dialog Programming' in 'IDE User's Guide'
DIALOG.SetDIR <label> <folder_path>

Sets a <folder_path> to the EDIT box. The <label> is the same user-defined label that is assigned to the EDIT box.

If the path contains wildcard characters, e.g. an asterisk *, a Browse to Folder dialog opens where you can browse for the folder you want.

Example:

DIALOG.view
   (  
       POS 1. 1. 20.
       myLAB: EDIT "" ""
       
       POS 11. 2. 10.
       BUTTON "[:coloryellow]Folder"
       (  
           DIALOG.SetDIR myLAB ~/demo/*
       )
       
       DEFBUTTON "OK" "CONTinue"
   )
STOP

&retVal=DIALOG.String(myLAB) ;get the string from the EDIT box
DIALOG.END                   ;and then close the dialog
IF "&retVal"!=""             ;if the user has selected a directory or
   DIALOG.OK "&retVal"       ;entered a path in the EDIT box
ENDDO
The **DIALOG.SetFile** commands are used to pick a file via an OS file dialog. The file name is then assigned to an **EDIT** or **DEFEDIT** control of a custom dialog opened with the **DIALOG.view** command.

The execution of your script-based workflow stops when an OS file dialog is opened. After users have opened, saved, or selected the file they want, the commands after the **DIALOG.SetFile** command are executed (if there are any).

However, **DIALOG.SetFile** is usually used within the command (or command block) executed when clicking a **BUTTON** of a custom dialog. In this case, there are usually no commands to execute after **DIALOG.SetFile**.

The table below provides an overview of the differences between the three OS dialog types.

<table>
<thead>
<tr>
<th>Dialog Type:</th>
<th>File open</th>
<th>File save</th>
<th>File select</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DIALOG.SetFile.open</strong></td>
<td>Open</td>
<td>Save</td>
<td>OK</td>
</tr>
<tr>
<td><strong>DIALOG.SetFile.SELECT</strong></td>
<td>accept if file should be replaced</td>
<td>ask user</td>
<td>accept</td>
</tr>
<tr>
<td><strong>DIALOG.SetFile.SAVE</strong></td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
</tr>
</tbody>
</table>

### See also
- **DIALOG.SetFile.open**
- **DIALOG.SetFile.SAVE**
- **DIALOG.SetFile.SELECT**
- **DIALOG.SetFile.DIR**
- **DIALOG.File**
- **DIALOG.view**

▲ 'Dialog Programming’ in 'IDE User’s Guide’

**DIALOG.SetFile.open**

OS file-open dialog > file name > EDIT element

Format: **DIALOG.SetFile.open** `<label> <filename>`

Creates a dialog box for choosing a file name and assigns that file name to an **EDIT** dialog element that has the specified `<label>`. The file name usually contains a wildcard character.

- Assumes read access to the file.
- The file chosen by the user always exists. (The file-open dialog will refuse to close if the user selects a non-existing file.)
Example:

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

```
DIALOG.view
(  
  POS 1. 1. 40.
myLAB: EDIT "" ""

  POS 20. 2. 10.
  BUTTON "[::coloryellow]File..."
  (  ;window title of file-open dialog
    WinPOS , , , , , , , , "Open File"
    ;display the file-open dialog, set file type filter to *.cmm
    DIALOG.SetFile.open myLAB "~:/demo/*.*"  
  )

  POSX 1.
  DEFBUTTON "[::edit]Edit"
  (  
    PRIVATE &file
    &file=DIALOG.STRing(myLAB)
    ;clicking Edit opens the file in the TRACE32 built-in editor
    PEDIT "&file"
  )
)
ENDDO
```

See also
- DIALOG.SetFile
- DIALOG.File.open

©1989-2018 Lauterbach GmbH
DIALOG.SetFile.SAVE

OS file-save dialog > file name > EDIT element

Format:

DIALOG.SetFile.SAVE <label> <filename>

DIALOG.SetFileW <label> <filename> (deprecated)

Creates an OS file-save dialog for choosing a file name and assigns that file name to an EDIT dialog element that has the specified <label>. The file name usually contains a wildcard character.

- Assumes write access to the file.
- The file chosen by the user does not need to exist.
- The dialog box will show a warning if the user selects an existing file.

For an example, see DIALOG.SetFile.open.

See also

- DIALOG.SetFile
- DIALOG.File.SAVE

DIALOG.SetFile.SELECT

OS file-select dialog > file name > EDIT element

Format:

DIALOG.SetFile.SELECT <label> <filename>

Opens an OS file-select dialog for choosing a file name and assigns that file name to an EDIT dialog element that has the specified <label>. The file name usually contains a wildcard character.

- Assumes proper access rights to the file.
- The file chosen by the user does not need to exist.
- Use DIALOG.File.SELECT if you do not intend to open the file or write to it immediately.

For an example, see DIALOG.SetFile.open.

See also

- DIALOG.SetFile
- DIALOG.File.SELECT
DIALOG.view [filename]

Compiles and shows a dialog window. Without parameters the dialog definition follows the command in round brackets.

The default extension for filename is .dlg. If no file name is given, DIALOG.view refers to the file t32.dlg in the current directory.

;content of *.cmm
DIALOG.view getfile.dlg

;content of getfile.dlg

    POS 1. 1. 10.
    LAB:     EDIT """
    POS 11. 1. 5.
    BUTTON "File"
    {
        DIALOG.SetFile LAB *.cmm
    }
    POS 1. 3. 5.
    DEFBUTTON "OK" "DIALOG.END"
    CLOSE "DIALOG.END"

See also

- DIALOG
- DIALOG.Area
- DIALOG.Enable
- DIALOG.Message
- DIALOG.Select
- DIALOG.YesNo
- DIALOG.Dir
- DIALOG.End
- DIALOG.Execute
- DIALOG.Program
- DIALOG.Set
- DIALOG.SetDir
- DIALOG.Disable
- DIALOG.File
- DIALOG.ReProgram
- DIALOG.SetFile

▲ ‘Dialog Programming’ in ‘IDE User’s Guide’
DIALOG.YESNO

Create dialog box with YES and NO buttons

Format: 

```
DIALOG.YESNO <message>
```

Creates a standard dialog box with a question mark icon and the buttons Yes and No. The result is returned like the result value of a subroutine.

```
LOCAL &result
DIALOG.YESNO "Program FLASH memory?"
ENTRY &result
IF !&result
   ENDDO
...
```

For information about line breaks and the line continuation character, see DIALOG.OK.

**NOTE:** The DIALOG.YESNO command is very useful if it is combined with SETUP.QUITDO.

With SETUP.QUITDO you can define a PRACTICE script which will be executed before TRACE32 quits.

See also

- DIALOG
- DIALOG.MESSAGE
- DIALOG.OK
- DIALOG.view
- FORMAT.Decimal()
- 'Dialog Programming’ in 'IDE User's Guide'
- 'Release Information’ in 'Release History’
- 'I/O Commands’ in 'Training Script Language PRACTICE’
DIR

List subdirectories and files

| Examples |

Format: `DIR [<pathname>] [/PATH] | [/Recursive <depth>]`

Opens a DIR window, listing the contents of the specified directory or the contents matching the search criterion. You can use the asterisk character (*) as a wildcard.

You can drag and drop files into the TRACE32 command line in order to execute the file, e.g. a PRACTICE script file (*.cmm). This is useful for executing PRACTICE script files that expect TRACE32 command line arguments.

`PATH` The PATH option displays all directories of the search path, which is defined by the PATH command.

`Recursive <depth>` Depth of recursion. Starting at <pathname>, this option includes the subdirectories and their files in the listing. If <depth> is not specified or set to 0, then all subdirectories and files are included in the listing.

Left-click a file to display additional information in the message line (path, size, and date-timestamp).

Right-click a file to open the File popup menu:

- **DUMP** creates a binary file dump.
- **TYPE** opens the file as read-only.
- **EDIT** opens the file in the internal TRACE32 editor, unless you have configured an external editor with `SETUP.EDITEXT`.
- **DO** starts a PRACTICE script (*.cmm).
- **PSTEP** lets you step through a PRACTICE script.
- **PEDIT** opens the file in the PRACTICE script editor.
- **Open folder** opens the file explorer and selects the file - useful when you want to place a PRACTICE script file under version control in a version manager such as SVN.

Double-clicking directory names printed in bold opens the selected directory in a new **DIR** window.

**Examples**

**Example 1:**

```
DIR *.c ; show all '.c' files
```

**Example 2:** The path prefix `~` expands to the system directory of TRACE32.

```
;List all cmm files under the specified path and include the next two
directory levels in the listing
DIR ~/demo/arm/compiler/*.cmm /RECURSIVE 2
```

**See also**

- `LS`
- `OS.Hidden`
- `PWD`
- `SETUP.EDITEXT`

▲ 'File and Folder Operations' in 'IDE User's Guide'
▲ 'Release Information' in 'Release History'
DUMP

Binary file dump

Format: DUMP [<filename> [<offset>]] [/NoHex | /NoAscii | <option> …]

<option>:

WIDTH [<columns>]
Track

Displays a binary file in hex and ASCII format. Without arguments the command displays the last file that gave an error during download.

Scroll to file offset.

File size in bytes.

File offset entered in the command line. A small black arrow indicates the content at the file offset.

Current selection.

Right-click for popup menu.

Offset of current selection in decimal and hex.

<table>
<thead>
<tr>
<th>&lt;offset&gt;</th>
<th>File offset can be specified in decimal or hex.</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoHex</td>
<td>Hex display is switched off.</td>
</tr>
<tr>
<td>NoAscii</td>
<td>ASCII display is switched off.</td>
</tr>
<tr>
<td>WIDTH</td>
<td>Define display width in columns. Default is to automatically adjust to selected window setting.</td>
</tr>
<tr>
<td>Track</td>
<td>With Track enabled, the DUMP window tracks the selections you are making in the TYPE window. Prerequisite: The same file is open in both windows.</td>
</tr>
</tbody>
</table>
DUMP mcc.abs 0x1000 ; display file in hex and ASCII,  
; start at file offset 1000 (hex)

Data.LOAD.Ieee mcc.abs

...

ERROR ENTRY NEAR OFFSET 1234. IN FILE mcc.abs

DUMP ; display the file which caused the error

See also

■ PATCH ■ TYPE ■ Data.dump

▲ 'File and Folder Operations' in 'IDE User's Guide'
The TRACE32 editor is primarily used to create and edit short files. The editor can handle multiple files in separate EDIT windows. In addition, one and the same file can be edited in multiple EDIT windows at the same time.

Use the PEDIT command to create and edit PRACTICE scripts (*.cmm) in the PRACTICE script editor PEDIT.

See also
- EDIT.CLOSE
- EDIT.EXTern
- EDIT:file
- EDIT.List
- EDIT.LOAD
- EDIT.OPEN
- EDIT.QUIT
- EDIT.SAVE

EDIT.CLOSE

Close a text file

Format: EDIT.CLOSE [<filename>]

The selected file is stored and erased from the editor buffer. This command includes the commands EDIT.SAVE and EDIT.QUIT.

If no file name is defined, all files in the editor buffer will be stored and closed.

```
EDIT.CLOSE test.txt ; close one file
EDIT.CLOSE            ; close all files opened by an EDIT command
```

See also
- EDIT
- EDIT:file
- Edit ‘Editing’ in ‘IDE User's Guide’
**EDIT.EXTern**

Use specified external ASCII editor to edit file

<table>
<thead>
<tr>
<th>Format:</th>
<th>EDIT.EXTern <code>&lt;filename&gt;</code> [&lt;linenumber&gt;]</th>
</tr>
</thead>
</table>

This command opens a file with an external editor. The command line for the OS call of the external editor is defined by the **SETUP.EDITEXT** command.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDIT.EXTern my.txt</td>
<td>opens the file my.txt at line 1</td>
</tr>
<tr>
<td>EDIT.EXTern main.c 123.</td>
<td>opens the file main.c at line 123</td>
</tr>
</tbody>
</table>

**See also**

- EDIT
- SETUP.EDITEXT
EDIT.file

**Format:**

```plaintext
EDIT.file [<filename>] [<linenumber>]
```

Lets you edit the specified file in the internal TRACE32 editor - unless you have configured an external ASCII editor. In this case, **EDIT.file** opens the file in the external editor.

The following command allows to specify an external ASCII editor:

**SETUP.EDITEXT ON** `<commandline>`

Define command for external editor.

Example:

```plaintext
SETUP.EDITEXT ON "C:\Program Files (x86)\TextPad 5\TextPad.exe ""**""
EDIT.file C:\T32_MPC\per5500m.per ;the file now opens in TextPad
```

TRACE32 provides various special purpose editors:

<table>
<thead>
<tr>
<th>Editor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PEDIT</strong> <code>&lt;file&gt;</code></td>
<td>Editor for PRACTICE scripts.</td>
</tr>
<tr>
<td><strong>PER.Program</strong> <code>&lt;file&gt;</code></td>
<td>Editor for programming the peripheral description.</td>
</tr>
<tr>
<td><strong>MENU.Program</strong> <code>&lt;file&gt;</code></td>
<td>Editor to write a script that customizes the TRACE32 menu.</td>
</tr>
<tr>
<td><strong>DIALOG.Program</strong> <code>&lt;file&gt;</code></td>
<td>Editor to write a custom dialog.</td>
</tr>
<tr>
<td><strong>BITMAPEDIT</strong> <code>&lt;file&gt;</code></td>
<td>Bitmap editor for new toolbar buttons.</td>
</tr>
<tr>
<td><strong>MCDS.Program</strong> <code>&lt;file&gt;</code></td>
<td>Editor for programming the on-chip trigger unit.</td>
</tr>
</tbody>
</table>

See also

- **EDIT**
- **EDIT.OPEN**
- **EDIT.QUIT**
- **EDIT.CLOSE**
- **PEDIT**
- **SETUP.EDITEXT**
- **TYPE**
EDIT.List

Lists all open editor files. By clicking on a file name in this list, the according file will be opened for editing.

Format: `EDIT.List`

EDIT config.t32 ; edit one file
EDIT *.c ; edit one file with previous select menu
EDIT test.txt ; open file and edit
EDIT test.txt ; make new window for the same file

See also
- EDIT
- 'Editing' in 'IDE User's Guide'

EDIT.LOAD

Load text files

Format: `EDIT.LOAD [filename]`

Reloads a file from the host system. The temporary work copy of the file is rejected. If no file name is defined, all files opened by the editor will be reloaded from the host system.

EDIT test.txt ; open file with editor
... ; change file
...EDIT.LOAD test.txt ; reload original file

See also
- EDIT
- EDIT.SAVE
- 'Editing' in 'IDE User's Guide'
EDIT.OPEN [filename] [linenumber] [option]

Format: EDIT.OPEN [filename] [linenumber] [option]

<option>: AutoSave
NoSave

Opens the specified file with the internal TRACE32 editor - regardless of whether you have additionally configured an external editor with the SETUP.EDITEXT command.

<linenumber> The cursor is placed to a specified line if the <linenumber> parameter is used.

AutoSave This option will open a file, which is saved automatically after leaving the editor window.

NoSave (default) The file is not saved to disk.

EDIT.OPEN config.t32 ; open file config.t32 for editing
EDIT C:\T32_MPC\menp4xxx.men 50. ; cursor is placed into line 50. if specified file is opened for editing
EDIT.OPEN *.* ; "*" allows to use file browser to select the file

The following commands allow you to use an external ASCII editor:

SETUP.EDITEXT ON <commandline> Define command for external editor.
EDIT.file <file> Open specified file in external editor.
TRACE32 provides various special purpose editors:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PEDIT [&lt;file&gt;]</td>
<td>Editor for PRACTICE scripts.</td>
</tr>
<tr>
<td>PER.Program</td>
<td>Editor for programming the peripheral description.</td>
</tr>
<tr>
<td>MENU.Program</td>
<td>Editor to write a script that customizes the TRACE32 menu.</td>
</tr>
<tr>
<td>DIALOG.Program</td>
<td>Editor to write a custom dialog.</td>
</tr>
<tr>
<td>BITMAPEDIT [&lt;file&gt;]</td>
<td>Bitmap editor for new toolbar buttons.</td>
</tr>
<tr>
<td>Data.PROGRAM</td>
<td>Editor to write an assembler program.</td>
</tr>
</tbody>
</table>

If your TRACE32 tool provides a trigger language for your processor architecture, a trigger programming editor is provided:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCDS.Program</td>
<td>Editor for MCDS trigger language (TriCore architecture only).</td>
</tr>
<tr>
<td>Trace(Program</td>
<td>Editor for analyzer trigger language for TRACE32-ICE/FIRE.</td>
</tr>
</tbody>
</table>

See also

- EDIT
- EDIT.file
- SETUPEDITTEXT
- 'Editing' in 'IDE User's Guide'

## EDIT.QUIT

**Discard modifications**

Format:

```
EDIT.QUIT [<filename>]
```

This command is used for removing files from the editor buffer. All changes of the files are rejected. If no file name is defined, all opened files within the editor buffer will be released.

```
EDIT.QUIT test.txt ; don't save file test.txt
EDIT.QUIT ; ignore all changes in all text files
```
After the file has been stored, it remains in operating memory for further editing. If no file name is defined, all open editor files will be stored. The save function may be defined when opening a window by the AutoSave option.

EDIT test.txt
...
EDIT.SAVE test.txt
...
EDIT.CLOSE test.txt
EDIT.SAVE test1.txt test2.txt
EDIT.SAVE
EDIT test.txt /AutoSave

See also
■ EDIT
■ EDIT.file
■ EDIT.LOAD

▲ ‘Editing’ in ‘IDE User’s Guide’
ERROR.RESet

Reset PRACTICE error

Format: ERROR.RESet

The information structure of PRACTICE which contains data of the last occurred error will be cleared.

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()
  ENDDO
```

See also

- ERROR.ADDRESS()
- ERROR.ID()
- ERROR.OCCURRED()

▲ 'ERROR Functions' in 'IDE Functions'
EVAL

Eval
Evaluates expression

Format: Eval <expression>

Evaluates an expression. The result can be returned with the Eval() functions.

To try this script, copy it to a test.cmm file, and then run it in TRACE32 (See “How to…”).

```
SETUP.RADIX Hex ;set the default interpretation of numbers that
do not have the prefix 0x or postfix . to hex
AREA.view        ;open an AREA window
PRINT "NOTE: For a HEX value, start with the prefix 0x"
PRINT "      For an integer, append the postfix . (a period)"
RePeaT           ;run the user prompt in the AREA window
                   ;as an endless loop
  ON ERROR CONTinue ;your text for the user prompt
  PRINT "val="      ;generate a user prompt in the AREA window
  ENTER &a         ;and wait for the user input
  Eval &a          ;evaluate the user input with the Eval command
  IF EVAL.TYPE()==0x0004
    PRINT "You have entered the hex value 0x" %Hex &a
  ELSE IF EVAL.TYPE()==0x0008
    PRINT "You have entered the integer " %Decimal &a "."
  ELSE
    PRINT "You have entered '" &a '""
)
```

A Without the prefix ‘0x’ or the postfix ‘.’ the user input is interpreted as a hex value, see code line SETUP.RADIX Hex in the above example.
Eval Register(pc)==1000 ; evaluate expression
... 
IF (EVAL()!)=0) ; use in other command

ENTRY &delayvalue 
Eval &delayvalue ; evaluate user input value
... 
IF EVAL.TYPE()!)=0x400 ; timeval entered?
GOSUB err_no_timevalue

See also

- EVAL()
- EVAL.STRing()
- EVAL.TYPE()

▲ 'Release Information' in 'Release History'
Searches in a file for the occurrence of a string or bytes.

Example 1:

```
FIND test.c, "main(" ;search for the string "main(" in whole file
```

Example 2 shows how to search for a string in a file and, if the string is found, open the file in the TYPE window.

```
LOCAL &file
&file="~~/demo/arm/compiler/gnu/src/sieve.c"

FIND &file, "main(" ;search for the string "main(" in whole file

IF FOUND()==TRUE()
(
    ;if found, open file in TYPE window and
    ;scroll to the line where the string was found
    TYPE &file TRACK.LINE() /LineNumbers
)
```

See also

- ComPare
- TYPE
- Data.GREP
- WinFIND
- TRACK.COLUMN()
- Data.Find
- TRACK.LINE()

▲ 'FOUND Functions’ in 'IDE Functions’
▲ 'File and Folder Operations’ in 'IDE User's Guide’
FramePOS

**Controls the position of TRACE32 in MDI window mode**

<table>
<thead>
<tr>
<th>Format:</th>
<th>FramePOS &lt;left&gt; &lt;up&gt; &lt;hsize&gt; &lt;vsize&gt; [&lt;state&gt;] [&lt;colormode&gt;]</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;state&gt;:</td>
<td>Normal</td>
</tr>
<tr>
<td>&lt;colormode&gt;:</td>
<td>Auto</td>
</tr>
</tbody>
</table>

Controls the position and size of the TRACE32 main window if TRACE32 is configured to work in MDI window mode (Multiple Document Interface). In MDI mode, the TRACE32 windows and dialog boxes float freely inside the TRACE32 main window. Use the optional `<colormode>` parameter to set the toolbar and/or MDI background color to one of the available eight colors that can be assigned to cores and windows for multicore debugging.

For more information about the user interface, see “Graphical User Interface” (ide_user.pdf).

- `<left>`
  - x-coordinate as a floating point or integer or percentage value.

- `<up>`
  - y-coordinate as a floating point or integer or percentage value.

- `<hsize>`
  - Horizontal frame size in cursor width or percentage (only valid for “Normal” state)

- `<vsize>`
  - Vertical frame size in cursor height or percentage (only valid for “Normal” state)

**Normal**

The TRACE32 application is positioned at the given x- and y-coordinate with the chosen horizontal and vertical size.

**Iconic**

The TRACE32 application is minimized and an icon is shown in the task bar. Position and size values can be set but will have no effect.

**Maximized**

The TRACE32 application is maximized and fills the whole desktop. Position and size values can be set but have no effect.

**Top**

The TRACE32 window is activated and positioned above all other top-level windows.

**NOTE**: This state is currently only available under Microsoft Windows OS. A change of z-order resulting in a loss of input focus of a window can be prohibited by other applications. This is shown to the user as a colored flashing icon in the Windows Explorer taskbar.
**Auto**
Color TRACE32 instance (MDI parent window) dependent on the CORE=<number> parameter in the config file.
If the CORE parameter is not used, no coloring is done.
This option is recommended for AMP systems.

**DEFault**
Set default colors for toolbar and MDI background.

**<colorindex>**
TRACE32 instance (MDI parent window) is colored as defined for the Cores 0 to 7 in the SETUP.COLOR window (see screenshot below).

CORE parameter in config file:

```
PBI= USB CORE=2
```
### Examples:

<table>
<thead>
<tr>
<th>FramePOS ,,, Auto</th>
<th>; color TRACE32 instance dependent on the CORE parameter in the config file; CORE=2 -&gt; color of Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>FramePOS ,,, 1.</td>
<td>; color TRACE32 instance as specified for Core 1.</td>
</tr>
<tr>
<td>FramePOS 12.286 2.4167 90. 70.</td>
<td>; Position and size of TRACE32 GUI specified by fixed values</td>
</tr>
<tr>
<td>FramePOS 33% 0% 33% 75%</td>
<td>; Position and size of TRACE32 GUI specified by percentage</td>
</tr>
<tr>
<td>FramePOS , , , , Auto</td>
<td>; color TRACE32 instance (MDI parent window) dependent on the CORE=&lt;number&gt; parameter in the config file; recommended for AMP systems</td>
</tr>
</tbody>
</table>

### See also

- CmdPOS
- SETUP.COLOR
- WinExt
- CORE.SHOWACTIVE
  - ‘Screen Display’ in ‘IDE User’s Guide’
  - ‘Commands’ in ‘IDE User’s Guide’
  - ‘Software Installation’ in ‘TRACE32 Installation Guide’
  - ‘Release Information’ in ‘Release History’
The TRACE32 help system is divided in two parts:

- The **HELP** window is used to navigate through the help files and to search for any topic.
- An external PDF viewer displays the selected topics.

You can configure the TRACE32 help system with a few mouse-clicks to display the PDF help files in your favorite PDF viewer; see "**Configure the Help System**" (ide_user.pdf).

The **HELP** window can be accessed by pressing **F1** , using the **Help** menu, or by typing the **HELP** command at the TRACE32 command line.

See also

- HELP:Bookmark
- HELP:checkUPDATE
- HELP:command
- HELP:FILTER
- HELP:Find
- HELP:Index
- HELP:PDF
- HELP:PICK
- HELP:PRinT
- HELP:Topics
- HELP:TREE
- HELP:WINHELP
- SETUP:PDFViewer

▲ 'HELP System' in 'IDE User's Guide'
▲ 'Release Information' in 'Release History'
Format: HELP.Bookmark

Opens the bookmark page of the online help and shows the current bookmarks. A double-click will show the bookmarked file at the right place.

NOTE: Unsaved help bookmarks are only available during the current TRACE32 session.

If you want to re-use your help bookmarks in future sessions, remember to store your help bookmarks. The best way to accomplish this is to modify your PRACTICE start-up script (*.cmm), so that help bookmarks are stored automatically. See “Store and Load Help Bookmarks Automatically” (ide_user.pdf).

See also

- HELP.Bookmark.ADD
- HELP.Bookmark.DELETE
- HELP.Bookmark.show
- HELP
HELP.Bookmark.ADD.file

Add file to bookmark list

Format: HELP.Bookmark.ADD.file <file> [<description> <title> /<option>]

<option>: Page <page_number>

Adds a new PDF file to the help bookmark list. When closing the TRACE32 software, all bookmarks will be stored automatically. To store the bookmarks manually, use the Store command.

<page> Set the bookmark at this page number.
<description> This description will be displayed in the bookmark list
<title> This title will be displayed in the bookmark list

; Add the file "CPUdata.pdf" to the bookmark list - a double click will open this file
  HELP.Bookmark.ADD.file CPUdata.pdf

; Additionally, the description will be displayed in the bookmark list
  HELP.Bookmark.ADD.file CPUdata.pdf "Contains CPU info."

; Additionally, the description and the file title will be displayed in the bookmark list
  HELP.Bookmark.ADD.file CPUdata.pdf "Contains CPU info." "Data of CPU"

; A double click will open the file "CPUdata.pdf" on page 10
  HELP.Bookmark.ADD.file CPUdata.pdf "" "" /Page 10

See also
- HELP.Bookmark.ADD
- 'Release Information' in 'Release History'
**Format:**
```
HELP.Bookmark.ADD.Find <file> <find> [<description> <title> |<option>] 
```

- `<option>`: `Page <page_number>`

Adds a new PDF file to the help bookmark list. With the Find option, a find text can be added, and the bookmark will be set on the first occurrence of the find string.

- **Page**
  - Start searching for the find text on this page.

- **<description>**
  - This description will be displayed in the bookmark list.

- **<title>**
  - This title will be displayed in the bookmark list.

For example, use the heading text of the desired page you want see.

```
HELP.Bookmark.ADD CPUdata.pdf "Register Values" 
```

; Add CPUdata.pdf to the bookmark list.
; On a double click, find the first occurrence of the string “Register Values” in document and go there.

```
HELP.Bookmark.ADD CPUdata.pdf "Register Values" /Page 5 
```

; Additionally, start the search on page 5 - because the “Register Values” is first found in the list of contents.
; Now the search starts on page 5 and will find the right heading!

**See also**
- HELP.Bookmark.ADD
- 'Release Information' in 'Release History'
 Adds a new PDF file to the help bookmark list. To specify the exact position in the file, you can use “named destinations” as described in the PDF specification. Add a bookmark on a named destination with this command:

```
HELP.Bookmark.ADD.Index CPUdata.pdf "g154634" ; Add a bookmark on the named destination "g154634" in the file "CPUdata.pdf"
```

Additionally, the description and the file title will be displayed in the bookmark list.

```
HELP.Bookmark.ADD CPUdata.pdf "g154634" "Chapter 17: Registers" "Data of CPU" ; Additionally, the description and the file title will be displayed in the bookmark list
```

See also
- HELP.Bookmark.ADD
- ‘Release Information’ in ‘Release History’

**HELP.Bookmark.DELete**

Delete from bookmark list

```
Format: HELP.Bookmark.DELete <value>
```

Deletes a bookmark from the list of bookmarks, `<value>` is the position in the bookmark list (counting starts with 0).

See also
- HELP.Bookmark
- HELP.Bookmark.show
HELP.Bookmark.show

Opens the bookmark page of the online help and shows the current bookmarks.

See also
- HELP.Bookmark
- HELP.Bookmark.ADD
- HELP.Bookmark.DELete

HELP.checkUPDATE

Automatic update check for new help-files

Format: HELP.checkUPDATE ON | OFF

With HELP.checkUPDATE ON an automatic update check for new help files is performed.

See also
- HELP

HELP.command

Command related support

Format: HELP.command [<command_name> | <system_name>]

The command HELP without an argument displays the table of contents. An argument can be a command, or a prompt name.

HELP ;Displays the table of contents
HELP Register ;Displays information about the Register command

Alternatively, you can get help on a command even quicker by entering the command name and a trailing blank, and then pressing the «HELP» key (F1 on WINDOWS).

See also
- HELP
When the TRACE32 software is started, help filters are set automatically by the used TRACE32 hardware and/or software. The TRACE32 online help is filtered in order to display only the information relevant to your debug environment. Information from documents that do not refer to your debug environment is not accessible via the HELP window.

To check which help filters are automatically set by TRACE32, choose Help menu > Index:

You can switch all help filters on/off, add, and delete help filters using the commands listed in the See also block below. The help filters are listed in “Appendix A - Help Filters”, page 292.

See also
- HELP.FILTER.Add
- HELP.FILTER.Delete
- HELP.FILTER.RESet
- HELP.FILTER.set

```
HELP.FILTER.Add
```

Add a filter to the filter list

The online help for a TRACE32 RTOS debuggers and a the third-party tool integration can be enabled in order to get the corresponding information.

```
; add information on Linux debugging to the online help
HELP.FILTER.Add rtoslinux

; add information on the integration for the Visual Basic interface to
; the online help
HELP.FILTER.Add intvbas
```

See also
- HELP.FILTER
- HELP.FILTER.set
The content of the online help is filtered by the used hardware. This filter is set at system start-up automatically, so usually you don’t have to use this command. But if you wish to change the filter manually, use this command to remove a filter from the filter list.

You can use HELP.FILTER.Add to add a filter, and HELP.FILTER.set to disable all filters.

**See also**
- HELP.FILTER
- HELP.FILTER.set

HELP.FILTER.RESet

Removes all help filters and deactivates the use filter checkbox. As a result, information from all documents is now accessible via the HELP window, regardless of whether particular documents refer to your debug environment or not.

**See also**
- HELP.FILTER
- HELP.FILTER.set
HELP.FILTER.set without argument toggles the help filter; see the use filter check box in the HELP window.

**ON** (default) Activates the help filters, i.e. the use filter check box is selected. Documents, commands, and index entries that do not match the filter criteria are hidden in the HELP window.

**OFF** Deactivates the help filters, i.e. the use filter check box is cleared.
- Documents that do not match the filter criteria are accessible again, but displayed in gray in the HELP window.
- Previously hidden index entries are included in the HELP.Index again.

```
HELP.FILTER.set OFF ; disable all help filters
```

See also
- HELP.FILTER
- HELP.FILTER.Add
- HELP.FILTER.Delete
- HELP.FILTER.RESet
The command HELP.Find without an argument opens the HELP window on the Find tab. You can use one or more keywords to perform a full-text search.

The search can be customized by the following options:

<table>
<thead>
<tr>
<th>Case (GUI: case sensitive)</th>
<th>Toggle case sensitive search: If set, the search will find the string “MMU” but not “mmu”.</th>
</tr>
</thead>
</table>
| Similar (GUI: similar terms) | Toggle search for similar words: If set, the search will find synonyms and spelling variants for frequent terms, e.g.:  
  - multi-core and multicore  
  - onchip, on chip, and on-chip  
  - filename and file name  
  - logical address and virtual address, which are unavoidable synonyms, as explained in the “TRACE32 Glossary” (glossary.pdf). |
The Find string can be modified using the following operators:

- **text** general find text
- "text" *text* find exact the string inside “text” or ’text’
- -text find only files without text

Here are some examples on how to use the search options:

- HELP.Find ; Open help find window
- HELP.Find "data" ; Find the string “data”
- HELP.Find "MMU" /case ; Find the string “MMU” but not “mmu”
- HELP.Find " time -out " ; Find the string “time”, but only files without “out”
- HELP.Find " ’time-out’ " ; Find exact the string “time-out”

The colors of the find results have the following meaning:

- **black text** Result is normal text.
- **blue text** Result is a command.
- **bold blue text** Result is a heading.
- cyan Result is in a table.
- grey Result is an example.

See also

- HELP
- ’Release Information’ in ‘Release History’
Opens the **HELP.Index** window, displaying a complete alphabetic list of the TRACE32 commands, functions, and other indexed terms.

By default, the index list is filtered to reduce the keywords - only the keywords which are applicable for your hardware target are shown. If you want to see all keywords, disable the **use filter** check box.

The help index contains the full and short forms of the commands and functions. For more information about short and full forms, see “**Full Form and Short Form of Commands and Functions**” in “IDE User’s Guide” (ide_user.pdf).

- **All** Show the complete index list.
- **Basic** Show only basic commands - these are the most frequently used commands.
- **Command** Show only commands in the index list.
- **Function** Show only functions in the index list.
- **Short** Show only a short index file (no commands, no functions). The Short option refers to the **short index file** check box.

The colors of the index entry show the type:

- **blue text** command
- **bold blue text** basic command
- **cyan text** function
- **black** other types

**To find help via the Help window:**

1. Choose Help menu > **Index**.
2. Type the short form in **Find Index** box, and then press **Enter**.
3. In the **Help** window, double-click the index entry to open the pdf file in a PDF viewer, e.g. Acrobat Reader. Double-clicking takes you right to the description of the selected index entry.

---

**See also**

- [HELP](#)
- [HELP System](#) in 'IDE User’s Guide'

---

©1989-2018 Lauterbach GmbH

IDE Reference Guide 123
HELP.PDF

Opens a PDF file in a PDF viewer.

HELP.PDF ~~\pdf\ide_ref.pdf ;Open the IDE Reference Guide of TRACE32

The path prefix ~~ expands to the system directory of TRACE32, by default C:\T32. In a default installation, the pdf files of the online help reside in the pdf folder.

See also

■ HELP

▲ ‘HELP System’ in ‘IDE User’s Guide’

HELP.PICK

Context-sensitive help

Format: HELP.PICK

Offers a help cursor to get help on buttons, dialog boxes etc. The same command is also available by clicking this button ? on the TRACE32 main toolbar.

See also

■ HELP

▲ ‘HELP System’ in ‘IDE User’s Guide’
HELP.PRinT.PRinTSel  Print selected files

Format:  HELP.PRinT.PRinTSel [/No DiaLoG | /DiaLoG]

Prints all selected files - options see HELP.PRinT.show

See also
- HELP.PRinT.PRinTSel
- HELP.PRinT.SELect
- HELP.PRinT.show
- HELP.PRinT.UNSELect

HELP.PRinT.SELect  Select files to print

Format:  HELP.PRinT.SELect [<value>]

Selects the file number <value> to add it to the print list. If value is not set, all files are selected.

See also
- HELP.PRinT
HELP.PRinT.show

**Show print help files**

Format:  
HELP.PRinT.show [/NoDiaLoG | /DiaLoG]

Opens the print-page of the online help and shows a list of files to print.

**NoDiaLoG**  
If set, disable the Acrobat Reader print dialog and print all selected files immediately.

**DiaLoG**  
Shows the Acrobat Reader print dialog to change printer options like number of pages, page format.

See also  
- HELP.PRinT

HELP.PRinT.UNSELect

**Unselect all print files**

Format:  
HELP.PRinT.UNSELect [<value>]

Removes the file number `<value>` from the print list. If value is not set, unselect all files.

See also  
- HELP.PRinT
HELP.Topics

Help content list - document tree

Format: HELP.Topics [/Close | /Open]

Shows the content list of all available help files.

**Close**

Close all open tree branches.

**Open**

Open all tree branches.

The content list is a structural overview of all help files. You should use it to get a quick access to the desired help file. The major headings of every help file are also listed here to jump directly to the right chapter. Which chapters belong to one file can be identified by the shaded plumb line.

The currently selected file title is displayed in a tooltip. A double-click will open this file in a PDF viewer and will scroll to the right chapter if possible.
The content list structure can be seen also in the PDF help files. Every document has a interactive table of contents to navigate quickly over all help documents, and it shows also the position of the current file in the whole content list. Another feature to ease navigation is the bookmark list in the left frame of Acrobat Reader - use it to go directly to a chapter.

By default, a filter is applied to the content list. This filter is created from your currently used hardware at TRACE32 startup and is used to remove irrelevant documentation from the list. The filter can be disabled using the HELP.FILTER command - then the complete list of all help files is displayed. The irrelevant documents are shaded to show the difference. The filter can be changed using the commands HELP.FILTER.Add and HELP.FILTER.Del.

See also
- HELP
- 'HELP System' in 'IDE User's Guide'
All available commands for your hardware configuration are listed in alphabetical order. To search for a specific command, type some letters in the Find Command input field. The list automatically scrolls to the first command matching your input.

Double-clicking the desired command opens the appropriate help file in a PDF viewer.

**Close**
- Close all open tree branches.

**Open**
- Open all tree branches.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HELP.TREE &quot;Register&quot;</td>
<td>opens the HELP.TREE window, showing the Register command and its subcommands</td>
</tr>
<tr>
<td>HELP.TREE /Open</td>
<td>open all help tree branches</td>
</tr>
<tr>
<td>HELP.TREE /Close</td>
<td>close all help tree branches</td>
</tr>
</tbody>
</table>

The command tree is a complete reference of all softkeys, and is in the same hierarchical order as the softkeys. For example, the Register command has the subcommands: Register.COPY, Register.Up,...

These subcommands can be seen:

- In the command tree when you click the tree symbol.
- In the command tree when you type Register in the Find Command field.
- In the softkey line when you click the Register softkey button.
- In the softkey line when you type Register in the command line.

See also

- HELP
- 'Screen Display' in 'IDE User's Guide'
Opens a winhelp file (*.hlp) in the external winhelp viewer.

**See also**

- HELP
HISTory

Command history of last executed commands

The last executed commands **you have typed at the TRACE32 command line** are stored in a history buffer. The history buffer also logs command inputs which contain syntax errors and are not executed.

The "Page-up" and "Page-down" key scroll through the history buffer. Entering a keyword before using "Page-up" causes a search in the history for lines matching the keyword.

Direct commands, or those commands executed under PRACTICE are not logged. They can be recorded by the **LOG** command group.

See also

- HISTory.eXecute
- HISTory.SAVE
- HISTory.Set
- HISTory.SIZE
- HISTory.type
- LOG

▲ 'Create a PRACTICE Script' in 'Training Script Language PRACTICE'

HISTory.eXecute

Execute command history

Format: HISTory.eXecute

Executes all commands in the history list.

See also

- HISTory
- HISTory.SAVE
- HISTory.type

▲ 'Commands' in 'IDE User's Guide'
**HISTory.SAVE**

**Store command history log**

---

**Format:**

\[ HISTory.SAVE [<filename>] \]

Saves only the commands from the history file to the specified file. The resulting file has the format of a PRACTICE script.

```plaintext
; save the commands from the history to this PRACTICE script file
HISTory.SAVE ~/myCommandHistory.cmm

; open the file in the PRACTICE script editor of TRACE32
PEDIT ~/myCommandHistory.cmm
```

The path prefix `~~` expands to the system directory of TRACE32, by default `c:\t32`.

You can consider this script as your first draft toward your final script. The next step is then to edit the draft version of your script by adding program flow controls, such as **IF** ... **ELSE**, **WHILE** loop, **RePeaT** loop, etc.

---

**See also**

- `HISTory.Set`
- `HISTory.SIZE`
- `HISTory`
- `HISTory.eXecute`

▲ ‘Commands’ in ‘IDE User's Guide’
▲ ‘Create a PRACTICE Script’ in ‘Training Script Language PRACTICE’
HISTory.Set

Stores entries into the parameter history. This command is usually only used by the STORE HISTory command.

**Format:**

```
HISTory.Set <item> <string> [<string> <string>]
```

**<item>:**

- CMD
- FILE
- HLL
- ADDRESS
- RANGE
- TraceFIND

**CMD <string>**

Adds an entry to the TRACE32 command history (which can be viewed with the command HISTory.type).

**FILE <str1> <str2> <str3>**

Adds an entry to the list of recently used files in the FILE menu.

- <str1> is the name of the icon shown left of the entry.
- <str2> is the name of the command to be executed when clicking on the entry.
- <str3> is the working directory in which the command is executed.

**HLL <string>**

Adds an entry to the list of recently used HLL expressions e.g. in the Var.Break.Set window.

**ADDRESS <string>**

Adds an entry to the list of recently used addresses e.g. in the Break.Set window.

You can also cycle through the list of recently used addresses by pressing the softkey button <address> in connection with a command.

**RANGE <string>**

Adds an entry to the list of recently used address ranges e.g. in the Break.Set window.

You can also cycle through the list of recently used address ranges by pressing the softkey button <range> in connection with a command.

**TraceFIND <string>**

Adds an entry to the list of recently searched items in the expert search of a trace recording.

**See also**

- HISTory.SAVE
- HISTory
- HISTory.type
- ‘Commands’ in ‘IDE User’s Guide’
Using the **HISTore.SIZE** command group, you can define the (a) number of commands that can be stored in the command history as well as (b) the number of recently used files that can be displayed in the **File** menu.

**See also**
- **HISTory.SIZE.cmd**
- **HISTory.SIZE.FILE**
- **HISTory.SAVE**
- **HISTory**

▲ ‘Commands’ in ‘IDE User’s Guide’
▲ ‘Create a PRACTICE Script’ in ‘Training Script Language PRACTICE’

### HISTory.SIZE.cmd

**Define log size of command history**

**Format:**

```
HISTory.SIZE.cmd [<size>]
HISTory.SIZE [<size>] (deprecated)
```

When defining the log size of the command history, all former entries to the history are erased. Without selecting a size, the history log is erased only. Due to time constraints, the command history log is **always stored in operating memory**. Therefore, its size should be minimized (10 to 100.). The size is the number of lines with a maximum length of 100 character. Due to an optimized storage the effective number of history lines which can be used, is higher (smaller lines results in a longer history).

```
HISTory.type ; view command history
HISTory.SIZE.cmd ; clear history and set default size

HISTory.SIZE.cmd 100. ; define history with 100 entries
```

**See also**
- **HISTory.SIZE**
HISTory.SIZE.FILE

Define number of recently used files in "File" menu

Format:  HISTory.SIZE.FILE <size>

Defines the number of recently used files that are listed in the file history of the File menu.

Increasing the size will not erase the existing file history. Decreasing the size will only erase the oldest entries which no longer fit inside the new size of the file history size.

<size>  Default size is 10. Maximum size is 30.

See also
- HISTory.SIZE

HISTory.type

Display command history log of last executed commands

Format:  HISTory.type

Displays the command history log in the HISTory.type window. The highlighted bar indicates the current read position.

- Left-clicking a line copies the selected command to the TRACE32 command line. The command is not executed unless you press Enter.
- Right-clicking opens the popup menu, see below.
- Double-clicking a line immediately executes the selected command.

Displays the online help for the selected command.

See also
- HISTory
- HISTory.eXecute
- HISTory.SAVE
- HISTory.Set
- "Commands" in "IDE User's Guide"
Using the **IFCONFIG** command group, you can configure and test the Ethernet or USB communication between the TRACE32 PowerView GUI and the power debug interface of the Lauterbach hardware. In addition, the usage of resources can be visualized.

You can accomplish these task via the TRACE32 command line or via the **IFCONFIG.state** window.

### See also

- **IFCONFIG.PROfile**
- **IFCONFIG.DEVICENAME()**
- **IFCONFIG.IPADDRESS()**
- **IFTEST.LATENCY()**
- "IFCONFIG and IFTEST Functions’ in ‘IDE Functions’
- ‘Starting a TRACE32 PowerView Instance’ in ‘Debugger Basics - Training’
- ‘Starting a TRACE32 PowerView Instance’ in ‘Debugger Basics - SMP Training’
- ‘Starting a TRACE32 PowerView Instance’ in ‘Basic Debugging Intel® x86/x64’

### IFCONFIG.PROfile

**Display operation profiles**

Format:  

```
IFCONFIG.PROfile <option>
PROfile.[<item>] (deprecated)
```

<**option**>:

SEND | RECV | COL | ERROR | RETRY | RESYNC | KBYTE
FILECACHEMISSES | FILECACHEHITS | DPACKETS | RCLPACKETS | MAINTHREAD | STREAMIN | STREAMCOM | STREAMBUFFER | STREAMOUT | STREAMFILE | STREAMTHREAD

Display a time profile about the usage of resources.
Window with time profile about the usage of resources.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COL</td>
<td>Collisions when sending packets on Ethernet.</td>
</tr>
<tr>
<td>DPACKETS</td>
<td>Debug access packets.</td>
</tr>
<tr>
<td>ERROR</td>
<td>Communication errors.</td>
</tr>
<tr>
<td>FILECACHEHITS</td>
<td>File cache hits. (CHITS)</td>
</tr>
<tr>
<td>FILECACHEMISSES</td>
<td>File cache misses. (CMISSES)</td>
</tr>
<tr>
<td>KBYTE</td>
<td>Communication utilization in KBytes, all packets received and sent.</td>
</tr>
<tr>
<td>MAINTHREAD</td>
<td>Main thread utilization.</td>
</tr>
<tr>
<td>RCLPACKETS</td>
<td>Remote control packets.</td>
</tr>
<tr>
<td>RECV</td>
<td>Received packets (including Multicast/Broadcast).</td>
</tr>
<tr>
<td>RESYNC</td>
<td>Resyncs after communication fails.</td>
</tr>
<tr>
<td>RETRY</td>
<td>Retransmitted packets.</td>
</tr>
<tr>
<td>SEND</td>
<td>Sent packets.</td>
</tr>
<tr>
<td>STREAMBUFFER</td>
<td>Trace streaming buffer fill state.</td>
</tr>
<tr>
<td>STREAMCOM</td>
<td>Trace streaming communication rate.</td>
</tr>
<tr>
<td>STREAMFILE</td>
<td>Trace streaming file output rate.</td>
</tr>
<tr>
<td>STREAMIN</td>
<td>Trace streaming target input rate.</td>
</tr>
<tr>
<td>STREAMOUT</td>
<td>Trace streaming processing rate.</td>
</tr>
<tr>
<td>STREAMTHREAD</td>
<td>Trace streaming processing thread utilization.</td>
</tr>
</tbody>
</table>

See also

- IFCONFIG
- IFCONFIG.state
- IFCONFIG.TEST

▲ 'Interface' in 'IDE User's Guide'
IFCONFIG.TEST

Test interface function and speed

Format: IFCONFIG.TEST [default | Read | Write | ReadWrite [/<option>]]

IFTEST (deprecated)

<option>: Download | Upload | Warp [<warp>] | Latency

Measures the performance of upload, download, and latency of the connection to the debug interface. The result is displayed in the message bar and in the AREA.view window.

AREA.view ;open an AREA window. The test results will be ;displayed in this window
IFCONFIG.TEST ;run the test. The AREA window will be updated ;with the test results
IFCONFIG.state ;alternatively, open the IFCONFIG.state window and ;click the TEST button

This test only tests and measures the connection between host and debug interface. It is not directly related to the upload / download performance from / to the target, but a slow connection to the host will effect the max. possible upload / download performance to the target.

Download Download speed from host to TRACE32
Upload Upload speed from TRACE32 to host
Warp [<warp>] High-speed trace upload (for PowerTrace and CombiProbe). TRACE32 automatically determines the optimal warp speed.
Latency Round-trip time for a small packet, similar to a ping

See also
- IFCONFIG
- IFTEST.LATENCY()
- IFTEST.DOWNLOAD()
- IFTEST.UPLOAD()
- IFCONFIG.PROfile
- IFCONFIG.state

▲ 'Interface' in 'IDE User's Guide'

©1989-2018 Lauterbach GmbH

IDE Reference Guide 138
Opens the IFCONFIG.state dialog used for configuring Ethernet / USB connections.

![IFCONFIG.state dialog](image)

The easiest way to set the device name for an Ethernet configuration is to start with a USB connection. Changing an existing TRACE32 USB configuration to a TRACE32 Ethernet configuration involves these main steps:

- Assign a host name to the TRACE32 device.
- Modify the configuration file for Ethernet.
- Power off the device, disconnect USB, re-connect it via an Ethernet cable, and power up again.

For a detailed step-by-step procedure, see “Minimal Manual Setup” (installation.pdf).

Functions can be used in PRACTICE scripts to return individual values from the dialog. For more information, refer to the functions listed in the See Also section below.

**ip address**

IP address for the debug interface. In order to change the field, you need to disable the options RARP, BOOTP or DHCP.

**ethernet address**

Displays the ethernet address of the debug interface (read-only)

**device name**

Device name that is used to retrieve the IP address via DHCP.

If the debug interface is connected via USB, this device name can be used to identify the debug interface, especially if multiple debug interfaces are connected via USB.

**RARP**

Use the Reverse Address Resolution Protocol (RARP) to retrieve the IP address.
**BOOTP**
Use the Bootstrap Protocol (BOOTP) to retrieve the IP address.

**DHCP**
Use the Dynamic Host Configuration Protocol (DHCP) to retrieve the IP address corresponding to the device name (see above).

**full duplex**
Enable full duplex for the ethernet port.

**licence key**
Licence key to unlock ethernet support for workstations (not any longer required since 07/2011 for workstations and 04/2006 for PC based TRACE32 software).

**statistics**
Displays a live chart in a `IFCONFIG.PROfile` window.

**TEST**
Tests the interface function and speed, see `IFCONFIG.TEST`.

**Save to device**
Saves the device name to the internal memory of the TRACE32 device (e.g. PowerDebug / PowerTrace device).

See also
- `IFCONFIG`
- `IFCONFIG.PROfile`
- `HOSTID()`
- `IFCONFIG.DEVICENAME()`
- `IFCONFIG.IPADDRESS()`

▲ 'IFCONFIG and IFTEST Functions’ in ‘IDE Functions’
▲ 'Interface’ in ‘IDE User's Guide’
The **INTERCOM** system allows the exchange of data between different TRACE32 systems. The exchange can be based on TCP/IP or, if not possible, through files on a network drive. The destination system is defined by an InterCom name. This name is either the name and port number of a UDP port used by this TRACE32 system or a file name. Using TCP/IP for communication is preferred. This requires an entry in the 'config.t32' file of any participating TRACE32 system:

```
IC=NETASSIST
PORT=20001
...
```

**NOTE:** If multiple TRACE32 systems are used on one host, the port numbers must differ!

The following example uses TCP/IP for the cross tracking:

```
emulator 1                   emulator 2
Analyzer.XTrack ste:20002    Analyzer.XTrack ste:20001
```

When multiple (PODBUS) systems are connected to ONE system controller, it is possible to select the different systems with an extended intercom address:

```
debugger 1 (100)              debugger 2 (010)
SYnch.Connect ste:20000.010   SYnch.Connect ste:20000.100
```

**See also**

- INTERCOM.Evaluate
- INTERCOM.execute
- INTERCOM.executeNoWait
- INTERCOM.PING
- INTERCOM.PipeCLOSE
- INTERCOM.PipeOPEN
- INTERCOM.PipeREAD
- INTERCOM.PipeWRITE
- INTERCOM.WAIT
- SETUP.INTERCOMACKTIMEOUT
- TargetSystem.state
- INTERCOM.PODPORT()
- INTERCOM.PORT()
- ‘INTERCOM Functions’ in ‘IDE Functions’
**Format:**

\[
\text{INTERCOM.Evaluate } <\text{intercom}\_\text{name}> [<\text{function}>]
\]

Retrieves the result of a function executed on the remote system. Once retrieved, the result can be accessed by using the (local) \textit{EVAL()} function. If no function is specified, the result of the remote \textit{EVAL()} function will be retrieved.

**Example:** Read the value of the register DEC of the remote system:

```
INTERCOM.Evaluate ste:20002 Register(DEC)

&remote_register_value=EVAL()

PRINT "DEC= " EVAL()
```

**See also**

- \textbf{INTERCOM}
- \textbf{INTERCOM.execute}
INTERCOM.execute

Execute command via INTERCOM

Format:

```
INTERCOM.execute <intercom_name> <command>
```

Executes TRACE32 commands on the remote system. The commands will be executed immediately. The local system will wait until the remote system has completed the command.

**NOTE:** When executing a PRACTICE script (*.cmm) on the remote system using `INTERCOM.execute <... DO <filename>` the local system will wait until the `DO` command has *invoked* the script, but not until the script has terminated.

For waiting until the script terminated, use `INTERCOM.WAIT`.

**Example:** Execute commands on the remote system:

```
INTERCOM.execute localhost:20002 Register.RESet
INTERCOM.execute localhost:20002 Go.direct
```

**See also**

- `INTERCOM`
- `INTERCOM.PipeCLOSE`
- `INTERCOM.WAIT`
- `INTERCOM.Eval`
- `INTERCOM.PipeOPEN`
- `TargetSystem.state`
- `INTERCOM.executeNoWait`
- `INTERCOM.PipeREAD`
- `INTERCOM.PipeWRITE`
- `INTERCOM.PING`

INTERCOM.executeNoWait

Execute command via INTERCOM

Format:

```
INTERCOM.executeNoWait <intercom_name> <command>
```

Executes TRACE32 commands on the remote system. The commands will be executed immediately, and the local system will *not wait* until the remote system has completed the command.

Use `INTERCOM.executeNoWait` if you don’t need to wait locally until the command has completed, or if the command takes a long time to complete.

**Example:** Execute commands on the remote system:

```
INTERCOM.NW ste:20002 Data.LOAD.Elf bigfile.elf
```

**See also**

- `INTERCOM`
- `INTERCOM.execute`
**INTERCOM.PING**

Test INTERCOM system

<table>
<thead>
<tr>
<th>Format:</th>
<th><strong>INTERCOM.PING</strong> <code>&lt;intercom_name&gt;</code></th>
</tr>
</thead>
</table>

Sends one test message through the InterCom system to another TRACE32 system. If everything works, the other system will display the message 'PING received' and the sending TRACE32 system will display the message 'PING response received'.

Checks the connection of the TRACE32 system with the InterCom UDP port 20002 on host node 'ste':

```
INTERCOM.PING ste:20002
```

Checks the communication to a TRACE32 system without TCP/IP using the file xxx as communication port:

```
INTERCOM.PING c:\tmp\xxx
```

**See also**

- **INTERCOM**
- **INTERCOM.execute**
- **INTERCOM.PING()**

---

**INTERCOM.PipeCLOSE**

Close named pipe

<table>
<thead>
<tr>
<th>Format:</th>
<th><strong>INTERCOM.PipeCLOSE</strong> <code>#&lt;fileno&gt;</code></th>
</tr>
</thead>
</table>

Closes the named pipe.

**See also**

- **INTERCOM**
- **INTERCOM.execute**
INTERCOM.PipeOPEN

Opens or creates a named pipe. Named pipes allow to exchange data between different applications. The usage depends on the host OS.

Format: INTERCOM.PipeOPEN #<fileno> <filename> [/<option>]

<option>: Read | Write | Create

Opens a pipe for listening on Windows
;NOTE: The directory name is fixed for Windows hosts!
INTERCOM.PIPEOPEN #1 \\.\pipe\mypipe /r /c

See also
■ INTERCOM ■ INTERCOM.execute

INTERCOM.PipeREAD

Read from named pipe

Format: INTERCOM.PipeREAD #<fileno> <macro>

Gets input from a named pipe. Similar to the PRACTICE READ command. If the pipe has no data ready the command returns empty strings.

See also
■ INTERCOM ■ INTERCOM.execute

INTERCOM.PipeWRITE

Write to named pipe

Format: INTERCOM.PipeWRITE #<fileno> <arglist>

Writes data to a named pipe. Similar to the PRACTICE WRITE command.

See also
■ INTERCOM ■ INTERCOM.execute
**INTERCOM.WAIT** has two main applications:

- Connect to a remote system via **INTERCOM** and wait until the remote system is responsive.
- Wait until the remote system finished executing a running script i.e. until the PRACTICE interpreter becomes “idle”. NOTE: When a PRACTICE script is interrupted, e.g. by an input dialog, it is considered to be idle and causes the **INTERCOM.WAIT** command to return.

The command can be used to establish the connection to a remote system as soon as the remote system has booted and listens to **INTERCOM**.

**Example:** Start a second TRACE32 system and wait until it can be controlled via **INTERCOM**:

```plaintext
DO start_trace32_b.cmm    ; start debugger that listens on port 10001
INTERCOM.WAIT localhost:10001
```

See also
- **INTERCOM**
- **INTERCOM.execute**
Using the LICENSE command group, you can list the serial numbers and maintenance contracts of your debugging product and update your maintenance license.

**License List**

Using the LICENSE command group, you can list the serial numbers and maintenance contracts of your debugging product.

**Example 1:** LICENSE.List window for a setup with a debug cable and preprocessor.

If you are using an In-Circuit Debugger, the window shows also the feature keys stored in your Debug Cable, Nexus Adapter or Preprocessor.

Opens a window which shows all serial numbers and corresponding maintenance contracts of your debugging product.

**Format:**

```
LICENSE.List
```

**Display all license information**

License functions in 'IDE Functions' 'Release Information' in 'Release History' 'Do you have a valid Software License Key?' in 'Software Updates'

License List

Opens a window which shows all serial numbers and corresponding maintenance contracts of your debugging product.

License List window for a setup with a debug cable and preprocessor.

Opens a window which shows all serial numbers and corresponding maintenance contracts of your debugging product.

License List window for a setup with a debug cable and preprocessor.
Example 2: LICENSE.List window for a software-only setup using LICENSE.REQuest:

See also
- LICENSE
- 'Release Information’ in 'Release History'

LICENSE.REQuest

Request a license

<table>
<thead>
<tr>
<th>Format 1:</th>
<th>LICENSE.REQuest.plain &lt;product&gt; [&lt;version&gt;]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format 2:</td>
<td>LICENSE.REQuest.&lt;subcmds&gt;</td>
</tr>
<tr>
<td>&lt;subcmds&gt;:</td>
<td>FRONTEND</td>
</tr>
</tbody>
</table>

Requests a specific license from TRACE32. If the requested license is not yet available to TRACE32, then the license is checked out from an RLM server. The checked-out license is then blocked for the duration of the TRACE32 session.

You can view the licenses used by TRACE32 in the LICENSE.List window. To check the state of the license in a PRACTICE script (*.cmm), use the function LICENSE.GRANTED().

- `<product>` License product name as a string, e.g. as given in a lauterbach-*.lic file. For example: “t32.trace.x86”
- `<version>` License version as a string, 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.
- FRONTEND Requests a frontend license for the current architecture and version.
- INTEGRATION Requests a license for the currently selected third party integration.
MULTICORE Requests a multicore license for the current software version.
SIMULATOR Requests a frontend license for the current architecture and version.
BACKEND Requests a license for the currently selected backend.
TRACE Requests a trace license for the current architecture and version.

Please note that it is possible to request and check out Lauterbach licenses (if the license server has them) that are not required to run the current TRACE32 version. This is convenient for testing, e.g. to make sure a particular license is available on the license server.

See also

- LICENSE
- LICENSE.GRANTED()

**LICENSE.state** Display the currently used maintenance contract

**Format:** LICENSE.state

Shows the state of the currently used maintenance contract. You can also update your maintenance license via this window.

![License state window](image)

**NOTE:** This window shows the build-date up to which you may use TRACE32. It does not show which CPU architectures you have licensed. Use LICENSE.List or VERSION.view to show which CPU architectures you can use with your debug system.

See also

- LICENSE
- ‘Do you have a valid Software License Key?’ in ‘Software Updates’
LICENSE.UPDATE

Update the maintenance contract(s) inside your plugged Debug Cable or Nexus Adapter.

Examples:

```bash
; example for <license_file>
; updates all maintenance contracts in currently used Debug Cable/
; Nexus Adapter from data in given file
LICENSE.UPDATE license095970.t32
```

```bash
; example for <maintenance>
; stores given maintenance contract to currently used Debug Cable/
; Nexus Adapter
LICENSE.UPDATE C09110125362 12/2011 9a090df28631ac9c

LICENSE.UPDATE "C09110125362 12/2011 9a090df28631ac9c"
```

```bash
; stores currently used maintenance contract to Debug Cable/Nexus Adapter
LICENSE.UPDATE
```

See also

- LICENSE

▲ 'Release Information' in 'Release History'
Using the **LOG** command group, you can trace all executed TRACE32 commands. Operations activated by the mouse will be changed to line-oriented commands. The commands can either be stored in a command log file (by default t32.log) or printed to an **AREA** window.

Regardless of which output you choose, the trace information is recorded line by line in the command line format.

### Command Log File (t32.log)

Every new **LOG.OPEN** command generates a new command log file, overwriting the old one. The size of the log file is *unlimited*. Once the command log file has been activated, command execution (especially in the case of PRACTICE) will slow down due to the recording.

### AREA Window

The size of an **AREA** window is by default limited to about 100 lines. However, you can increase the number of lines with the **AREA.Create** command. To route command log entries to the **AREA** window, use the command **LOG.toAREA**.

---

See also

- **LOG.CLOSE**
- **LOG.OPEN**
- **LOG.toAREA**
- **LOG.type**
- **LOG.ON**
- **LOG.OFF**
- **HISTory**
- **SYStem.LOG**

"Command Logging" in 'IDE User's Guide'

---

### LOG.CLOSE

**Close command log**

**Format:**

```
LOG.CLOSE
```

The activated command log file is closed.

```
LOG.OPEN ; opens file 't32.log'
...
LOG.CLOSE ; close file and terminate logging function
```

See also

- **LOG**
- **LOG.OPEN**
- **LOG.type**

"Command Logging" in 'IDE User's Guide'

"Create a PRACTICE Script" in 'Training Script Language PRACTICE'
LOG.OFF

Switch off command log

Format: LOG.OFF

The commands are no longer logged. However, the command log remains operational. It can be reactivated by the LOG.ON command.

```
LOG.OPEN  ; opens file 't32.log' and commands are logged
...       
LOG.OFF   ; temporarily switch off log function -> commands are not logged
...       
LOG.ON    ; switch on log function -> commands are logged
...       
LOG.CLOSE  ; close file 't32.log' and terminate log function
```

See also
- LOG.ON
- LOG
- LOG.type
- ‘Command Logging’ in ‘IDE User’s Guide’
- ‘Create a PRACTICE Script’ in ‘Training Script Language PRACTICE’

LOG.ON

Switch on command log

Format: LOG.ON

All commands are logged. This command can be used after the log has been turned off with the command LOG.OFF.

```
LOG.OPEN  ; opens file 't32.log' and commands are logged
...       
LOG.OFF   ; temporarily switch off log function -> commands are not logged
...       
LOG.ON    ; switch on log function -> commands are logged
...       
LOG.CLOSE  ; close file 't32.log' and terminate log function
```

See also
- LOG.OFF
- LOG
- LOG.type
- ‘Command Logging’ in ‘IDE User’s Guide’
- ‘Create a PRACTICE Script’ in ‘Training Script Language PRACTICE’
LOG.OPEN

A new command log file will be generated. Only one LOG command can be activated at one time. Nesting of files is not possible. If no file name is defined, the file ‘t32.log’ will be used.

LOG.OPEN ; opens file 't32.log'
...
LOG.CLOSE ; close file 't32.log' and terminate log function

See also
- LOG
- LOG.CLOSE
- LOG.type
- ‘Command Logging’ in ‘IDE User’s Guide’
- ‘Create a PRACTICE Script’ in ‘Training Script Language PRACTICE’
LOG.toAREA

Log commands by writing them to an AREA window

| Examples |

Format:  

LOG.toAREA ON | OFF ["<prefix>"] [/<option>]

<option>: 

ALL
IndentCalls
AREA <name>
COLOR <color>

<color>:  

NORMAL | BLACK | MAROON | GREEN | OLIVE | NAVY | PURPLE
TEAL | SILVER | GREY | RED | LIME | YELLOW | BLUE | FUCHSIA
AQUA | WHITE

Writes log entries about commands to the default AREA window A000 or a user-defined AREA window before they are executed. After pre-processing, the PRACTICE macros are replaced by their contents and comments are stripped before logging.

If an error occurs during the actual execution of the command, the error message is printed directly below the command that has just been executed.

In contrast to the LOG.OPEN command, the executed commands are not recorded in a command log file but printed to the AREA window. However, if you want to additionally record the log entries in a *.txt file, then use the AREA.OPEN command, as shown in example 2.

<prefix>  

User-defined prefix text. Each line in the log output of the AREA window can start with a <prefix>.

ALL  

With ALL:

- All commands executed by a PRACTICE script (*.cmm) are displayed in the AREA window.
- Commands you enter on the TRACE32 command line are also shown.

Without ALL (default):

- Only commands from the script are displayed in the AREA window.
- However, the following commands are exceptions; they are not shown: ON, GLOBALON, GOSUB, RETURN, GOTO, JUMPTO, DO, END, ENDDO, IF, ELSE, REPEAT, WHILE, Var.IF, Var.WHILE, GLOBAL, LOCAL, PRIVATE, ENTRY, PARAMETERS, RETURNVALUES.

AREA <name>  

Specifies the AREA window to which the log entries are written. By default, the log entries are written to the AREA window A000.

Alternatively, specify a user-defined AREA name you have created with the AREA.Create command.

COLOR <color>  

Prints the command log entries in color to the AREA window.

©1989-2018 Lauterbach GmbH
**IndentCalls**

The lines of sub-scripts called with **DO** and sub-routines called with **GOSUB** are indented. The calls themselves are also displayed in the **AREA** window. When using **IndentCalls**, the commands **DO** and **GOSUB** are shown irrespective of whether the **ALL** option is used or not.

The commands are indented with a single plus symbol for every hierarchic level opened by **GOSUB** or **DO**. The plus symbol is used instead of a space to allow you to see the hierarchic level of every command more easily. This is especially useful if there are lots of other messages in the **AREA** window in between the log messages.

### Example 1

In case of an error, an error message is printed in red below the command that has caused the error.

```
AREA.view A000 ;display the default AREA window.

LOG.toAREA ON /ALL /COLOR.GREEN ;log commands by writing them
to the AREA window.

CD.DO ~~\my-test-script-012.cmm ;for demo purposes, let's call
;a non-existing file to cause
;an error.

SYSem.state ;for demo purposes, a typo in the
;command SYStem.state to cause
;an error.

LOG.toAREA OFF ;terminate the log.
```

The commands that have caused the errors. The error messages are printed directly below.

![Error messages]

**A** The commands that have caused the errors. The error messages are printed directly below.
Example 2

The log entries are printed to the default AREA window A000 and are at the same time stored in a *.txt file using the AREA.OPEN command.

```
AREA.view A000
AREA.OPEN A000 ~~\my-log.txt
LOG.toAREA ON /ALL /COLOR.GREEN
; two commands for demo purposes:
Data.Set VM:0x0 "Hello World!" 0
Data.Set VM:0x30 "Hello Universe!" 0
LOG.toAREA OFF
AREA.CLOSE A000
EDIT.OPEN ~~\my-log.txt
```

;display the default AREA window.
;save output that will be shown in the AREA window to a file.
;log commands by printing them to the AREA window.
;set two zero-terminated strings to the TRACE32 virtual memory.
;terminate the log.
;close the output file.
;open the file in an EDIT window.
A *user-defined AREA* window is created for command logging, and all lines are preceded by a user-defined prefix.

;create a user-defined AREA window named *myLogAREA* for command logging
AREA.Create **myLogAREA**
AREA.view **myLogAREA**

;optionally, select the default AREA window **A000** if you want to prevent
;the result of any PRINT command from showing up in **myLogAREA**
AREA.Select **A000**

;log commands by printing them to **myLogAREA**, and format them in green
LOG.toAREA ON "user-defined prefix - " /AREA **myLogAREA** /COLOR.GREEN

;these commands are logged to **myLogAREA**
List.auto
Break.List
Trace.List
PRINT Register(PP)

;deactivate the logging function
LOG.toAREA OFF

---

**A** The return value of PRINT Register(PP) does **not** show up in the command log because
AREA.Select A000 routes the return value to the default **AREA** window **A000**.

**B** The return value of PRINT Register(PP) shows up in the command log if AREA.Select A000
is omitted from the above example script.

For information about how to save the contents of the **AREA** window as an *.html* file, see **PRinTer.FILE**.

See also
- **LOG**
- **LOG.type**
- **AREA**
- 'Message Windows' in 'IDE User's Guide’
LOG.type

Display command log

Format: LOG.type

Displays the current command LOG file.

A User ID assigned to ID= in the config.t32 file as well as the creation date and time of the command log file.

B This commented-out line indicates that command logging was temporarily suspended by LOG.OFF and resumed later on by LOG.ON.

C Diagonal lines indicate that the command log file has been closed with LOG.CLOSE.

See also
- LOG
- LOG.OPEN
- LOG.CLOSE
- LOG.OFF
- LOG.ON
- LOG.toAREA
- 'Command Logging' in 'IDE User's Guide'

LS

Display directory

Format: LS [<pathname>] [PATH] [Recursive]

For a description of the LS command, see DIR.

See also
- DIR
- 'File and Folder Operations' in 'IDE User's Guide'
- 'Release Information' in 'Release History'

©1989-2018 Lauterbach GmbH
The **MENU** command group allows to customize the following elements of the user interface:

- **Main menu bar**
- Accelerators, see `MENU.AddMenu` or `MENUITEM`
- **Main toolbar**
- **Local popup menus**
- **Local buttons**

The default configuration for the menu and toolbar is loaded from the `t32.men` file. This file must be present in the TRACE32 system directory. Additional items can be added to this menu by the `ADD` dialog statement without modifying this file.

**See also**

- `MENU.AddMenu`
- `MENU.AddTool`
- `MENU.Delete`
- `MENU.Program`
- `MENU.RESet`
- 'Icons' in 'IDE User's Guide'

### MENU.AddMenu Add one standard menu item

**Format:**

```
MENU.AddMenu <name> <command>
```

Adds a menu to the main menu bar. By default, this menu is named **User**. This command can be used to quickly add one item for temporary use. If more (or more complex) items need to be added, it is recommended to use the `Menu.Program` or `Menu.ReProgram` command. The parameters are the same as described for the `MENUITEM` statement.

```
MENU.AddMenu "In Byte,ALT+F10" "Data.In io:0x100"
```

**See also**

- `MENU.AddTool`
- `MENU`
- `MENU.RESet`
- 'Screen Display' in 'IDE User's Guide'
- 'Release Information' in 'Release History'
- 'TRACE32 PowerView' in 'Debugger Basics - Training'
- 'TRACE32 PowerView' in 'Debugger Basics - SMP Training'
- 'TRACE32 GUI' in 'Training FIRE Basics'
- 'TRACE32 GUI' in 'Training ICE Basics'
Add a button to the main toolbar. This command is useful to quickly add one button for temporary use. This means, the button is only available for the current TRACE32 session. If more (or more complex) items need to be added it is recommended to use the \texttt{MENU.Program} or \texttt{MENU.ReProgram} command. The parameters are the same as described for the \texttt{TOOLITEM} statement.

\texttt{<shorttext>} Max. two letters, case-sensitive, i.e. “sT” is displayed as “sT” on the button.

\texttt{<color>} To receive an overview of the supported colors, choose \texttt{Misc} menu > \texttt{Tools} > \texttt{Edit bitmap template}.

\texttt{<predefined_image>} To receive an overview of the built-in icons, choose \texttt{Misc} menu > \texttt{Tools} > \texttt{Display internal icon library}.

In this example, four temporary buttons are added to the main toolbar:

\begin{verbatim}
;icon only
MENU.AddTool "List functions" ":[aview]" "Help.Index , /FUNCTION"

;button with red text
MENU.AddTool "Open Data.List window" "DL,R" "Data.List /Track"

;button with white text against background icon. Icon name is :reg
MENU.AddTool "Register window" "R,W,:reg" "Register.view /Spotlight"

;button with black text
MENU.AddTool "Input Byte" "IB,B" "Data.In io:0x100"
\end{verbatim}
MENU.Delete

Delete nested menu

Format: **MENU.Delete** `<filename>`

Deletes a previously added menu from the TRACE32 menu definition.

**See also**

- **MENU**
- **MENU.AddTool**

MENU.Program

Interactive programming

Format: **MENU.Program** `<filename>`

Opens the **MENU.Program** editor, with which you can create menu or toolbar definition files. The editor provides an online syntax check. The input is guided by softkeys. The syntax for the definition file is described in section “Programming Commands”.

**Example:** In the View menu, a user-defined menu item called **Set PC to main** is added below the Dump menu item.

![Example of menu item addition](image)

**See also**

- **MENU**
- **MENU.ReProgram**

▲ ‘Screen Display’ in ‘IDE User’s Guide’
▲ ‘Customizing the TRACE32 PowerView GUI’ in ‘Training Menu’
MENU.ReProgram

Format:  

MENU.ReProgram [<filename>]

If you enter the command at the commandline without parameter, then the default menu file t32.men in the system directory is executed.

If the command is used in a PRACTICE script (*.cmm) without parameter, a menu or main toolbar definition is embedded in the PRACTICE script. The definition block must be enclosed in parentheses and follow the command MENU.ReProgram as shown in example 2.

With parameter, the corresponding file is compiled and executed. You receive an error message if the file contains any errors.

Example 1: The menu or toolbar definition is stored in a separate *.men file. It is executed by a PRACTICE script (*.cmm) with MENU.ReProgram <file>.men:

```plaintext
;your code

MENU.ReProgram ~/mymenu.men ;add or modify menu or main toolbar using a *.men file

;your code
```

Example 2: The menu definition is embedded in a PRACTICE script (*.cmm).

```plaintext
;your code

MENU.ReProgram ;embed menu definition (...) here

(  
  ADD
  MENU
  ( ;  <menu_name>
    POPUP "MyMenu"
    ( ;   <icon><item_name>  <command>
      MENUITEM "[:reg]MyMenuItem"  "Register.view /SpotLight"
      MENUITEM "[:syslog]System Log"  "SYStem.LOG.state"
    )
  )
)

;your code
```

©1989-2018 Lauterbach GmbH

IDE Reference Guide 162
Example 3: The definition of a user-defined main toolbar button is embedded in a PRACTICE script (*.cmm).

```plaintext
;your code

;embed toolbar button definition (...) here
MENU.ReProgram
{
    ADD
    TOOLBAR
    {
        <tooltip> <icon> <command>
        TOOLITEM "MyToolBarButton" "[:colors]" "Register.view /SpotLight"
    }
}
;your code
```

![Example toolbar button](image)

Example 4: The ampersand character & allows you to pass a PRACTICE macro to the user-defined main toolbar button, which is embedded in a PRACTICE script (*.cmm).

```plaintext
LOCAL &myPath

&myPath="%userprofile%\SVN\demo"

MENU.ReProgram
(& ;note that the ampersand (&) character is required here.
    ADD
    TOOLBAR
    {
        <tooltip><icon> <command>
        TOOLITEM " " "[:folder]" "OS.Hidden explorer.exe /root, &myPath"
    }
)
```

See also
- [MENU](#)
- [MENU.Program](#)
- ‘Screen Display’ in ‘IDE User’s Guide’
- ‘Customizing the TRACE32 PowerView GUI’ in ‘Training Menu’
MENU.RESet

Restores the default configuration of the menus and the main toolbar.

See also

- MENU
- MENU.AddMenu
- MENU.AddTool
- ‘Screen Display’ in 'IDE User's Guide'
- ‘TRACE32 PowerView’ in ‘Debugger Basics - Training’
- ‘TRACE32 PowerView’ in ‘Debugger Basics - Training’
- ‘TRACE32 PowerView’ in ‘Debugger Basics - SMP Training’
- ‘TRACE32 PowerView’ in ‘Debugger Basics - SMP Training’
- ‘TRACE32 GUI’ in 'Training FIRE Basics'
- ‘TRACE32 GUI’ in 'Training FIRE Basics'
- ‘TRACE32 GUI’ in 'Training ICE Basics'
- ‘TRACE32 GUI’ in 'Training ICE Basics'
- ‘Customizing the TRACE32 PowerView GUI’ in 'Training Menu'
Programming Commands

The syntax of a definition file is line oriented. Blanks and empty lines can be inserted to structure the script. Comment lines start with a semicolon. Examples of definitions reside in the directory ~/demo/menu.

ADD
Add definition to existing menu

Format: ADD

The menu definition is added to the existing menu definition. Without this command, the new definition replaces the old one. This command can be used on the top-level of the script only. It is valid for the whole file.

Behavior of subsequent ADDs after the first ADD:

<table>
<thead>
<tr>
<th>Usage of a different menu file or different PRACTICE script</th>
<th>The existing menu is retained and the new menu items are added.</th>
</tr>
</thead>
</table>
| Definition is embedded in the same PRACTICE script file (*.cmm) | Executing the embedded block again: MENU.Program

```
...
```

replaces the last embedded block with the new one. |
| Definition is in the same menu file (*.men) | Executing MENU.Program <my_men_file> with the same path and name replaces the previous menu definition with the new menu definition. |

ADDHERE
Define hook

Format: ADDHERE

When items are added to an existing menu, they are usually added to the end of the menu. The ADDHERE command allows to choose a different insertion point for additional items.
**AFTER**
Place a new menu option or separator after the named menu option

Format:  
**AFTER** "<menu_option_name>"

MENU.ReProgram ;embed menu definition in a PRACTICE script file (*.cmm)  
(  
ADD  
MENU  
( ;in the View menu...  
POPUP "&View"  
( ;... place two new menu options after the menu option "Registers"  
AFTER "Registers"  
MENUITEM "[:syslog]System Log" "WinResist.WinExt.SYStem.LOG.state"  
AFTER "Registers"  
MENUITEM "[:config]Interface Config." "WinResist.IFCONFIG.state"  
;... place a separator after the menu option "Registers"  
AFTER "Registers"  
SEPARATOR  
;... place a separator before the menu option "Dump..."  
BEFORE "Dump..."  
SEPARATOR  
)  
)  
)

Two new menu options and a separator have been inserted by the menu command **AFTER**.

**BEFORE**
Place a new menu option or separator before the named menu option

Format:  
**BEFORE** "<menu_option_name>"

For an example, see menu command **AFTER**.

©1989-2018 Lauterbach GmbH
BUTTONS

Format: \texttt{BUTTONS <window>}

Adds one or more user-defined local buttons to a window. The \texttt{BUTTONS} command can be used on the top-level of the script only. User-defined local buttons cannot be added to all windows.

The \texttt{<window>} parameter must be the short form of a command that opens a window, e.g. \texttt{F.} for \texttt{Frame.view} or \texttt{R.} for \texttt{Register.view}. Simply omit all lower-case letters to get the command short form.

You can include icons in user-defined local buttons and adjust the local button width with \texttt{WIDTH}. If a button text is longer than the permissible number of characters in a button, the button text starts to shrink or is cut off.

- \textit{With} an icon, the max. width of a button text is 6 characters, if you do not use \texttt{WIDTH}.
- \textit{Without an icon}, the max. width of a button text is 9 characters, if you do not use \texttt{WIDTH}.

This script adds two user-defined local buttons to the \texttt{Register.view} window, as shown above, for opening the \texttt{List.auto} and \texttt{BookMark.List} window. You can now easily navigate between the current position of the program counter (PC) and your bookmarks - if you have created any bookmarks.

```ide
MENU.ReProgram
{
ADD
BUTTONS "R."
{
;1st button
MENUITEM "[:list]List at PC"
{;determine whether the named window is already open
IF WIN.EXIST("myList")==FALSE()
{;apply a user-defined name to the window
WinPOS 0. 0. , , , , , myList
List.auto /MarkPC /Track
}
Data.GOTO Register(PC) ;go to the program counter (PC)
}
;2nd button
MENUITEM "[:bookmark]BookMrk" "Bookmark.List"
}
};let's make the modified window float above the other windows
WinExt.Register.view /SpotLight /Stack ;and open the window
```
DEFAULT

Define default item

Format: DEFAULT

Marks the next item as the default item of a menu. On some hosts, this item can be selected by double clicking on the popup menu which contains the default button.

DELETE

Delete a certain item

Format: DELETE <name>

The user given name string will be searched inside the specified popup menu and deleted if a corresponding menu item is found.

```
ADD
MENU
(
  POPUP "&OSE Delta"
  (
    DELETE "Enable OSEDelta awareness"
    DEFAULT
    MENUITEM "Display &Processes" "TASK.DProc"
    ...
  )
)
```

ELSE

Conditional compile

Format: ELSE

Used together with the IF statement to define a block that is only compiled when the IF condition is false.
Enables the next **MENUITEM** within a **MENU** block only if the condition is TRUE. Otherwise the **MENUITEM** is shaded out and cannot be selected.

**Example:** A menu definition is embedded in a PRACTICE script with **MENU.ReProgram**. The first menu item is always active, because it is used without **ENABLE**. The second menu item is used together with **ENABLE** and two conditions. As a result, the second menu item is only active if the two conditions are true.

```plaintext
LOCAL &path &exe
&path="~/demo/arm/compiler/arm" ;path to PRACTICE demo scripts
&exe=OS.PresentExecutableFile() ;get path and file name of TRACE32
exe=OS.FILE.NAME(&exe) ;executable

MENU.ReProgram ;embeds a menu definition in a PRACTICE script
(&
    ADD
        MENU
            ( ;this menu block creates a user-defined menu with two menu items
                POPUP "MyMenu"
                ( ;this menu item is always enabled
                    MENUITEM "Browse..." "CD.PSTEP ~/demo/*.cmm"
                )
                ( ;this menu item is enabled if TRACE32 runs as an instruction
                    ;set simulator and the TRACE32 executable is t32marm.exe
                    ENABLE (INTERFACE.SIM()==TRUE())&&("&exe"=="t32marm.exe")
                    MENUITEM "Start Demo" "CD.DO &path/arm9.cmm"
                )
            )
        )
&)
```

**A** Both conditions are TRUE. As a result, the second menu item is active.

**B** One of the two conditions is FALSE. As a result, the second menu item is grayed out and inactive.
HELP

Define a help item

Format: HELP <name>

tbd.

IF

Conditional compile

Format: IF <condition>

The following block is compiled only when the condition is true. The block may be followed by an ELSE statement. The condition is evaluated when the menu is compiled.
The following block contains the definition of a menu.

- Without parameters, a new menu is added to the main menu bar.
- With parameters, the menu can be a local popup menu in a specific window or a special local popup.

This command can be used on the top-level of the script only.

<table>
<thead>
<tr>
<th><code>&lt;cmd&gt;</code></th>
<th>Short form of a command. For information about command short forms, see “Long Form and Short Form of Commands and Functions” (ide_user.pdf).</th>
</tr>
</thead>
</table>
| `<special_name>` | - "DATA" adds a user-defined menu item to the built-in popup menu Program Address. It opens when you select an item which has an address.  
- "VAR" adds a user-defined menu item to the built-in popup menu Variable. It opens when you select an item which is a variable. |

**Example 1:** The MENU command is used without parameter to add a new menu called **MyPopup** to the main menu bar.

```
MENU.ReProgram ;embed menu definition in a PRACTICE script file (*.cmm) 
(
  ADD 
  ;add a menu to the main menu bar 
  MENU 
  ( 
    POPUP "MyPopup"
    ( 
      MENUITEM "MyItem" "HELP.Index"
    )
  )
)
```

©1989-2018 Lauterbach GmbH
Example 2: The `MENU` command takes a command short form as an argument to add a local popup menu to a specific window, here to the `List.auto` window. The command short form of `List.auto` is `L`.

```
MENU.ReProgram ;embed menu definition in a PRACTICE script file (*.cmm)
{
    ADD
    MENU "L"
    {
        SEPARATOR
        POPUP "MyPopup"
        {
            MENUITEM "MarkPC" "List.auto Register(PC) /MarkPC /Track"
            DEFAULT
            MENUITEM "Bookmark List" "WinExt.BookMark.List"
        }
    }
}
```

A Local popup menu.

B Menu items on the new local popup menu.

Example 3: The `MENU` command takes the `<special_name> `DATA` as an argument to add a menu item to the built-in popup menu `Program Address`:

```
MENU.ReProgram ;embed menu definition in a PRACTICE script file (*.cmm)
{
    ADD
    MENU "DATA"
    {
        MENUITEM "My Dump" "Data.dump TRACK.ADDRESS() /Long"
    }
}
```
**ITEM**

Defines an item in a menu, popup menu or a local button. The name of a menu can optionally contain a hotkey, and a mnemonic or an accelerator.

- The hotkey is the character that can be used to select the item. It must be a character of the name and is marked by prepending a "&" to the character.
- The mnemonic can be an abbreviation of the menu entry, e.g. EBU for External Bus Unit. The menu name and its mnemonic are separated by the semicolon character ";". The mnemonic is displayed right-aligned and has no special meaning.
- The accelerator is the name of a special key or combination, which can be used to activate the menu directly without browsing through the menu (e.g. F10 or ALT-X). The accelerator is separated from the menu name by a comma and displayed right-aligned.

The concurrent use of accelerators and mnemonics is not supported and results in undefined behavior.

The instruction for the menu can either be included as additional parameter, or as an embedded script after the **ITEM** definition.

**POPUP**

Defines a new popup menu. The popup can be part of a main menu or of another popup menu. The definition follows the command, embedded in round brackets.
The following menu item will replace an existing item with the same name. Otherwise the menu item will be added to the menu, even when the names are the same. The names are compared without menu labels and without accelerators. This allows also to change the labels of accelerators of the default menu.

```
ADD
MENU
(
    POPUP "File"
    ( REPLACE
        MENUITEM "Load..." "Data.Load.Ieee * e: /Puzzled /ZP2"
        ...
    )
    POPUP "Run"
    ( ; change the accelerator for step over call
        REPLACE
        "Step Over Call,F8" "Step.Over"
    )
    ...
)
```

**SEPARATOR**

Separator definition

- **Format:** `SEPARATOR`

Inserts a separator in a menu or toolbar.

**TEAROFF**

Define tearoff menu

- **Format:** `TEAROFF`

Marks the next popup menu as tearoff menu. Tearoff menus can be disconnected from the menu and placed anywhere on the screen. Tearoff functionality may not be available on all hosts.
TOOLBAR

Format: TOOLBAR

The following block contains the definition of the main toolbar. This command can be used on the top-level of the script only. This example shows how to add a button to the main toolbar.

```
ADD TOOLBAR
  ( TOOLITEM "Switch Operation Mode" "MD,X" "MODE"
  )
```

TOOLITEM

Format: TOOLITEM <tooltip_text> [<image>] [<command>]

<image>:
  ":<predefined_image>"
  "<text>[<color>[<predefined_image>]]"
  "[<bitmap_image>]"

Defines a button in the main toolbar. The tooltip text is displayed when the mouse is moved above the button. The toolbar image defines the layout of the button. It can contain a short text, a fixed image, the combination of both, or a user-defined image. A user-defined image can either be embedded in square brackets in the string or included after the TOOLITEM command embedded in square brackets. The instruction for the button can either be included as additional parameter, or as an embedded script after the TOOLITEM definition (round brackets).

The following colors can be used for the image and bitmap definition:

- **r, R** dark red / light red
- **g, G** dark green / light green
- **b, B** dark blue / light blue
- **m, M** dark magenta / light magenta
- **y, Y** dark yellow / light yellow
- **c, C** dark cyan / light cyan
The names of the predefined images can be found in `~/demo/menu/t32icon.h`. The images can also be used as a template for new bitmaps. Just copy the desired string contents. The bitmaps can be viewed or modified with the `BITMAPEDIT` command.

```c
MENU.ReProgram
{
  ADD
  TOOLBAR
  {
    TOOLITEM "Switch Operation Mode" "MD,X" "MODE"
    TOOLITEM "Dump File" ":Dump" "DUMP *"
    TOOLITEM "Load Binary File" "LF" "Data.LOAD.Binary *"
    TOOLITEM "Map and Load IEEE File" "LF,R"
    {
      MAP.RESet
      MAP.Default 0--0xffff
      Data.Load.Ieee *.x
    }
    TOOLITEM "Edit .c File" ".c,R,:edit" "EDIT *.c"
    TOOLITEM "Reload File" "DO reload"
    [ 
      XX
      XXX
      XXX
      XXXXXX
      XXXXXXXX
      XX XXXX
      XX XXX
      XX XX
      XX
      XX
      XXX
      XXX XXX
      XXXXXXXXX
      XXXXXX
      ]
  }
}
```
Sets the width of the next button that is defined with `MENUITEM` inside the group `BUTTONS`.

**Example:** The same `PERipherals` button is added five times to the `Register.view` window to illustrate the various button width settings. To try this script, simply copy and paste it into the `TRACE32` command line.

```
MENU.ReProgram
{
    ADD
    BUTTONS "R."
    {
        WIDTH NORMAL
        MENUITEM "[:chip]PERipherals" "PER.view"

        WIDTH WIDER
        MENUITEM "[:chip]PERipherals" "PER.view"

        WIDTH WIDEST
        MENUITEM "[:chip]PERipherals" "PER.view"

        WIDTH SMALLER
        MENUITEM "[:chip]PERipherals" "PER.view"

        WIDTH SMALLEST
        MENUITEM "[:chip]PERipherals" "PER.view"
    }
}

Register.view
```

A. NORMAL  B. WIDER  C. WIDEST  D. SMALLER  E. SMALLEST
**MKDIR**

Create new directory

<table>
<thead>
<tr>
<th>Format:</th>
<th>MKDIR &lt;pathname&gt;</th>
</tr>
</thead>
</table>

This built-in TRACE32 command **MKDIR** creates a new subdirectory.

**Example 1:**

```plaintext
MKDIR sub1 ; create directory  
ChDir sub1 ; change to directory  
ChDir ... ; go back
```

**Example 2:** The following example creates a folder only if it does not exist. In addition, the TRACE32 command **OS.Command** executes the host command *start* on the host operating system (OS) level: The Windows Explorer is started and the newly created folder is selected in Windows Explorer.

```plaintext
LOCAL &folder  
&folder="c:\temp2"  

; if the folder does not exist,  
IF OS.DIR(&folder)==FALSE()  
(  
    ; then create it  
    MKDIR &folder  
)  

; start Windows Explorer and select the folder  
OS.Command start explorer.exe /select, &folder
```

The host command is printed in blue. TRACE32 expands the PRACTICE macro **&folder** before it is passed to the host shell.

For more information about how to execute host commands on the host shell from within TRACE32, refer to the OS command group.

**See also**
- ChDir
- RMDIR
- ‘File and Folder Operations’ in 'IDE User's Guide'
Format: **MV** `<oldname> <newname>`

Renames a file. This command is not possible with wildcard characters in the file name.

See also

- **REN**
- **OS.FILE()**
The **OS** commands allow the execution of host commands on the host shell from within TRACE32.

The **OS** commands **OS.Area** and **OS.Window** and **OS.Hidden** read back the output of a host command from a temporary file in order to display the output in TRACE32 PowerView. Therefore, the TRACE32 configuration variables `SYS=` or `TMP=` in the config.t32 file have to point to a read and writable directory.

**Which OS commands are blocking to prevent forks in PRACTICE scripts (*.cmm)?**

- **Blocking:** The TRACE32 commands **OS.Area**, **OS.Window**, and **OS.Hidden** block PRACTICE script execution and wait for the host command to finish. Once the host command has finished, PRACTICE script execution continues. Use a blocking TRACE32 **OS** command if you want the PRACTICE script to process the output of the host command.

- **Non-blocking:** The TRACE32 command **OS.screen** does not block PRACTICE script execution. Consequently, the PRACTICE script and the host command will run in parallel.

  On Windows, the TRACE32 command **OS.Command** is always non-blocking. On Linux/Unix, append an ampersand ‘&’ to the command called with **OS.Command** to get a non-blocking behavior.

**What is the difference between **OS.screen** and **OS.Window**?**

- **OS.screen** alone opens just the command prompt window of the host shell. **OS.screen cmd /C** allows to execute a host command.

- **OS.Window** executes the host command and re-reroutes all outputs of this host command to a TRACE32 window called **OS.Window**. Use the **OS.Window** command if you are interested in immediately displaying the output of a host command in TRACE32, e.g. a directory listing of the host command dir.

**NOTE:** The Windows dir and the TRACE32 DIR command are not identical.

**What is the difference between **OS.Window** and **OS.Area**?**

- **OS.Window** *automatically* opens a TRACE32 window with the same name, executes the host command, and displays the host command output; any further user interaction is *not* necessary.

- **OS.Area** redirects the host command output to the active TRACE32 message area. *It is up to you* to decide when you want to view the output by executing the **AREA.view** command at the TRACE32 command line.

**See also**

- **OS.Area**
- **OS.Command**
- **OS.Hidden**
- **OS.screen**
- **OS.Window**

▲ ‘OS Functions’ in ‘IDE Functions’
Re-route host command output to AREA window

Format: `OS.Area <commandline>`

Executes a command on the host operating system (OS) level. Outputs of this host command are re-routed to the AREA window.

Outputs of the operating system may be viewed only. Running under DOS, most terminal-oriented programs do not use the operating system! During program execution nothing is displayed on the terminal. Therefore interactive program entries cannot be carried out. The host interface of the TRACE32 remains in active mode during execution. Executing the command without parameters will start the shell invisible to the user.

**Example:** The Windows `dir` command is executed from within TRACE32, and the output can then be viewed in the AREA window of TRACE32.

TRACE32 commands are formatted in bold. Windows commands are formatted in regular font.

```
;open an AREA window
AREA.view

; in the AREA window, list the file names of all PRACTICE scripts (*.cmm) that start with ‘dia’ and reside in the system directory of TRACE32.
OS.Area dir /b C:\T32\dia*.cmm

; list time stamps and file sizes of all *.cmm files starting with ‘dia’
OS.Area dir C:\T32\dia*.cmm
```

See also

- OS
- OS.screen
- OS.ENV()
- OS.FIRSTFILE()
- ‘Host Commands’ in ‘IDE User’s Guide’
OS.Command

Execute a host command

[About Blocking and Non-blocking OS Commands]

Format:  OS.Command [<commandline>]

If the command contains an argument, it will immediately be executed by the shell of the host. A single OS.Command can also pass multiple host commands to the host. In addition, PRACTICE macros can be used in the <commandline> passed from TRACE32 to the host. This allows you to combine PRACTICE, the Lauterbach script language for TRACE32, with the script language of the host. You can run the resulting PRACTICE script from within TRACE32.

If OS.Command does not contain any argument, it opens just a system shell.

To illustrate the OS.Command, the following examples are provided.

- Example 1: The copy command of the host copies files starting with ‘ide’ from folder A to folder B. The folders A and B are specified by two PRACTICE macros and two PRACTICE functions. After a successful copy operation, the start command of the host opens Windows Explorer, directly in the destination folder B.

- Example 2: The environment variables are written to a txt file, which is then opened in Notepad.

- Example 3: *.csv files are opened in Excel without starting additional Excel instances.

TRACE32 commands and functions are formatted in bold. Host commands are formatted in regular font. The conditional processing symbols && of the operating system are formatted in red.

Example 1 - Copy files (Windows)

To try this script, simply copy it to a test.cmm file, and then run it in TRACE32 (See “How to…”).

LOCAL &sFld &dFld ;declare TRACE32 PRACTICE macros

;initialize the PRACTICE macros using two PRACTICE functions
&sFld=OS.PresentHELPDirectory() ;source A: the pdf online help
directory of TRACE32

&dFld=OS.PresentTemporaryDirectory() ;destination B: the temporary
directory of TRACE32

;copy the files, then open Windows Explorer in the destination folder
OS.Command copy &sFld\ide*.pdf &dFld && start explorer.exe &dFld

For more information about conditional processing symbols, refer to the Windows Command-Line Reference.
Example 2 - List environment variables (Windows, Linux)

Windows:

```
; write environment variables to txt file
OS.Command set > %temp%\environment_variables.txt

; open txt file in an editor
OS.Command start notepad.exe %temp%\environment_variables.txt
```

Linux:

```
; write environment variables to txt file
OS.Command env > $TMPDIR/env.txt

; open txt file in an editor
OS.Command emacs $TMPDIR/env.txt &
```

Depending on your Linux installation, the environment variable for your TEMP folder might have a different name. You can list your Linux environment variables within TRACE32 by using the `OS.Window` command.

For more information about PRACTICE, the Lauterbach script language for TRACE32, refer to “PRACTICE Script Language User’s Guide” (practice_user.pdf).

Example 3 - Open *.csv files exported from TRACE32 in Excel without additional Excel instances

```
; set the working directory to c:\t32
PWD c:\t32

; export the function nesting to a *.csv file in the working directory
Trace.EXPORT.CSVFunc func.csv

; export the variables 'flags' and 'ast' to a *.csv file in the working directory
Var.EXPORT variables.csv %Type %Location %Index flags ast

; start only one instance of the default application associated with the file type *.csv, e.g. Excel.
; if Excel is already running, then open the two files in that instance, i.e. another Excel instance will NOT be started.
OS.Command start func.csv
WAIT 0.5s
OS.Command start variables.csv
```
**Execute a host command in silent mode**

Is similar to the **OS.Window** command. However, the outputs of the operating system level are discarded. This is suitable for commands that do not require data inputs and whose outputs are not of interest to the user.

**Examples:**

```plaintext
; opens Windows Explorer and selects the file arm9.cmm

; useful when you want to place a PRACTICE script file under version control in a version manager such as SVN

LOCAL &file

&file=OS.FILE.ABSPATH(~/demo/arm/compiler/arm/arm9.cmm)

**OS.Hidden** explorer.exe /select, &file

; opens a hidden shell command window and starts a batch file with two parameters

**OS.Hidden** cmd.exe /C ""D:\my test.bat" "D:\Path To Scripthome\myScript.py" "--signal COMMAND { "path": "MySpecialCommandName" }"

The TRACE32 command is formatted in bold. Host commands are formatted in regular font.

**See also**

- **OS**
- **OS.screen**
- **OS.Window**
- **DIR**
- **OS.ENV()**
- 'Host Commands’ in 'IDE User’s Guide'
If the **OS.screen** command contains an argument, it will immediately be executed by the shell of the host.

If **OS.screen** does not contain any argument, it opens just a system shell. Returning to the TRACE32 system is then dependent on the host. In the case of the Windows shell, the EXIT command is used; in the case of UNIX, CTRL-D will be the standard function key. Before program execution the host interface is deactivated and the terminal and keyboard operating modes are initialized.

In the examples below, the TRACE32 commands are formatted in bold. Host commands are formatted in regular font. The Windows host command `cmd /C (or cmd.exe /C)` is highlighted in red to emphasize its importance for the **OS.screen** command.

**Example 1**

This example shows how to call up the command shell of the host from within TRACE32, run a few host commands, and then return to TRACE32.

<table>
<thead>
<tr>
<th><strong>TRACE32 Command Line</strong></th>
<th><strong>Command Shell of the Host</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>;Call up the command shell</code></td>
<td><code>rem Change from a network drive to the system directory of TRACE32</code></td>
</tr>
<tr>
<td><strong>OS.screen</strong></td>
<td><code>J:\&gt;cd /d C:\T32</code></td>
</tr>
<tr>
<td><code>rem List all PRACTICE script files residing in C:\T32</code></td>
<td><code>C:\T32&gt;dir *.cmm</code></td>
</tr>
<tr>
<td><code>rem Close the command shell</code></td>
<td><code>C:\T32&gt;exit</code></td>
</tr>
<tr>
<td><code>;Continue your TRACE32 session</code></td>
<td><code>...</code></td>
</tr>
</tbody>
</table>

**Example 2 - Start another application from the TRACE32 command line**

`;NOTE: omit the Windows "start" command in case of the OS.screen command

**OS.screen** notepad.exe`
Example 3 - Write file names to a txt file (Windows)

The PER files of TRACE32 reside in the system directory of TRACE32, which is C:\T32 by default for Windows. In this example, all *.per file names are written to a txt file. The resulting txt file is saved to your TEMP folder. The exact folder path depends on the parameter assigned to the environment variable %temp% of your host.

```
;list the *.per files
OS.screen cmd /C dir /b C:\t32\*.per > %temp%\perfilenames_only.txt
```

Example 4 - Write a string to a txt file (Windows)

```
OS.screen cmd.exe /C echo Hello World! > %temp%\file1.txt
```

The TRACE32 commands Data.WRITESTRING and WRITE can also be used to write strings to a file.

Example 5 - Print the path of the Windows environment variable %temp% to the command shell

```
OS.screen cmd /C echo %temp% && pause

;produces the same result as OS.screen above, but display the path of the environment variable %temp% in the OS.Window of TRACE32
OS.Window echo %temp%
```

See also

- OS
- OS.Area
- OS.Command
- OS.Hidden
- "Host Commands" in 'IDE User's Guide'
- "Release Information" in 'Release History'
A TRACE32 window will be generated and then the host command will be executed. All outputs of this host command are re-routed to the TRACE32 window.

To illustrate the **OS.Window** command, the examples below show how to create a directory listing, a tree structure of a directory, and how to list the environment variables of the host within TRACE32.

| Outputs of the operating system may be viewed only. While running under DOS, most terminal-oriented programs do not use the operating system! During program execution nothing is displayed on the terminal. Therefore interactive program entries cannot be carried out. The host interface of the TRACE32 remains in active mode during execution. |

TRACE32 commands are formatted in bold. Host commands are formatted in regular font.

**Example 1 - Directory listing, tree structure, and environment variables (Windows)**

```
OS.Window dir c:\t32
OS.Window tree c:\t32\demo /f /a
OS.Window set
```

**Example 2 - Directory listing and environment variables (Linux)**

```
OS.Window ls -l /home/user/t32
OS.Window env
```

See also

- OS
- OS.Hidden
- OS.screen
- OS.ENV()

▲ 'Host Commands' in 'IDE User's Guide'
PACK

Compress files (with LZW algorithm)

Format:  PACK <source> [<destination>]

The source file is compressed to about 10-60% of the original file size by a Lempel-Ziv-Welch algorithm. The source and destination file names must be different. The PACK command can be used to compress the data files of the analyzer ('.ad' files), or the boot files generated by the dynamic linker (boot00.t32 etc.). If only one argument is supplied, the source file is packed. When opening files, TRACE32 recognizes all packed files automatically.

```
E::PACK ref1.ad ref1.pak ; pack analyzer file
E::Analyzer.LOAD ref1.pak ; un-packing is done automatically
::{PACK \t32\boot00.t32 ; pack boot file
```

See also

- UNPACK
- UNZIP
- ZIP

▲ 'File and Folder Operations’ in 'IDE User’s Guide’
PATCH

Binary file patching

Format: PATCH [filename] [offset] <data> …

Patches bytes in a binary file.

See also

- DUMP
- EDIT
- TYPE

▲ ‘File and Folder Operations’ in 'IDE User’s Guide'
PATH

Define searchpath

Format: PATH [+][<pathname> …]

Defines or modifies the search path for files. This search path is used for some frequent used file formats. The files will be searched first in the current or specified directory.

This command can not be used to search the source files for HLL debugging.

Searching is enabled for the following commands:

<table>
<thead>
<tr>
<th>Extension</th>
<th>Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmm</td>
<td>DO, RUN, PEDIT</td>
</tr>
<tr>
<td>ts</td>
<td>Analyzer.Program, Analyzer.ReProgram</td>
</tr>
<tr>
<td></td>
<td>(TRACE32-ICE, TRACE32-FIRE only)</td>
</tr>
<tr>
<td>ps</td>
<td>PERF.Program, PERF.ReProgram</td>
</tr>
<tr>
<td>per</td>
<td>PERF.Program, PER.ReProgram, PER.view</td>
</tr>
</tbody>
</table>

PATH \t32\exam\cmm \use\me\mycmm ; define two search directories
PATH + ...\cmm ; add one more directory
Using the **PRinTer** command group, you can send every window or the complete screen from TRACE32 to:

- The default printer
- The clipboard
- A file
- The default **AREA** window A000

You can define the format, e.g. font, font size, file type ASCII, enhanced ASCII, XHTML, XML, and HTML for each output medium. When printing to file, you can specify path and file name or browse for an existing file.

You can configure printouts via the TRACE32 command line or via the **PRinTer.select** window:

![PRinTer.select window](image)

For descriptions of the commands in the **PRinTer.select** window, please refer to the **PRinTer.*** commands in this chapter.

Example: For information about the **FILE** option, see **PRinTer.FILE**.

For PRACTICE script examples, see:

- **PRinTer.FILE**
- **PRinTer.OPEN**
- **PRinTer.HardCopy**
- **PRinTer.Area**

See also

- **PRinTer.Area**
- **PRinTer.FILE**
- **PRinTer.OPEN**
- **PRINT**
- **PRinTer.ClipBoard**
- **PRinTer.FileType**
- **PRinTer.PRINT**
- **WinPrint**
- **PRinTer.CLOSE**
- **PRinTer.HardCopy**
- **PRinTer.select**
- **WinPRT**
- **PRinTer.EXPORT**
- **PRinTer.OFFSET**
- **PRinTer.SIZE**

▲ ‘PRINTER Function’ in ‘IDE Functions’
▲ ‘Printer Operations’ in ‘IDE User’s Guide’
Re-directs the printer output to the currently selected AREA window. To specify the window you want to print to the AREA window, use one of the following commands:

- `WinPrint.<command>`
- `WinPRT`
- `PRinTer.HardCopy`

To select an AREA window to which you want to re-route the printer output, use the `AREA.Select` command.

<format> If the parameter <format> is omitted, the format used to print to clipboard stays unchanged.

**ASCII**

Enhanced ASCII format, underlines are displayed, graphic characters are converted and displayed as ASCII characters where feasible.

```
Register.view ;optional step: let's display the window we want
to print
AREA.Select  A000 ;select and display the default AREA window
AREA.view    A000
PRinTer.Area ;instruct TRACE32 to re-route the printer output
to the selected AREA window
WinPrint.Register.view ;print the window
```

Printed to the selected AREA window

**See also**

- PRinTer
- PRinTer.select
PRinTer.ClipBoard  
Re-route printer output to clipboard in specified format

| Format: | PRinTer.ClipBoard [<format>]  
| PRinTer.ClipType (deprecated) |
| <format>: | ASCIE | CSV | XHTML |

Re-directs the printer output to the clipboard. To specify which window you want to print to the clipboard, use **WinPrint.<command>**. For an example, see **PRinTer.HardCopy**.

| <format> | If the parameter <format> is omitted, the format used to print to clipboard stays unchanged. |
| ASCII | Enhanced ASCII file, underlines are displayed, graphic characters are displayed as ASCII characters. |
| CSV | Comma-separated value. |
| XHTML | XML-formatted file with HTML tags. |

### See also
- PRinTer  
- PRinTer.select  
- 'Window System' in 'IDE User's Guide'  
- 'Release Information' in 'Release History'

---

PRinTer.CLOSE  
Close file after multiple printer outputs

| Format: | PRinTer.CLOSE |

The file, opened by the **PRinTer.OPEN** command, is closed. Alternatively, click the close file button in the **PRinTer.select** window.

### See also
- PRinTer  
- PRinTer.OPEN  
- PRinTer.select  
- 'Window System' in 'IDE User's Guide'  
- 'Printer Operations' in 'IDE User's Guide'  
- 'Document your Results’ in ‘Training FIRE Basics’  
- 'Document your Results’ in ‘Training ICE Basics’

©1989-2018 Lauterbach GmbH
PRinTer.EXPORT

Export formatted printer output to file

Format 1:  PRinTer.EXPORT.<file_format> [<filename>] [/Append]

<file_format>:  ASCIIE | CSV | XHTML

Format 2:  default [<filename>] [/Append] (deprecated)

Defines an output file and sets the output format to the specified <file_format>. To actually export a TRACE32 window, use the WinPrint pre-command. It re-directs the window contents to the output file in the format specified with PRinTer.EXPORT.<file_format>.

The output file is opened when executing a print function, and closed immediately after it.

Format 1:

**ASCIIE**

Sets the output format to Enhanced ASCII. Additionally, TRACE32 appends the extension*.txt if you have not specified any extension. You can change the default extension with the command SETUP.EXTension TEXT. Underlines are displayed, graphic characters are displayed as ASCII characters. See example 1.

**CSV**

Sets the output format to CSV (Comma-Separated Values). Additionally, TRACE32 appends the extension*.csv if you have not specified any extension. You can change the default extension with the command SETUP.EXTension CSV. Use the CSV format if you want to import the exported data to other applications.

**XHTML**

Sets the output format to HTML. Additionally, TRACE32 appends the extension*.html if you have not specified any extension. You can change the default extension with the command SETUP.EXTension XHTML. You can set the file extension to *.xml or *.html or *.xhtml depending on how you want the browser to interpret the file. See example 2.

**NOTE:** PRinTer.EXPORT.<file_format> and PRinTer.FILE are rather similar. The minimal difference between the two commands is:

- **PRinTer.EXPORT.<file_format>** automatically adds the file name extension for the selected format in case you have omitted the extension.
- **PRinTer.FILE** supports more (but uncommon) file formats.
<table>
<thead>
<tr>
<th>Format 2:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>default (deprecated)</strong></td>
</tr>
<tr>
<td>Sets the output format to CSV (Comma-Separated Values), but does <strong>not</strong> append the file name extension *.csv automatically. As a result, the exported files do not have an extension - unless you explicitly specify the extension. See example 3.</td>
</tr>
<tr>
<td><strong>NOTE:</strong> If <code>&lt;filename&gt;</code> is omitted, the default file name <code>t32.lst</code> is used.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Options for Format 1 and Format 2:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>&lt;filename&gt;</strong></td>
</tr>
<tr>
<td>In order to simplify multiple file generation, a decimal number contained in the file name (e.g. <code>exam01.csv</code>) is incremented automatically after each print to that file.</td>
</tr>
<tr>
<td>If <code>&lt;filename&gt;</code> is omitted, the printer output gets redirected to the previously chosen output file name (incremented if the file name contained a decimal number). And <code>PRinTer.EXPORT.&lt;file_format&gt;</code> will only append the extension.</td>
</tr>
<tr>
<td><strong>Append</strong></td>
</tr>
<tr>
<td>Use the option <strong>Append</strong>, to append new data to the existing file. Without <strong>Append</strong>, contents are overwritten if the file already exists.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Example 1:</strong> The file name extension omitted by the user is added automatically by TRACE32. Using the <strong>Append</strong> option, three windows are printed to the same file.</td>
</tr>
</tbody>
</table>

```plaintext
;TRACE32 automatically completes the file name with the extension .txt
PRinTer.EXPORT.ASCIIE "~~~\line_tree_var" /Append

;print the first window to the specified file
WinPrint.Trace.STATistic.Line

;append the next two windows to the same file
WinPrint.Trace.STATistic.TREE
WinPrint.Trace.STATistic.Var

;right-click the file in the DIR window to open the file
DIR "~~~\line_tree_var.txt"
```
Example 2:

; in the *.xml file, insert the tag <?xml-stylesheet ...href="..."?>
; with the specified *.xsl file as href="..." attribute
SETUP.XSLTSTYLESHEET "file:///c:/myfiles/mywinprint.xsl"

; export the Register.view and Var.Watch window to the same file
PRinTer.EXPORT.XHTML "c:\t32\win_export.xml" /Append
WinPrint.Register.view
WinPrint.Var.Watch %SpotLight flags ast

; display the file on a browser tab:
OS.Command start firefox "c:\t32\win_export.xml"

; view the XML source code on another browser tab:
WAIT 2.s
OS.Command start firefox "view-source:file:///c:/t32/win_export.xml"

The reference to your XSLT stylesheet is only included if the XSLT stylesheet is explicitly specified with the SETUP.XSLTSTYLESHEET command. The *.xsl file is not created by TRACE32.

Basic formatting provided by TRACE32.

The line with the *.css file name is included for your convenience to allow a user-definable formatting. The *.css file is not created by TRACE32.
Example 3 - regarding the deprecated command PRinTer.EXPORT.default: An output file name with a decimal number is defined. In the next block, three windows are printed to separate files. For each print operation, the decimal number in the file name is incremented.

```plaintext
PRinTer.EXPORT.default "~~~\test-5.csv" ;start with this file name
WinPrint.Trace.STATistic.Line ;print to test-5.csv
WinPrint.Trace.STATistic.TREE ;print to test-6.csv
WinPrint.Trace.STATistic.Var ;print to test-7.csv
DIR "~~~\test-*\.csv" ;list the files in the TRACE32 window.
;right-click to open a file
```

See also

- PRinTer
- PRinTer.FILE
- PRinTer.select
- WinPrint

▲ 'Printer Operations' in 'IDE User's Guide'
▲ 'Release Information' in 'Release History'
PRinTer.FILE

Re-route printer output to a file in specified file format

[Examples]

Format: PRinTer.FILE [<filename>] [<format>] [/Append]

<br />&lt;format&gt;: ASCII | ASCIIP | ASCIIIE | CSV | WS | WSX | PSxxx | XHTML

Re-directs the printer output to a file, which is opened when executing a print function, and closed immediately after it. You can specify the file format together with the file name.

To specify which window you want to print to file, use WinPrint.<command>, as shown in the PRACTICE script examples below.

<table>
<thead>
<tr>
<th>&lt;filename&gt;</th>
<th>In order to simplify multiple file generation, a decimal number contained in the file name (e.g. exam00.lst) is incremented automatically. If &lt;filename&gt; is omitted, the printer output is redirected to the previously chosen output file name (incremented if the file name contained a decimal number).</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;format&gt;</td>
<td>If &lt;format&gt; is omitted, the format used to print to file stays unchanged.</td>
</tr>
<tr>
<td>Append</td>
<td>Use the option Append to append new data to the existing file. Without Append, file contents are overwritten if the file already exists.</td>
</tr>
</tbody>
</table>

| ASCII | Pure ASCII file format. All non-ASCII characters are displayed as an '*'. The output is packed without paging. |
| ASCIIP | Same as ASCII, but paged output format with fixed line length. |
| ASCIIIE | Enhanced ASCII file, underlines are displayed, graphic characters are displayed as ASCII characters. |
| CSV | Comma-separated value. |
| WS | Wordstar compatible output format. |
| WSX | Special Wordstar format. |
| PSxxx | POSTSCRIPT output format. Different resolutions, orientations and fonts are available. The output styles are defined in the prolog file for postscript. The prolog file ('t32pro.ps') is searched on the current directory and the system directory. For more information, see below. |
| XHTML | XML-formatted file with HTML tags. |
| NOTE: In the PRinTer.FILE command, where you specify the file name, set the file extension to *.xml or *.html or *.xhtml depending on how you want the browser to interpret the file. See example 2 and example 3. |

©1989-2018 Lauterbach GmbH
Example 1

**Data.dump** windows are printed to separate files in ASCII format.

```
PRinTer.FILE ~~/exam00.lst ASCII ; choose output file name and format
WinPrint.Data.dump 0x100--0x1ff     ; print window to exam00.lst
WinPrint.Data.dump 0x200--0x2ff     ; print window to exam01.lst
WinPrint.Data.dump 0x300--0x3ff     ; print window to exam02.lst
PRinTer.FILE , PSPS12               ; print window to exam03.lst in
WinPrint.Data.dump 0x400--0x4ff     ; POSTSCRIPT format
```

Example 2

**List.Mix** windows are printed to separate files in HTML format.

```
PRinTer.FILE ~~/file01.html XHTML ; choose output file name and format
WinPrint.List.Mix func7--func17    ; print window to file01.html
WinPrint.List.Mix func18--func25    ; print window to file02.html

; display the files on two tabs in a browser:
OS.Command start firefox c:\temp\file01.html c:\temp\file02.html

; view the source on a third browser tab:
WAIT 2.s
OS.Command start firefox "view-source:file:///c:/temp/file01.html"
```

A Basic formatting provided by TRACE32.

B The line with the *.css file name is included for your convenience to allow a user-definable formatting. The *.css file is **not** created by TRACE32.
Example 3

List.Mix windows are exported to separate XML files, and each XML file contains a reference to a user-defined XSLT stylesheet.

;in the *.xml file, insert the tag <?xml-stylesheet ...href="..."?> with the specified *.xsl file as href="..." attribute
SETUP.XSLTSTYLESHEET "file:///c:/myfiles/mywinprint.xsl"

PRinTer.FILE c:\t32\file01.xml XHTML ;choose output file name and format
WinPrint.List.Mix func7--func17 ;print window to file01.xml
WinPrint.List.Mix func18--func25 ;print window to file02.xml

;display the files on two tabs in a browser:
OS.Command start firefox c:\t32\file01.xml c:\t32\file02.xml

;view the source on the third browser tab:
WAIT 2.s
OS.Command start firefox "view-source:file:///c:/t32/file01.xml"

A The reference to your XSLT stylesheet is only included if the XSLT stylesheet is explicitly specified with the SETUP.XSLTSTYLESHEET command. The *.xsl file is not created by TRACE32.
The style of POSTSCRIPT outputs can be widely varied by modifying the prolog file 't32pro.ps'. This file is prepended to all POSTSCRIPT outputs send to a file or to a printer. The file also contains the definitions of printout formats made available to TRACE32. New printer formats, extra page headers or other fonts can be added by modifying this file. The produced POSTSCRIPT files can use as encapsulated postscript files to include them in documentations produced by desktop publishing software.

For information on POSTSCRIPT:

- Adobe Systems Inc.  
  Addison Wesley 1991,  
  ISBN 0-201-18127-4

- Adobe Systems Inc.  
  Postscript Language Tutorial and Cookbook,  
  Addison Wesly 1985  
  ISBN 0-201-10179-3

  Emphasizes examples to illustrate the many capabilities of the PostScript language. Should give enough information to make your own prologue.

See also

- PRinTer  
- PRinTer.EXPRT  
- PRinTer.OPEN  
- PRinTer.select

- SETUP:XSLTSTYLESHEET  
- WinPrint

- 'Printer Operations’ in 'IDE User's Guide’
- 'Window System’ in 'IDE User's Guide’
- 'Release Information’ in 'Release History’

PRinTer.FileType

Select file format

Format: PRinTer.FileType [<format>] (deprecated)

Deprecated command. Set file format with the commands PRinTer.FILE or PRinTer.OPEN or PRinTer.EXPRT instead.

See also

- PRinTer  
- PRinTer.select

- 'Release Information’ in 'Release History’
- 'Document your Results’ in 'Training FIRE Basics’
- 'Document your Results’ in 'Training ICE Basics’
PRinTer.HardCopy

Make a hardcopy of the screen

Prints the full screen layout.

The following example is for demo purposes only. It provides an overview of how to use the PRinTer.HardCopy command to send a window from TRACE32 to:

- The default printer
- The clipboard
- A file

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

```c
;Let's open and print a window and a dialog for demo purposes
Register.view ;open the Register.view window
PRinTer.select ;open the PRinTer dialog with the current
                ;TRACE32 printer settings

;output to printer
PRinTer WINC12 ;select printer, font and size: Windows Courier 10pt
PRinTer.HardCopy ;send hardcopy to your printer (or click Cancel)

;output to clipboard
PRinTer.ClipBoard ASCIIIE ;select the clipboard with format ASCIIIE
PRinTer.HardCopy ;send hardcopy to your clipboard

;output to file
PRinTer.FILE C:\temp\t32.lst ASCIIIE ;specify file path and format
PRinTer.HardCopy ;send hardcopy to specified file
```

See also

- PRinTer
- PRinTer.select
- WinPRT

▲ 'Printer Operations' in 'IDE User's Guide'
▲ 'Window System' in 'IDE User's Guide'
**PRinTer.OFFSET**

Specify print-out borders

<table>
<thead>
<tr>
<th>Format:</th>
<th><strong>PRinTer.OFFSET</strong> [&lt;columns&gt;] [&lt;lines&gt;]</th>
</tr>
</thead>
</table>

This command is used to adjust the position of the print-out on the paper. It is very useful to leave a white margin on the left side of the page. The size of the print-out must be changed accordingly.

```plaintext
PRinTer.select LjL          ; choose printer
PRinTer.OFFSET 12.          ; leave space for perforation
PRinTer.SIZE 80.            ; adjust printout size, make it smaller
WinPrint.HELP Data.dump     ; print chapter of manual
```

See also

- **PRinTer**
- **PRinTer.select**
- **WinPrint**

▲ ‘Printer Operations’ in ‘IDE User’s Guide’
PRinTer.OPEN

Re-route multiple printer outputs to the same file

Format:  PRinTer.OPEN  [<filename>]  [<format>]  [/Append]

Redirects all printer output generated with the WinPrint pre-command to the same file. You can specify the file format together with the file name.

Use PRinTer.CLOSE to close the file and end the output redirection.

- **<filename>**  
  - If <filename> is omitted, the default file name t32.lst is used.  
  - If the specified file already exists, it will be overwritten by default.

- **<format>**  
  - If <format> is omitted, the format used to print to file stays unchanged.  
  - For a list of available file formats, see command PRinTer.FILE.

- **Append**  
  - Use the option Append to append new data to the existing file.  
  - Without Append, file contents are overwritten if the file already exists.

**NOTE:** Only one file can be open at a time. The message line displays an error if you run the PRinTer.OPEN command again without having closed the open file.

In case of an error, open the PRinTer.select window, and then click the close file button, or run the PRinTer.CLOSE command.
Example 1

The following example is for demo purposes only. The contents of the **List** window and the **sYmbol.Browse.Function** window are printed to file. Then the file is opened in TRACE32.

```plaintext
PRinTer.select
PRinTer.OPEN "~~~/myfile.txt" ASCIIE ;create and open a file for
;writing in ASCIIE file format

WinPrint.List main ;WinPrint.* prints the contents of
WinPrint.sYmbol.Browse.Function ;the two windows to file

PRinTer.CLOSE ;close the file

TYPE "~~~/myfile.txt" ;show the resulting file
```

Example 2

Some commands require some processing time before the result is complete, like **Trace.STATistic** or **Trace.Chart**. The command **SCREEN.WAIT** will ensure that processing of the window has completed before script execution continues.

```plaintext
LOCAL &cmd
&cmd="Trace.STATistic.sYmbol" ;assign a command to a macro

&cmd ;issue the command to open window
SCREEN.WAIT ;wait until processing completed

PRinTer.OPEN "~~~/myfile.txt" ASCIIE ;create and open a file for
;writing in ASCIIE file format

WinPrint.&cmd ;WinPrint.* prints the contents of
;the completed window to the file

PRinTer.CLOSE ;close the file

TYPE "~~~/myfile.txt" ;show the resulting file
```

The path prefix `~~~` expands to the temporary directory of TRACE32.

**See also**
- **PRinTer**
- **PRinTer.CLOSE**
- **PRinTer.FILE**
- **PRinTer.select**
- **SCREEN.WAIT**
- **WinPrint**

▲ 'Printer Operations' in 'IDE User’s Guide'
▲ 'Window System' in 'IDE User’s Guide'
▲ 'Document your Results' in 'Training FIRE Basics'
▲ 'Document your Results' in 'Training ICE Basics'

©1989-2018 Lauterbach GmbH
PRinTer.PRINT

Print to opened printer file

Format: PRinTer.PRINT [<format>] <data>

<format>: ASCII | BINary | Decimal | Hex | String

Writes the specified data to the file selected with PRinTer.OPEN. Use this command to store additional information to the printed windows.

Example: A time-stamp is printed at the beginning of the file, and then two windows are printed to file.

```
PRinTer.OPEN "~~~/myfile.txt" ASCIIE ;create and open a file for writing in ASCIIE file format
PRinTer.PRINT "
PRinTer.PRINT " Time: "+CLOCK.TIME() ;print time-stamp to file
PRinTer.PRINT ""
WinPrint.Register ;WinPrint.* prints the contents of
WinPrint.List ;these two windows to file
PRinTer.CLOSE ;close the file
TYPE "~~~/myfile.txt" ;show the resulting file
```

See also
- PRinTer
- PRinTer.select
**PRinTer.select**

Selects a physical printer or opens the **PRinTer.select** window, where you can configure all printing options.

- **With argument:** If the command is used with the `<printer>` parameter, all further printing is redirected to the specified physical printer.

- **Without argument:** If the command is used without the `<printer>` parameter, the **PRinTer.select** window is displayed. In this window, you can choose whether you want to send the printout to a printer, to the clipboard, to a file, or to an AREA window. You can define the format, e.g. font, font size, ASCII, enhanced ASCII for each output medium. You can specify path and file name or browse for an existing file.

To specify which TRACE32 window you want to send to the printer or to the clipboard or to file, use **WinPrint**, as shown in the **PRinTer.FILE** example. Printers must be configured in the host system to appear in the **printer** drop-down list of the **PRinTer.select** window.

A For descriptions of the commands in the **PRinTer.select** window, please refer to the **PRinTer.** commands in this chapter. Example: For information about FILE, see **PRinTer.FILE**.

B When is the close file button visible?
- It becomes visible after a file has been opened with **PRinTer.OPEN**. The button remains visible until you close the file again (a) by clicking the close file button or (b) by running the **PRinTer.CLOSE** command.
- It is hidden if a file is opened with **PRinTer.FILE** because **TRACE32 automatically closes** that file again after the print operation.

C If you select a list entry with a file name extension, then the extension is automatically appended to the file name. In all other cases, you can define your own extension or omit the extension.

**See also**

- **PRinTer**
- **PRinTer.EXPORT**
- **PRinTer.OFFSET**
- **PRinTer.Area**
- **PRinTer/File**
- **PRinTer.OPEN**
- **PRinTer.ClipBoard**
- **PRinTer/FileType**
- **PRinTer.CLOSE**
- **PRinTer/HardCopy**
- **PRinTer/SIZE**

▲ 'Window System' in 'IDE User's Guide'
▲ 'Printer Operations' in 'IDE User's Guide'
PRinTer.SIZE

Specify print-out size

Format:  

PRinTer.SIZE [<columns>] [<lines>]

This command is used to adjust the size of the print-out to the parameters of the printer. If a file is selected as output, the lines value can be set to zero, to switch to a non-paged file structure. A column value of zero causes a packed file, i.e. trailing blanks are cut.

```
PRinTer.SIZE 70. 50. ; make printer-output smaller
PRinTer.OFFSET 10. 5. ; shift output to get space for headers
PRinTer.FILE list1 ; redirect output to file
PRinTer.SIZE 0. 0. ; output without paging and without ; trailing blanks
```

See also

- PRinTer
- PRinTer.select
- ‘Printer Operations’ in ‘IDE User’s Guide’
### PWD

**Change directory**

If used *without* `<pathname>`, **PWD** displays the current working directory in the TRACE32 message bar.

If used *with* `<pathname>`, **PWD** changes the working directory as specified in `<pathname>` and displays the new working directory in the TRACE32 message bar.

On Windows environments, the drive may be selected too. When used as a command prefix, the directory is changed to the path used in the command line (implicit change).

#### Example 1:

```
PWD /t32 ; change directory
```

#### Example 2:

```
; set the working directory to c:\t32
PWD c:\t32

; export the function nesting to a *.csv file in the working directory
Trace.EXPORT.CSVFunc func.csv

; export the variables 'flags' and 'ast' to a *.csv file in the working directory
Var.EXPORT variables.csv %Type %Location %Index flags ast

; start a new Excel instance and open the two *.csv files in the new Excel instance
OS.Command start excel.exe func.csv variables.csv
```

---

**See also**

- ChDir
- OS.DIR()
- DIR
- OS.PresentWorkingDirectory()

- 'Commands' in 'IDE User's Guide'
- 'File and Folder Operations' in 'IDE User's Guide'
QUIT

Return to operating system

<table>
<thead>
<tr>
<th>Format:</th>
<th><strong>QUIT</strong> [&lt;os_return&gt;]</th>
</tr>
</thead>
</table>

Closes TRACE32.

After executing **QUIT**, all settings and memory contents are lost! If a continuation of the same setting is wanted, the saving via the **STORE** command will be necessary.

With **SETUP.QUITDO** you can define a PRACTICE script (*.cmm) which will be executed before TRACE32 quits.

Example for Unix/Cygwin to use the `<os_return>` value in a script:

```sh
./t32marm
echo $?
```

Example for Windows to use the `<os_return>` value in a batch file:

```cmd
start "" /wait t32marm.exe
echo %ERRORLEVEL%
```

See also
- **ABORT**
- **SETUP.QUITDO**
- ‘Program End’ in ‘IDE User’s Guide’
REN

Renames a file. This command is not possible with wildcard characters in the file name.

See also

- MV
- ‘File and Folder Operations’ in 'IDE User's Guide'
RM, RMDIR

RM

Delete file

Format: \textbf{RM} \texttt{<filename>}

Removes a file. Wildcard characters within the filename will open the browser for selecting a file.

\texttt{RM c:\t32\test.bak}

\textbf{See also}

\begin{itemize}
\item \texttt{DEL}
\item ‘File and Folder Operations’ in ‘IDE User’s Guide’
\end{itemize}

RMDIR

Remove directory

Format: \textbf{RMDIR} \texttt{<pathname>}

Removes a sub-directory. The directory must be empty.

\textbf{See also}

\begin{itemize}
\item \texttt{MKDIR}
\item ‘File and Folder Operations’ in ‘IDE User’s Guide’
\end{itemize}
SCreenShot

Save a screenshot of a window to a file

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;image format&gt;:</td>
<td>BMP</td>
</tr>
</tbody>
</table>

Captures a screenshot of the whole user interface or a single window and saves the captured image with a selected image format to a file. The default image type is the BMP format. If the file name contains any wildcards, a file-save dialog opens. File names containing any space characters must be enclosed in quotation marks.

As of build no. 86141 (July 2017), the behavior of the command has changed: It now displays an error message in the TRACE32 message line if the specified <windowname> does not exist.

**NOTE:** This command is currently only available if running under Microsoft Windows operating system. Depending on the used OS version, some image formats may not be available.

- **<imageformat>**
  - **BMP:** Windows Bitmap format (default, lossless, uncompressed).
  - **TIFF:** Tagged Image File Format (lossless, uncompressed).
  - **PNG:** Portable Network Graphics format (lossless, compressed).
  - **JPG:** JPEG File Interchange format (lossy, compressed).
  - **GIF:** Graphics Interchange Format (lossy, compressed, 256 colors). Screenshots saved in GIF format are stored with a 256 color lookup table and are dithered to this fixed palette. This reduces the image quality. For higher quality images choose one of the other available formats.

- **<windowname>**
  - Use the WinPOS command to assign a name to built-in windows or built-in window-style dialogs.
  - Use NAME to assign a name to a custom dialog.
  - Window names are case-sensitive. That is, the window names w001 and W001 are not the same.

- **ACTive**
  - Captures a screenshot of the topmost window in the z-order. You can bring a window to the top of the z-order by using the WinTOP command or by clicking inside the desired window. Windows having the window pre-command WinExt are not captured.
; Capture a screenshot of the main window and save in JPEG format:
SCreenShot "~~~/screenshot.jpeg" JPG

; Capture a screenshot of window named W001 and save in GIF format:
WinPOS ,,,,,,W001
SYStem.state
WAIT 200.ms
SCreenShot "~~~/screenshot.gif" GIF W001

; Open the SYStem.CONFIG window and capture a screenshot of it:
WinPOS ,,,,,,myWin
SYStem.CONFIG
SCreenShot "~~~/screenshot.png" PNG myWin
**SETUP**

**Setup commands (part 1 of 2)**

Using the **SETUP** command group, many window system and user interface parameters can be modified, and rarely-used system functions can be executed.

For additional **SETUP** commands, refer to the **SETUP** commands in "General Commands Reference Guide S" (general_ref_s.pdf).

### See also

- SETUP.ASCIITEXT
- SETUP.CClear
- SETUP.DEVNAME
- SETUP.DITEXT
- SETUP.HOLDDIR
- SETUP.INTERCOMACKTIMEOUT
- SETUP.PDFViewer
- SETUP.RADIX
- SETUP.ReDraw
- SETUP.SOUND
- SETUP.STORE
- SETUP.TIMEFORM
- SETUP.WARNSTOP

▲ ‘SETUP’ in ‘General Commands Reference Guide S’
Configure ASCII text display

Format: \texttt{SETUP.ASCIITEXT FULL8 | FULL7 | PART8 | PART7 | UTF-8}

Configures the display mode for all non-standard characters in dump windows.

- **FULL8**: All 8 bits are used for display. Non-standard characters are displayed in graphic mode.
- **FULL7**: Only 7 bits are used for display. Non-standard characters are displayed in graphic mode.
- **PART8**: All 8 bits are used for display. Non-standard characters are not displayed.
- **PART7**: Only 7 bits are used for display. Non-standard characters are not displayed.
- **UTF-8**: Support for UTF-8 characters in the Data.dump and Var.View windows.

```
\texttt{E::Data.dump /Byte /NoHex}

\begin{verbatim}
E::Data.dump /Byte /NoHex
0123456789ABCDEF0123456789ABCDEF
................................
!"#$%&'()*/+,-./0123456789:;<=>?
\@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_ `abcdefghijklmnopqrstuvwxyz{|}~.
................................
E::SETUP.ASCIITEXT PART7
\end{verbatim}

```

See also
- \texttt{SETUP}
- Data.dump
- ‘System Setup and Configuration’ in 'IDE User's Guide'
- ‘Release Information’ in 'Release History'
### SETUP.BAKfile

**Set backup file mode**

| Format: | SETUP.BAKfile [ON | OFF] |

Default: ON.

If active, all 'save' commands will maintain a copy of the old version of the program. This will prevent the loss of data, if a wrong 'save' command is issued. If the option is turned off, the 'backup' files are only generated by the text editor.

Editor commands did not generate a backup file, if the name for the file extension is deleted (see `SETUP.EXT BAK`).

**See also**

- SETUP
- 'System Setup and Configuration' in 'IDE User's Guide'

### SETUP.CClear

**Erase caches**

| Format: | SETUP.CClear |

This command is used, whenever host file time stamps are incorrect. All TRACE32 system file cache buffers are cleared.

**See also**

- SETUP
- 'System Setup and Configuration' in 'IDE User's Guide'
SETUP.COLOR

Change colors

Format:

```
SETUP.COLOR [object] <red> <green> <blue>
```

If the command is entered without parameters, the **SETUP.COLOR** window is opened.

A  **<object>** identifier column.

B  Click **change** to modify a color.

C  Scroll down to view the colors that can be applied to TRACE32 windows for multicore debugging. For example, if green stands for core 1, then information from core 1 will be displayed in windows with a green window background.

Under Windows or LINUX and QT only the lower 8 bits of the 16 bit values are significant (0xff80 is handled like 0x0080).

```
SETUP.COLOR 40. 0xff 0x14 0x93 ; Change the Info Message
               ; Background (40.) to DeepPink
```

Under UNIX and MOTIF the full 16 bit value range is used e.g. DeepPink (FF 14 93) results as command:

```
SETUP.COLOR 40. 0xff00 0x1400 0x9300 ; Change the Info Message
                                           ; Background (40.) to DeepPink
                                           ; (RGB values: FF 14 93)
```

For Unix derivates the X11 color values, see [http://en.wikipedia.org/wiki/web_colors](http://en.wikipedia.org/wiki/web_colors)

The command modifies the TRACE32 configuration file too. (e.g. config.t32)

```
SCREEN=
PALETTE 40. = 0xff 0x14 0x93 ; Change the Info Message
                                ; Background (40.) to DeepPink
```

An additional PRACTICE script is included in your TRACE32 installation. To access the script, run this command:

```
B::CD.PSTEP ~/demo/practice/colors/presentation.cmm
```

See also

- SETUP
- sYmbol.ColorDef
- sYmbol.List.ColorDef
- FramePOS
- CORE.SHOWACTIVE

► ‘Screen Display’ in ‘IDE User’s Guide’
► ‘System Setup and Configuration’ in ‘IDE User’s Guide’
SETUP.DEVNAME

Set logical device name

Format:  

<table>
<thead>
<tr>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>SETUP.DEVNAME [sysname]</td>
</tr>
</tbody>
</table>

Defines an new device name for the selected device. This command is used when more than one device is used in a debug environment, e.g. multicore debugging.

```
B::SETUP.DEVNAME JTAG1 ; sets new name
JTAG1:: ; from now the device name is "JTAG1"
JTAG1::Data.List main ; next command
```

See also

- SETUP
- 'System Setup and Configuration' in 'IDE User's Guide'
The command without parameters displays a list of the loaded drivers. With parameters, the command can reconfigure a loaded driver. The valid driver commands are similar to the file `config.t32`.

```plaintext
;Instruct the Windows Screen driver to change the window background color to orange. Note the following for this command:
; - the individual PALETTE arguments must be separated by only one space
; - one or more trailing spaces are NOT permissible
SETUP.DRV 4 PALETTE 01 = 255 128 64

;recommended: use the SETUP.COLOR command to open the color configuration window
SETUP.COLOR

;set the window background, i.e. element 01, to yellow
;the periods instruct TRACE32 to interpret the values as decimal values
;and not as hex values
SETUP.COLOR 01. 225. 255. 128.

;restore the window background to its default color
SETUP.COLOR 01.

;change the caption of the TRACE32 main window
SETUP.DRV 4 HEADER=TRACE32 Project X

;recommended: use the TITLE command
TITLE "TRACE32 myProject X"

SETUP.DRV 8 DEV=xfile
```

See also
- SETUP
- ‘Screen Display’ in 'IDE User's Guide'
- ‘System Setup and Configuration’ in 'IDE User's Guide'
**SETUP.EDITEXT**

Define an external editor

<table>
<thead>
<tr>
<th>Format:</th>
<th>SETUP.EDITEXT ON</th>
<th>OFF [commandline]</th>
</tr>
</thead>
</table>

Replaces the internal editor call with an external editor call.

<table>
<thead>
<tr>
<th>ON</th>
<th>The <code>EDIT.file</code> command starts your external editor - instead of the built-in editor. See <a href="#">Example 1</a>.</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFF</td>
<td>The external editor is only started when you execute the <code>EDIT.EXTern</code> command. See <a href="#">Example 2</a>.</td>
</tr>
<tr>
<td>&lt;commandline&gt;</td>
<td>This string contains the command that TRACE32 sends to your OS to start the external editor. In this string, the following replacements will be made:</td>
</tr>
<tr>
<td></td>
<td>- <code>*</code> will be replaced by the actual file name.</td>
</tr>
<tr>
<td></td>
<td>- <code>#</code> will be replaced by the actual line number.</td>
</tr>
</tbody>
</table>

[Example 1](#) shows how to configure TextPad (or JEDIT or UltraEdit) as an external editor for TRACE32 PowerView with the **ON** setting:

```plaintext
;configure TextPad as an external editor
SETUP.EDITEXT ON "C:\Program Files\TextPad 5\TextPad.exe ""* (#)"
```

```plaintext
;configure JEDIT as an external editor
;SETUP.EDITEXT ON "C:\eclipse\jedit5.0.0\jedit.exe ""*"" +line:#
```

```plaintext
; configure UltraEdit as an external editor
;SETUP.EDITEXT ON "C:\IDM\UltraEdit\uedit32.exe ""**"
```

```plaintext
; PRACTICE script file opens in the external editor
EDIT.file ~/my-script.cmm
```
Example 2 shows how to configure TextPad as an external editor for TRACE32 PowerView with the OFF setting:

```
; configure an external editor
SETUP.EDITEXT OFF "C:\Program Files\TextPad 5\TextPad.exe ""* (")"

; Text file opens in the built-in editor as usual
EDIT.file ~/mylog.txt

; Text file now opens in the external editor
EDIT.EXTern ~/mylog.txt
```

See also

- SETUP
- DIR
- EDIT:EXTern
- EDIT.file

▲ 'File and Folder Operations' in 'IDE User's Guide'
▲ 'External Editors and Syntax Highlighting' in 'PRACTICE Script Language User's Guide'
▲ 'Release Information' in 'Release History'
SETUP.EXTension

Set default file name extensions

<table>
<thead>
<tr>
<th>Format:</th>
<th>SETUP.EXTension &lt;type&gt; [&lt;extension&gt;]</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;type&gt;:</td>
<td>AL</td>
</tr>
</tbody>
</table>

Usually for every type of file, a file name extension is specified. The name of this extension could be changed. The command

E::DO p1
executes the PRACTICE script 'p1.cmm' whenever the 'DO' default extension is set to '.cmm'. The default settings are indicated by brackets.

AL (.ad) Extension for the A.Load and A.Save commands.

AP (.ts) Extension for the Analyzer.Program and Analyzer.ReProgram commands.


BAK (.bak) Extension for all backup files.

BNK (.bnk) Extension for the SYStem.BankFile command.

BSDL (.bsdl) Extension for boundary scan description files.

COV (.acd) Extension for the coverage database.

CSV (.csv) Extension for CSV formatted files.

DIALOG (.dlg) Extension for dialog description files.

DO (.cmm) Extension for the DO and PEDIT commands.

ELF (.elf) Extension for executable and linking format files.

LOG (.log) Extension for the LOG.OPEN command.

LUA (.lua) Extension for LUA scripts.

MENU (.men) Extension for TRACE32 menu description files.

ORTI (.ort*) Extension for the OSEK run-time interface.

OS () Extension for TYPE and EDIT commands.

PER (.per) Extension for all PER commands.

PERF (.ps) Extension for the PERF.Program and PERF.ReProgram commands.

PRT (.lst) Extension for the PRinTer.OPEN and PRinTer.FILE commands.

STORe (.cmm) Extension for the STORe command.

TAD (.tad) Extension for the A.Load and A.Save commands for ICE TA32 module only.

**TEXT (.txt)**
Extension for plain text files.

**XHTML (.html)**
Extension for files formatted in the extensible hypertext markup language.

---

**SETUP.HOLDDIR**
Configure working directory

Format:
```
SETUP.HOLDDIR [ON | OFF]
```

Default: OFF.

When switched to OFF, the working directory of the TRACE32 system can change, if an operating system command or `ABORT` will be executed. Otherwise the working directory can be changed by the command `ChDir` only.

---

See also
- SETUP
- ‘System Setup and Configuration’ in ‘IDE User’s Guide’

---

**SETUP.ICONS**
Display icons in popup menus

Format:
```
SETUP.ICONS [ON | OFF]
```

Default: ON.

**SETUP.ICONS** without argument toggles the icons in the popup menus.

- **ON**
  - Displays the icons in the popup menus (also referred to as context menus).

- **OFF**
  - Switches the icons off.

---

See also
- SETUP
- SOFTKEYS
- STATUSBAR
- SUBTITLE
- TITLE
- TOOLBAR

---

©1989-2018 Lauterbach GmbH

IDE Reference Guide 225
**SETUP.INTERCOMACKTIMEOUT**  
Sets the intercom acknowledge timeout

Format:  
**SETUP.INTERCOMACKTIMEOUT** <time> | <value>

Using this command, you can increase the default intercom acknowledge timeout from 500 milliseconds to a maximum of 5 seconds.

<time>  
You can specify the time in milliseconds or in seconds.  
The minimum is **500.ms**  
The maximum is **5.s**

<value>  
Without time specification (i.e. without .ms or .s), the value is interpreted to mean milliseconds.

```plaintext
SETUP.INTERCOMACKTIMEOUT 5.s ; Increase timeout to maximum
```

See also
- SETUP
- INTERCOM

**SETUP.MEMory**  
Display free and used SCU memory

Format:  
**SETUP.MEMory**

Displays the amount of free and used SCU memory in the TRACE32 message line.

See also
- SETUP
- ‘System Architecture TRACE32-ICE’ in ‘Training ICE Basics’
Opens the **SETUP.PDFViewer** dialog window, where you can configure TRACE32 to context-sensitively display the *.pdf files of the help system in your favorite PDF viewer.

Configuration takes only a few mouse-clicks. In addition, you do not need to re-start TRACE32 because your settings take immediate effect. Your settings are stored in the TRACE32 user preferences and re-read on the next start-up of TRACE32.

**A** Click the **DETECT** button to detect your default PDF viewer.
The remaining input boxes are automatically populated with the command line parameters for the selected PDF viewer. (The command line parameters are pre-configured in TRACE32.)

**B** Alternatively, click **browse** to browse for the PDF viewer you want to use. Then click the **preset** button.
The remaining input boxes are automatically populated with the command line parameters for the selected PDF viewer. (The command line parameters are pre-configured in TRACE32.)

**C** The **test** buttons allow you to immediately test the configuration suggested by the **SETUP.PDFViewer** dialog window.

TRACE32 provides pre-configured parameters for well-known PDF viewers on Windows and Linux in order to reduce the configuration effort for users to a few mouse-clicks.

**See also**
- **SETUP**
- **HELP**
- 'Release Information' in 'Release History'
SETUP.PDFViewer.EXEcutable  Path and executable of your PDF viewer

<table>
<thead>
<tr>
<th>Format:</th>
<th>SETUP.PDFViewer.EXEcutable &lt;executable&gt;</th>
</tr>
</thead>
</table>

Sets up the PDF executable which is called to open the PDF files of the TRACE32 help system.

This command is only used for scripting and corresponds to the EXEcutable input box in the SETUP.PDFViewer dialog window.

SETUP.PDFViewer.OPEN  Open a PDF of the help system

<table>
<thead>
<tr>
<th>Format:</th>
<th>SETUP.PDFViewer.OPEN.&lt;subcmd&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;subcmd&gt;:</td>
<td>DOCument</td>
</tr>
</tbody>
</table>

DOCument  
Set up the command line parameters for the executable to open a PDF on the first page.

This command is only used for scripting and corresponds to the OPEN.DOCument input box in the SETUP.PDFViewer dialog window.

NamedDest  
Set up the command line parameters for the executable to open a PDF at a named destination.

This command is only used for scripting and corresponds to the OPEN.NamedDest input box in the SETUP.PDFViewer dialog window.

SETUP.PDFViewer.PRinT  Print PDF via HELP window

<table>
<thead>
<tr>
<th>Format:</th>
<th>SETUP.PDFViewer.PRinT</th>
</tr>
</thead>
</table>

Sets up the command line parameters for the executable to open a PDF file and start printing it.

This command is only used for scripting and corresponds to the PRinT input box in the SETUP.PDFViewer dialog window.
Resets the settings in the *SETUP.PDFViewer* dialog window. However, the settings continue to remain active for the current TRACE32 session. As soon as the TRACE32 session is closed, the settings are also cleaned from the TRACE32 user preferences.

This command is only used for scripting and corresponds to the *RESet* button in the *SETUP.PDFViewer* dialog window.

**NOTE:** As long as no PDF viewer is configured for the TRACE32 help system, TRACE32 tries to access the PDF files through one of the two methods from the previous releases. See “*Previous TRACE32 Releases*” (ide_user.pdf).
The `SETUP.PDFViewer.TEMPorary` command group is only used for internal and support purposes.

### SETUP.PDFViewer.TEMPorary.EXEcutable

**PDF viewer for demo purposes**

Format: `SETUP.PDFViewer.TEMPorary.EXEcutable`

Same meaning as `SETUP.PDFViewer.EXEcutable` but nothing is stored in the user preferences.

### SETUP.PDFViewer.TEMPorary.OPEN

**Open a PDF of the help system**

Format: `SETUP.PDFViewer.TEMPorary.OPEN.<subcmd>`

<subcmd>: `DOCument` | `NamedDest`

**DOCument**

Open PDF on the first page; same meaning as `SETUP.PDFViewer.OPEN.DOCument` but nothing is stored in the user preferences.

**NamedDest**

Jump to named destination in PDF; same meaning as `SETUP.PDFViewer.OPEN.NamedDest` but nothing is stored in the user preferences.

### SETUP.PDFViewer.TEMPorary.PRinT

**Print PDF via HELP window**

Format: `SETUP.PDFViewer.TEMPorary.PRinT`

Same meaning as `SETUP.PDFViewer.PRinT` but nothing is stored in the user preferences.
Format: \texttt{SETUP.PDFViewer.TEMPorary.RESet}

Same meaning as \texttt{SETUP.PDFViewer.RESet} but nothing is stored in the user preferences.
SETUP.QUITDO

Define quit PRACTICE script file

<table>
<thead>
<tr>
<th>Format: SETUP.QUITDO [&lt;filename&gt;]</th>
</tr>
</thead>
</table>

Defines a PRACTICE script, which is called when leaving the TRACE32 system. The PRACTICE script file must end with a QUIT command to really quit the system. The PRACTICE script can be used to automatically save session settings. The command is typically placed in the TRACE32 start-up script t32.cmm.

The following script is an example of a PRACTICE file called by SETUP.QUITDO.

```plaintext
DIALOG.YESNO "Save window positions?"

LOCAL &answer
ENTRY &answer
IF &answer==TRUE()
    STOre ~/windows.cmm Win ; save file in the
    ; home directory of the user
QUIT
```

See also

- SETUP
- STOre
- QUIT
- STOre

▲ 'System Setup and Configuration' in 'IDE User's Guide'

©1989-2018 Lauterbach GmbH

IDE Reference Guide 232
The radix mode (number base) is specified by this option. Numbers without type prefix like “0X” or “0Y” respectively postfix “.” are interpreted in the selected number base.

### Formats

- **Classic**
  - Number base is hex - all input formats for operands and operators permitted.

- **Decimal**
  - Number base is decimal - old-fashioned operators and operands are locked.

- **Hex (default)**
  - Number base is hex - old-fashioned operators and operands are locked - default.

If `SETUP.RADIX.` is entered at the command line, the currently used RADIX mode is displayed in the message line.

The preferred location for a different radix mode (not the default hex mode) is the user’s own start-up script.

**See also**

- SETUP
- RADIX()
## SETUP.RANDOM

Set seed for RANDOM() function

Format: `SETUP.RANDOM [<seed>]`

Sets a seed value for the internal pseudo random number generator. `<seed>` is an unsigned 64bit number. If `<seed>` is skipped, the current system timer is used to define an arbitrary seed number. The seed value affects the pseudo random number sequence delivered by the PRACTICE functions `RANDOM()` and `RANDOM.RANGE()`. Note that some other TRACE32 functions which need random values are also affected by this seed value.

See also
- `SETUP
- `RANDOM()`
- `RANDOM.RANGE()`

## SETUP.ReDraw

Update whole screen

Format: `SETUP.ReDraw`

Usually only some parts of the screen are updated. This command can be used for updating, whenever a background program has overwritten the screen (e.g. messages from network drivers).

See also
- `SETUP
- ‘System Setup and Configuration’ in ‘IDE User’s Guide’`

## SETUP.RESTARTDO

Define restart PRACTICE script file

Format: `SETUP.RESTARTDO [<filename>]`

Defines a PRACTICE script (*.cmm), which is called when restarting the TRACE32 system, after an `ABORT` command. The arguments of the driver program are passed to the PRACTICE script parameters. As a default no PRACTICE script is executed after `ABORT`.

See also
- `SETUP
- ABORT
- ‘System Setup and Configuration’ in ‘IDE User’s Guide’`
SETUP.SOUND  Set sound generator mode

Format:  SETUP.SOUND [ON | ERROR | OFF]

OFF       Sound generator switched off.
ERROR     Sound generator active for input errors and program execution errors.
ON        Sound generator is active too when mouse is used (click sound).

See also
- SETUP
- BEEP
  ▲ 'System Setup and Configuration' in 'IDE User's Guide'

SETUP.STATistic  Cache and memory usage

Format:  SETUP.STATistic

This command indicates the memory and cache usage of

- System Controller Unit (SCU) of TRACE32-ICE
- PODBUS Ethernet Controller for TRACE32-ICD and TRACE32-FIRE

The displayed values have the following meaning:

- allocs: memory allocations
- misses: cache page misses
- hits: cache page hits
- frees: cache page frees due to memory request
- flushes: cache page flushes to host due to memory request

See also
- SETUP
  ▲ 'System Setup and Configuration' in 'IDE User's Guide'
Configure output of the STOre commands

<table>
<thead>
<tr>
<th>Format:</th>
<th>SETUP.STOre.&lt;subcommand&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;sub command&gt;:</td>
<td>INDENTation TAB</td>
</tr>
<tr>
<td></td>
<td>SYMBOLIC [ON</td>
</tr>
<tr>
<td></td>
<td>RESet</td>
</tr>
</tbody>
</table>

Configures the output of the commands STOre, ClipSTOre, and AutoSTOre, which list the current TRACE32 settings in the format of a PRACTICE script (*.cmm).

**INDENTation**
Default: 1 space

Sets the type of indentation inside the generated output: One tab or number of <spaces>.

**SYMBOLIC**
Default: ON

Saves breakpoints, markers, and groups as addresses or as symbol names.
- **ON**: Stores the symbol name, but not the address of the symbol.
- **OFF**: Stores the address, but not the symbol name.

**RESet**

Resets the user-defined settings to the TRACE32 default settings.

Example:

```plaintext
Break.Set func2 ; for demo purposes, let’s set a breakpoint on the symbol func2
SETUP.STOre.INDENTATION 4. ; let’s indent with 4 spaces
SETUP.STOre.SYMBOLIC OFF ; OFF: store only the address of the symbol
; ON: store only the symbol name
STOre ~~~\my-store.cmm Break Analyzer
PEDIT ~~~\my-store.cmm
```

![Image](image_url)

A Indentation: 4 spaces
B Only the address, here **R:0x1064** of **func2**, is stored (SYMBOLIC OFF).

See also
- SETUP
Selects the number of spaces generated by a TAB character. The default is 8. Useful in conjunction with source level debuggers, if the TAB count defines the block nesting level and the TAB expansion value is not 8 (like on DOS).

**See also**
- SETUP
- 'System Setup and Configuration' in 'IDE User's Guide'
Time values are displayed by TRACE32 in an easily readable format. If this option is activated, time values are displayed in a scientific floating point format. This format is easier to process by external tools.

### Format:

```
SETUP.TIMEFORM [ON | OFF]
```

<table>
<thead>
<tr>
<th>default (OFF)</th>
<th>scientific (ON)</th>
</tr>
</thead>
<tbody>
<tr>
<td>12.345us</td>
<td>12.34e-6</td>
</tr>
<tr>
<td>12.345ms</td>
<td>12.34e-3</td>
</tr>
<tr>
<td>12.345s</td>
<td>12.345</td>
</tr>
<tr>
<td>12.345ks</td>
<td>12.345e3</td>
</tr>
</tbody>
</table>

### See also

- SETUP
- ’System Setup and Configuration’ in 'IDE User's Guide'
SETUP.UpdateRate

**Format 1:**  
SETUP.UpdateRate \(<time> \mid <value>\)

**Format 2:**  
SETUP.UpdateRate \(<rate>\)

**Format 1 applies to PowerDebug modules:** The information of the visible windows is updated 10 times/s by default. This update is done for all windows if the program execution is stopped or for all windows with run-time/dualport access while the CPU is executing the program.

The defined update rate is not guaranteed:

- The update rate is lower e.g. if the host system is busy.
- Immediate updates are done when the mouse is moved.

```plaintext
SETUP.UpdateRate 500.ms ; update the window information all 500 ms
SETUP.UpdateRate 3. ; update the window information 3 times/s
```

**Format 2 applies to In-Circuit Emulators (ICE):** Limits the update rate of the TRACE32 window. As the default the update rate is not limited, i.e. it is as fast as the windows can be calculated. This can lead to update rates of 100 screens/s, e.g. if only a small dump window with a running timer of the target is displayed. This fast updates can cause a slow host to go down under the burden of the fast updates. It causes also a lot of traffic on the ethernet, if the TRACE32 system is connected by net. The **SETUP.UpdateRate** command limits the update to the given rate (in updates/s). Updates caused by user input are not slowed down.

```plaintext
SETUP.UpdateRate 10. ; limit the update to 10 times/s
```

**See also**

- **SETUP**
- **SYstem.POLLING**

▲ 'System Setup and Configuration' in 'IDE User's Guide'
If enabled, PRACTICE scripts (*.cmm) will stop on warnings. Otherwise only errors stop PRACTICE scripts, warnings don’t stop.

See also
- SETUP
- ‘System Setup and Configuration’ in 'IDE User’s Guide'
SETUP.XSLTSTYLESHEET  Reference to XSLT stylesheet for XML files

Format:  SETUP.XSLTSTYLESHEET ["<xsl_file>""]

Use this command if you want to configure which of your XSLT stylesheets is used for the transformation of XML files in a web browser after they have been exported by TRACE32.

NOTE:  The *.xsl file itself is not created by TRACE32.

The SETUP.XSLTSTYLESHEET command only creates a reference to your XSLT stylesheet.

Without parameter:  Resets the XSLT stylesheet to the default (t32transform.xsl).

With parameter:  Inserts the tag `<xml-stylesheet ...href="..."/>` in the XML file during file export from TRACE32 and sets the attribute `href="..."` to the specified `<xsl_file>`.

- The command does not check if the `<xsl_file>` is a valid URL or not.
- To reference an absolute path to a stylesheet, the path must be in URL syntax; for example, if the path of the XSLT stylesheet is `c:\users\john\foo.xsl`, you have to write:

  SETUP.XSLTSTYLESHEET "file:///c:/users/john/foo.xsl"

- If path and file name contain spaces, replace each space with `%23`. Example: `\john doe\` must be specified as `\john%23doe\`

Examples of TRACE32 commands that create XML export files are the commands of the List.EXPRT and COVerage.EXPRT command group or the PRinTer command group.

For an example of how to use SETUP.XSLTSTYLESHEET, see PRinTer.FILE. In contrast to the other XML export commands, PRinTer.FILE will only emit the tag `<xml-stylesheet ...href="..."/>` if a stylesheet was explicitly specified with SETUP.XSLTSTYLESHEET.

See also

- SETUP
- PRinTer.FILE
- COVerage.EXPRT

©1989-2018 Lauterbach GmbH
SILENT

Suppress informational messages in AREA window

Format: **SILENT.<command>**

Pre-command for suppressing *informational messages* in the default **AREA** window **A000**. The **SILENT** pre-command has no effect on error and warning messages. These messages are always printed to the default **AREA** window **A000**.

**<command>**

Examples of commands where the **SILENT** pre-command suppresses informational messages in the default **AREA** window **A000**:
- Data.Find, Trace.Find
- Data.LOAD.*
- PWD
- ChDir
- SYStem.Option commands that are *manually* toggled at the TRACE32 command line by omitting the keywords **ON** / **OFF**, e.g. SYStem.Option MMUSPACES

**Example:** For demo purposes, the same two commands are executed with and without the **SILENT** pre-command. The result is shown in the **AREA** window below.

```
AREA.view A000

PRINT %COLOR.Red "With the pre-command SILENT:"
SILENT.Data.Load.ELF "~~/demo/arm/compiler/gnu/sieve.elf" /RelPATH
SILENT.Data.Load.ELF "~~/demo/arm/compiler/arm/armle.axf" /RelPATH

PRINT "" ;print an empty line

PRINT %COLOR.BLUE "Without SILENT, informational messages are printed:"
Data.Load.ELF "~~/demo/arm/compiler/gnu/sieve.elf" /RelPATH
Data.Find D:0x0--0xffffff 0xE9
```

See also
- **AREA**
SOFTKEYS

Toggle the buttons on the softkey bar

Format:  

```
SOFTKEYS [ON | OFF]
```

The SOFTKEYS command without argument toggles the buttons on the softkey bar.

- **ON**  
  Activates the buttons on the softkey bar.

- **OFF**  
  Deactivates the buttons on the softkey bar.

See also
- SETUPICONS
- STATUSBAR
- SUBTITLE
- TITLE

TOOLBAR
The **STATUSBAR** command without argument toggles the TRACE32 state line.

- **ON** Displays the state line.
- **OFF** Hides the state line.

**See also**
- **SETUPICONS**
- **SOFTKEYS**
- **SUBTITLE**
- **TITLE**
STOre

STOre

Store a setting

Format:  STOre <filename> [%<format>]<item> …] [/<option>]

<format>:  sYmbol | NosYmbol

<item>:  ALL | HISTory | Win | WinPAGE | …

<device_specific_settings>

<option>:  NoDate

Stores the settings in the format of a PRACTICE script (*.cmm). They can be executed by using the DO command. The command is available also in other systems, like emulators or analyzers, with more system specific options.

For a detailed description of <format> and <option>, refer to the STOre command in general_ref_s.pdf.

HELP

Store help settings and bookmarks.

HISTory

Store command history to file.

PBREAK

Store the breakpoints created for PRACTICE scripts (*.cmm).

Win

Store entire window configuration (all pages).

WinPAGE

Store current window page.

…

All other keywords refer to the commands of the same name.

See also

- SETUPQUITDO
- AutoSTOre
- ClipSTOre
- DO

- 'Window System' in 'IDE User's Guide'
- 'Breakpoint Handling' in 'Debugger Basics - Training'
- 'Breakpoint Handling' in 'Debugger Basics - SMP Training'
- 'Breakpoint Handling' in 'Training FIRE Basics'
- 'Breakpoints' in 'Training ICE Basics'

©1989-2018 Lauterbach GmbH
Subtitle

Define a window subtitle for AMP debugging

Allows to automatically add text to the header of each window. This takes effect only for the windows opened after the subtitle definition. A `SUBTITLE` command without any parameter will delete a previous setting.

The most common field of application is in AMP (asymmetric multiprocessing) debugging. The `SUBTITLE` command helps you to easily distinguish between different TRACE32 PowerView GUIs of a multicore target.

**Example:** Let's assume you want to append the flag `; main cluster` to the main window and all other windows of the first TRACE32 PowerView GUI. To accomplish this, include these two lines in your PRACTICE start-up script (*.cmm) for the first TRACE32 instance:

```cmm
;maincluster.cmm
;;; your code
TITLE "TRACE32 PowerView ; main cluster"
SUBTITLE %String " ; main cluster"
;;; your code
```

To flag the main window and all other windows of the second TRACE32 PowerView GUI with `; secondary cluster`, include these two lines in your PRACTICE start-up (*.cmm) for the second TRACE32 instance:

```cmm
;secondarycluster.cmm
;;; your code
TITLE "TRACE32 PowerView ; secondary cluster"
SUBTITLE %String " ; secondary cluster"
;;; your code
```
In this example, the two TRACE32 instances were started in the FDI window mode. For this mode you need the following setting in the configuration file (config.t32):

```
SCREEN=
FDI
```

Alternatively, you can select the FDI window mode from the WindowMode drop-down list in the T32Start application; see “Advanced Settings and Default Advanced Settings” (app_t32start.pdf).

See also
- SETUPICONS
- SOFTKEYS
- STATUSBAR
- TOOLBAR
- TITLE
Pack files into an archive

Format: \texttt{\textsc{TAR} \langle archive\_name\rangle \langle file\_selector\rangle [\langle options\rangle]} \\
\texttt{\langle option\rangle:} \texttt{NoRecursion} \texttt{ListOnly}

Packs the selected files \textit{without compression} into a tape archive formatted archive. The files are selected from the directory path given by the \texttt{\langle file\_selector\rangle}.

By default, the given directory from the \texttt{\langle file\_selector\rangle} and all its subdirectories are scanned recursively down. All selected files from this directory tree are then stored into the archive.

\texttt{\langle archive\_name\rangle} File name of the archive to be created.

\texttt{\langle file\_selector\rangle} The file selector may contain a directory and a file name with wildcard characters to select appropriate files.

\texttt{NoRecursion} Switch off subdirectory tree scanning. Store only files from the given directory of the \texttt{\langle file\_selector\rangle}.

\texttt{ListOnly} The files are not packed into an archive but just listed in the default \texttt{AREA \textcolor{red}{view \textcolor{black}{window A000}}}. The size of an \texttt{AREA \textcolor{red}{view \textcolor{black}{window}}} is by default limited to about 100 lines. However, you can increase the number of lines with the \texttt{AREA.Create} command.

\textbf{Example 1:}

```
;store all PRACTICE script files (*.cmm) from the TRACE32 demo directory and all its subdirectories. The archive "scripts.tar" is created within the home directory of the user.
TAR ~/scripts.tar    ~/demo/*.cmm
```

\textbf{Example 2:}

```
;list all *.c files from the TRACE32 demo directory and all its subdirectories in the default \texttt{AREA.view} window
TAR ~/archive.tar    ~/demo/*.* /ListOnly

;display the file listing
AREA.view
```
Example 3:

`; to compress the *.tar archive to a zipped tape archive file (.tar.gz),
; use the ZIP command afterwards
TAR ~/arm.tar ~~/demo/arm/*.cmm
ZIP ~/arm.tar ~/arm.tar.gz

; optional: start Windows Explorer and select the file
OS.Command start explorer.exe /select, %USERPROFILE%\arm.tar.gz

The host command is printed in blue.
TIMEOUT

Specify time-out for TRACE32 command

Format:  TIMEOUT <timevalue> <trace32_command>

Terminates a <trace32_command> after the specified <timevalue> has elapsed. The TIMEOUT command has same effect as clicking the STOP button on the TRACE32 main toolbar after a defined time.

Example:

;your start-up script

TIMEOUT 500.ms Data.Copy D:0--0x3fffffff VM:0 /Byte /Verify

IF TIMEOUT()==TRUE()
(
  PRINT %WARNING "'Data.Copy D:0--0x3fffffff VM:0' canceled after 50.ms"
)

See also

_TIMEOUT()
Define a screen title for TRACE32 PowerView GUI

The command defines the header of the TRACE32 main window. Running the **TITLE** command without any parameter will delete the previous setting - the header will be empty.

The most common field of application is to distinguish between different TRACE32 GUI's of a multicore or multi CPU target.

```
TITLE %String "TRACE32 Debugger for CPU0"
TITLE %String "TRACE32 for MPC5676R"
```

See also
- **TOOLBAR**
- **SUBTITLE**
- **SETUP.ICONs**
- **TARGET.ICONs**
- **SOFTKEYS**
- **STATUSBAR**

---

**Format:**

TITLE [%<formats>] "<your_text>" ...

**<format>:**

Ascii
Binary
Decimal
Hex
String
The **TOOLBAR** command without argument toggles the TRACE32 main toolbar.

- **ON** Displays the toolbar.
- **OFF** Hides the toolbar.

---

**See also**

- TITLE
- SETUP ICONS
- SOFTKEYS
- STATUSBAR
- SUBTITLE
- 'Screen Display' in 'IDE User's Guide'
- 'Customizing the TRACE32 PowerView GUI' in 'Training Menu'
The file will be opened only, if the generated window is active. When exiting from the window, it will be frozen automatically. In the tracking mode the file is always open.

Current selection.

Right-click for popup menu.

EDIT opens the file in the TRACE32 editor. To configure an external editor, use SETUP.EDITEXT.

Offset of current selection:

- in decimal and hex as well as
- in line and column number

;display file and scroll to line 7
;display line numbers
TYPE ~/demo/arm/compiler/arm/arm.c 7. /LineNumbers

See also
- ComPare
- DUMP
- PATCH
- EDIT.file
- FIND
- TRACK.COLUMN()
- TRACK.LINE()

▲ 'File and Folder Operations' in 'IDE User's Guide'
Using the **UNARchive** commands, you can extract files from Linux libraries (.a) and Microsoft libraries (.lib) to a directory. **UNARchive.Table** and **UNARchive.Show** help to determine the contents of the library and to check the result of the extract operation.

**UNARchive.extract** extracts all files of a library into a given directory on disc. If the directory is not given, then the temporary directory of TRACE32 is used instead.

See also

- **UNARchive.extract**
- **UNARchive.Show**
- **UNARchive.Table**
- **UNPAck**
- ‘File and Folder Operations’ in ‘IDE User’s Guide’
- ‘Release Information’ in ‘Release History’

**UNARchive.extract**

Extract files from Linux library and Microsoft library

**Format:**

```
UNARchive.extract <library_name> [directory]
```

Extracts all files of a library into a given directory on disc. If the directory is not given, then the temporary directory of TRACE32 is used instead.

See also

- **UNARchive**
UNARchive.Show

Extract files from library and list them in window

Format: **UNARchive.Show** <library_name> [<directory>]

Same behavior as the **UNARchive** command, but additionally lists the names of all extracted files in the **UNARchive.Show** window.

See also
- UNARchive

UNARchive.Table

Display table of contents of library

Format: **UNARchive.Table** <library_name>

Displays the table of contents of the library in the **UNARchive.Table** window without extracting the library files to disc.

See also
- UNARchive
UNPACK

Expand files (with LZW algorithm)

Format: \texttt{UNPACK \textit{source} [\textit{destination}]}\footnote{\texttt{UNPACK} \texttt{<source>} \texttt{[<destination>]}\texttt{; \text{compress object file}}\texttt{; \text{restore original file}}}

The compressed file in expanded back to the original file format. The source must be a file in LZW encoding, generated by the \texttt{PACK} command. The source and the destination file names must be different. If only one argument is supplied, the resulting file will have the same name as the source file.

\begin{verbatim}
::PACK mcc.abs mcc.pak ; compress object file
...::UNPACK mcc.pak mcc.abs ; restore original file
\end{verbatim}

See also
\begin{itemize}
\item UNARchive
\item UNZIP
\item PACK
\item ZIP
\end{itemize}

\texttt{\% File and Folder Operations\% in 'IDE User's Guide'}
UNZIP

Expand GZIP archive file (with DEFLATE algorithm)

Format:  UNZIP <source> [<destination>]

Unzips a file that was compressed to a GZIP archive. The source and the destination file names must be different. If only one argument is supplied, the resulting file will have the same name as the source file.

::UNZIP \t32\man.t32 ; un-pack online manual

See also

UNARchive  UNPACK  PACK  ZIP

▲ 'File and Folder Operations' in 'IDE User's Guide'
Using the `VERSION` command group, you can display version information about the TRACE32 hardware modules and software as well as the TRACE32 environment settings.

**See also**

- `VERSION.ENVironment`
- `VERSION.HARDWARE`
- `VERSION.SOFTWARE`
- `VERSION.view`
- `LICENSE`

▲ 'VERSION Functions’ in 'General Functions'

## VERSION.ENVironment

**Display environment settings**

<table>
<thead>
<tr>
<th>Format:</th>
<th><code>VERSION.ENVironment</code></th>
</tr>
</thead>
</table>

The currently used environment settings of the TRACE32 software are displayed. This includes e.g. the currently started executable, TRACE32 system directory, TRACE32 configuration file, etc.

![Image of environment settings](image)

PRACTICE functions can be used in PRACTICE scripts (*.cmm) to return individual values from the window. For more information, refer to the functions() listed below.

**See also**

- `VERSION`
  - `OS.ID()`
  - `OS.PresentExecutableDirectory()`
  - `OS.PresentHELPDirectory()`
  - `OS.PresentSystemDirectory()`
  - `OS.PresentWorkingDirectory()`
  - `VERSION.ENVironment()`

- `VERSION.view`
  - `OS.PresentConfigurationFile()`
  - `OS.PresentExecutableFile()`
  - `OS.PresentHomeDirectory()`
  - `OS.PresentTemporaryDirectory()`
  - `OS.VERSION()`
**VERSION.HARDWARE**

Display hardware versions

Displays the serial numbers and revision information of the TRACE32 hardware modules in the **VERSION.HARDWARE** window.

PRACTICE functions can be used in PRACTICE scripts to return individual values from the window. For more information, refer to the functions() listed below.

See also

- `VERSION
- `VERSION.view
- `CABLE.NAME()
- `ID.PREPROcessor()
- `SYStem.USEMASK()
- `VERSION.SERIAL.CABLE()
- `VERSION.SERIAL.DEBUG()

▲ 'VERSION Functions’ in ‘General Functions'
▲ 'Version Management and Licensing’ in ‘IDE User’s Guide’

**VERSION.SOFTWARE**

Display software versions

The versions of the TRACE32 software modules are displayed in the **VERSION.SOFTWARE** window.

PRACTICE functions can be used in PRACTICE scripts to return individual values from the window. For more information, refer to the functions() listed below.

See also

- `VERSION
- `VERSION.view
- `OS.PresentExecutableFile()
- `VERSION.BUILD()
- `VERSION.BUILD.BASE()
- `VERSION.SOFTWARE()

▲ 'Version Management and Licensing’ in ‘IDE User’s Guide’
The versions of the TRACE32 modules (hardware and software) and TRACE32 hardware serial numbers are displayed.

See also

- VERSION
- VERSION.ENVironment
- VERSION.HARDWARE
- VERSION.SOFTWARE

▲ 'Version Management and Licensing' in 'IDE User's Guide'
Welcome to TRACE32

The **WELCOME** command group provides quick access to important manuals and allows you to search for PRACTICE demo scripts (*.cmm).

We recommend that you familiarize yourself with the **WELCOME** command group by starting with the description of the **Welcome to TRACE32!** dialog, see **WELCOME.view**.

**See also**
- **WELCOME.CONFIG**
- **WELCOME.SCRIPTS**
- **WELCOME.STARTUP**
- **WELCOME.view**

**WELCOME.CONFIG**

Configure search paths for PRACTICE demo scripts

Using the **WELCOME.CONFIG** command group, you can add and remove the paths where the **WELCOME.SCRIPTS** window searches for PRACTICE demo scripts (*.cmm). In addition you can set a filter to limit the search to file names that match the filter criterion. The search directories are automatically re-scanned after you have modified the search paths or the filter. You can abort the re-scan at any time.

We recommend that you use the **WELCOME.CONFIG.state** window for configuration.

Any changes you have made to the default search directories and the default filter can be reset.

**See also**
- **WELCOME.CONFIG.ADDDIR**
- **WELCOME.CONFIG.RemoveDIR**
- **WELCOME.CONFIG.state**
- **WELCOME.view**
- **WELCOME.CONFIG.FILTER**
- **WELCOME.CONFIG.RESet**
- **WELCOME**

IDE Reference Guide

©1989-2018 Lauterbach GmbH
WELCOME.CONFIG.ADDDIR

Add a new script search path

Format: WELCOME.CONFIG.ADDDIR <path>

See also
■ WELCOME.CONFIG

WELCOME.CONFIG.FILTER

Set the script search filter

Format: WELCOME.CONFIG.FILTER "<filter>"

Default: *.cmm

See also
■ WELCOME.CONFIG

WELCOME.CONFIG.ReMoveDIR

Remove a script search path

Format: WELCOME.CONFIG.ReMoveDIR <path>

See also
■ WELCOME.CONFIG

WELCOME.CONFIG.RESet

Reset the script search configuration

Format: WELCOME.CONFIG.RESet

See also
■ WELCOME.CONFIG
Opens the script search configuration window, listing the directories where the **WELCOME.SCRIPTS** window searches for PRACTICE demo scripts (*cmm).

When you initially open the window, you will see the search directories that apply to the TRACE32 executable (**t32m<architecture>.exe**) you have started.

To reset the search directories, run the **WELCOME.CONFIG.RESet** command.

**See also**
- **WELCOME.CONFIG**
**WELCOME.SCRIPTS**

Open the script search window

Displays the **Search for scripts** window, where you can search and browse for PRACTICE scripts (*.cmm) in the TRACE32 demo folder. For a step-by-step procedure of how to search for, preview, and execute PRACTICE demo scripts, see “**Demo Scripts in the TRACE32 Demo Folder**” (practice_user.pdf).

See also

- WELCOME
- WELCOME.view
- ‘Release Information’ in ‘Release History’

**WELCOME.STARTUP**

Open the welcome window if not disabled

Displays the **Welcome to TRACE32!** window unless it was disabled by the user; see check box in the **Welcome to TRACE32** window (**WELCOME.view**).

See also

- WELCOME
- WELCOME.view
- ‘Release Information’ in ‘Release History’
Open the welcome window

Format: WELCOME.view

Displays the Welcome to TRACE32! window. Using this command, the dialog window pops up even if it was disabled by the user, see [C].

A Explains what to observe before you can start debugging.
B Manuals you should read. The list is dynamic, i.e. it adjusts to the TRACE32 executable (t32m<architecture>.exe) you are using.
C Activates/deactivates this window. Your setting is stored in the TRACE32 user preferences.
D Opens the HELP window.
E Opens the Search for scripts window (see WELCOME.SCRIPTS command).

See also
- WELCOME
- WELCOME.CONFIG
- WELCOME.SCRIPTS
- WELCOME.STARTUP

▲ 'Release Information' in 'Release History'
Window handling (size, position, font size, etc.)

There are two types of commands in the **Win** command group:

1. **Window commands**

   Examples of window commands are **WinPOS**, which determines size, position, and name of the next window, or **WinCLEAR**, which closes a named window.

2. **Window pre-commands**

   Examples of window pre-commands are **WinLarge.** *<window>* , which increases the font size for a particular window, and **WinFreeze.** *<window>* , which creates a frozen window.

The following examples are for demo purposes only. To try a script, simply copy it to a `test.cmm` file, and then step through the script (See “How to…”).

**Example 1:** The window command **WinPOS** determines size, position, and name of the next window.

```
  ;  <x>  <y>  <width>  <height>  <optional_parameters>  <name>
  WinPOS 0.  0.  130.  36. , , ,  myWin01
  List.auto ;open the List window displaying the source listing
```

**Example 2:** Window pre-commands are used to open a window in large font size and a frozen window.

```
  WinLarge. Register.view  ;open the Register window in large font size
  WinFreeze. Register.view  ;open the Register window as a frozen window
```

**See also**

- WinBack
- WinExt
- WinMid
- WinPAGE.Delete
- WinPAGE.select
- WinPRT
- WinTABS
- WinCLEAR
- WinFIND
- WinOverlay
- WinPAGE.List
- WinPAN
- WinResist
- WinTOP
- WinDEFaultSIZE
- WinFreeze
- WinPAGE
- WinPAGE.REName
- WinPOS
- WinRESIZE
- WinTrans
- WinDuplicate
- WinLarge
- WinPAGE.Create
- WinPAGE.RESet
- WinPrint
- WinSmall

▲ ‘WINDOW Functions’ in ‘IDE Functions’
WinBack

Generate background window

Format: \texttt{WinBack.\langle command\rangle}

Pre-command for creating a background window, i.e., the window is pushed into the background after operations.

See also

- \texttt{Win}
- \texttt{WinFreeze}
- \texttt{WinResist}
- \texttt{WINDOW.NAME()}
- "Window System" in 'IDE User’s Guide'

WinCLEAR

Erase windows

Format: \texttt{WinCLEAR [WinTOP | \{<windowname>\} | <pagename>]} [Example]

If no parameters are set, all windows of one page are erased. If multiple window names are specified, only those windows will be cleared.

Resistant windows cannot be cleared by this command. That is, windows with the pre-command \texttt{WinResist.<window> or WR.<window> are not} cleared.

\begin{description}
  \item[WinTOP] (or TOP as an alias) Deletes the uppermost window.
  \item[<windowname>] Window names are case-sensitive. They are created with the \texttt{WinPOS} command.
  \item[<pagename>] Page names are case-sensitive. They are created with the \texttt{WinPAGE.Create} command.
\end{description}
Example:

```
WinPOS , , , , , , W1  ;open window 1 and name it W1
Register.view

WinPOS , , , , , , W2  ;open window 2 and name it W2
PER.view

WinPOS , , , , , , W3  ;open window 3 and name it W3
List.Mix

WinPOS , , , , , , myTraceWin ;open window 4 and name it myTraceWin
Trace.List

WinCLEAR TOP  ;clear only the uppermost window
    ;i.e. window myTraceWin in this example

WinCLEAR W1 W3  ;clear only the windows named W1 and W3
    ;the remaining window is W2
```

See also

- Win
- WinPAGE.RESet
- WinResist
- WINDOW.NAME()

- WINPAGE.EXIST()
- 'Window System' in 'IDE User's Guide'
- 'I/O Commands' in 'Training Script Language PRACTICE'
WinDEFaultSIZE

Apply a user-defined default size to windows

Format:  WinDEFaultSIZE [<hsize> | <vsize>]

Applies a user-defined default size (width and height) to TRACE32 windows that are used to output data. The WinDEFaultSIZE command has no effect on dialog-style windows, such as the SYStem.state or Break.Set window, which are used to configure data.

Your settings are applied to all windows that are opened after running the WinDEFaultSIZE command. Windows that are already open are not resized. The user-defined default size is valid for the current TRACE32 session or until you specify a new default size.

- `<hsize>` Applies a user-defined default width to windows.
- `<vsize>` Applies a user-defined default height to windows.
- no parameters Restores the TRACE32 settings for window default sizes.

TRACE32 ignores any user-defined setting (width or height or both) that is outside the built-in minimum or maximum size for a particular window. You can override your own defaults for any window by executing a WinPOS command just before opening the next window.

The following script is just intended to illustrate the effects of the various window-sizing commands on TRACE32 windows. To try this script, simply copy it to a test.cmm file, and then step through the script (See “How to...”).

```
WinDEFaultSIZE 100. 9.; Defines the user-defined window default size
; The user-defined default size is applied to the next two windows
Trace.CHART
Data.List

; Overrides the user-defined default size - but only for the next window
WinPOS , , 70. 15. , , , myWin01 ; myWin01 is a user-defined window name
Trace.List

; The user-defined window default size takes effect again
Symbol.Browse

; Resize the window named myWin01
WinRESIZE 120. 20. myWin01

; WinDEFaultSIZE has no effect on dialog-style windows, such as:
SYSteem.state
```

See also

- Win
- WinPOS
- WinRESIZE
- ‘Release Information’ in ‘Release History’
WinDuplicate

**Format:**  
WinDuplicate

Pre-command to duplicate the last active window. The command can only be used for non-dialog windows.

See also
- Win

WinExt

**Generate external window**

**Format:**  
WinExt.<command>

Pre-command for creating an external window, i.e., the window is handled independently of the TRACE32 main window. It's useful in an MDI configuration to move a window out of the main window.

**NOTE:** Using the WinExt pre-command, you can detach an individual window from the TRACE32 main window - even if TRACE32 is in MDI window mode.

```plaintext
; In MDI mode, you cannot detach a window from the TRACE32 main window
SYStem.state

; However, by prepending the WinExt pre-command, you can detach the
; window from the TRACE32 main window
WinExt.SYStem.state
```

The position and size of TRACE32 on start-up can be defined in the SCREEN= section of the configuration file. For more information, refer to "Screen/Windows" (installation.pdf).

See also
- Win
- FramePOS
- 'Window System' in 'IDE User's Guide'
Searches for text in the uppermost window or in the window that has the specified window name. The function `FOUND()` returns TRUE if the search string was found. As an alternative to the `WinFIND` command, click the window you want, and then press **Ctrl+F** or choose **Edit menu > Find**.

As of build no. 86141 (July 2017), the behavior of the command has changed: It now displays an error message in the TRACE32 message line if the specified `<windowname>` does not exist.

### Format

```
WinFIND [[<lines>] "<string>"] [WinTOP | <windowname>] [\<options>]
```

### <options>

- **Back**
- **Case**

### Details

- **WinTOP** (or **TOP** as an alias)
  - Performs a search operation in the uppermost window.

- **<windowname>**
  - Window names are case-sensitive. They are created with the `WinPOS` command.

- **Back**
  - This option is used to search backward.

- **Case**
  - This option is used to compare case-sensitive, otherwise lower and upper-case characters are not distinguished.
; open a terminal window
TERM.METHOD COM COM3 115200. 8 NONE 1STOP NONE
TERM.Mode VT100
TERM.SCROLL ON
TERM.SIZE 80. 1000
WinPOS 76.0 25.0 80. 0. 0. TermWin
TERM.view
; your code...

PRINT "Wait for KBoot prompt in terminal window"

RePeaT
{
    WAIT 1.s
    ; update screen display
    SCREEN.display
    WinFIND "KBoot>" TermWin
}
WHILE !FOUND()

; your code...

See also
- Win
- FIND
- Data.Find
- Data.GOTO
- Data.GREP
- FOUND()
- WINDOW.NAME()

▲ ‘Window System’ in ‘IDE User’s Guide’
WinFreeze

Generate frozen window

Format:  \texttt{WinFreeze.<command>}

Pre-command for generating a frozen window. Note that the window is then not updated to the current state.

You can also choose \texttt{Freeze} from the window manager menu (left mouse) to freeze or unfreeze the window contents.

Diagonal lines indicate that the window contents are frozen.

Click the top left icon to open the window manager menu.

For more information about the window manager menu, see “\texttt{Window Manager Menu}” (ide_user.pdf).

\texttt{WinFreeze.Register.view} ;Open the Register window as a frozen window

See also

\begin{itemize}
\item Win
\item WinBack
\item WinResist
\item \texttt{WINDOW.NAME()}
\item ‘Window System’ in ‘IDE User's Guide’
\end{itemize}
**WinLarge**

Generate window with large font

Format: `WinLarge.<command>`

Pre-command for generating a window with large font. Switching to large font is very useful in presentations before large audiences.

```
WinPOS , , , , , , WinL ; user-defined window name
WinLarge.Register.view ; large font
WinPOS , , , , , , WinM
WinMid.Register.view ; regular font (default)
WinPOS , , , , , , WinS
WinSmall.Register.view ; small font
```

See also

- Win
- WinMid
- WinSmall
- WINDOW.NAME()  
- 'Window System' in 'IDE User’s Guide'

---

**WinMid**

Generate window with regular font

Format: `WinMid.<command>`

Pre-command for generating a window with regular font. This pre-command is included for backward compatibility.

See also

- Win
- WinLarge
- WinSmall
- WINDOW.NAME()  
- 'Window System' in 'IDE User’s Guide'
WinOverlay

Pile up windows on top of each other

Format:  \texttt{WinOverlay.<command>}

Superimposes the next window on the active window.

This behavior is used in a \texttt{List} or \texttt{Data.GREP} window to open a new \texttt{List} window on top of and with nearly the same size as the active window. Press \texttt{Esc} to return to the previous window, or drag the new window to a new position to make the previous window visible again.

For example, the \texttt{Data.List} and \texttt{List.auto} windows display the same type of content and can thus be exactly superimposed in terms of position and size.

\begin{verbatim}
Data.List ;active window
WinOverlay.List.auto func2 ;next window is superimposed on Data.List,
;displaying a listing for the function func2
\end{verbatim}

Double-clicking a function or variable name in an HLL listing executes the \texttt{WinOverlay} command by default.

\begin{itemize}
\item Window sizes may vary for windows that do not display the same type of content.
\item The double-click behavior within a \texttt{List} or \texttt{Data.GREP} window can be changed by the \texttt{SETUP.LISTCLICK} command.
\end{itemize}

See also

- Win
- Data.GREP
- List
- SETUP.LISTCLICK

\textsuperscript{▲} ‘Window System’ in ‘IDE User’s Guide’
The **WinPAGE** command group is used to create and manage window pages. A window page is a collection of windows displayed on the screen. The pages allow you to quickly switch between different window collections.

Right-click the toolbar to create a new page or switch to another page. Alternatively, use **WinPAGE.List**.

**NOTE:** Page names are case-sensitive.

See also
- **Win**
- **WinPAGE.REName**
- **WinPAGE.Create**
- **WinPAGE.RESet**
- **WinPAGE.Delete**
- **WinPAGE.select**
- **WinPAGE.List**
- **WINPAGE.EXIST()**

▲ ‘Screen Display’ in ‘IDE User’s Guide’
▲ ‘Window System’ in ‘IDE User’s Guide’
WinPAGE.Create
Create and select page

Format: \texttt{WinPAGE.Create} \[\texttt{<pagename>} [/\texttt{NoSELect}] \] | \[\texttt{/,NoSELect}]\]

Creates a new page and selects the new page. If no parameters are set, the new page is assigned an auto-incremented default window page name \texttt{P000}, \texttt{P001}, etc.

- \texttt{<pagename>}
  - If the page name does not exist, then a new page with that name is created and selected.
  - If the page name corresponds to the name of an existing page, then this page is selected.
  - Page names are case-sensitive.

- \texttt{,}
  - Auto-increments the name of the next page; additionally you can use \texttt{NoSELect}.

- \texttt{NoSELect}
  - A new page is created in the background, but not selected. The current page continues to remain the active page.

\begin{quote}
\texttt{WinPAGE.Create ANALYZER}\quad;\texttt{create page for Analyzer windows}
\texttt{Analyzer.List}\quad;\texttt{create an Analyzer window on this page}
\texttt{WinPAGE.select P000}\quad;\texttt{select the default page}
\end{quote}

See also
- \texttt{WinPAGE}
- \texttt{WinPAGE.List}
- \texttt{Win}
- \texttt{WINPAGE.EXIST()}

\texttt{▲ 'Window System' in 'IDE User's Guide'}

WinPAGE.Delete
Delete page

Format: \texttt{WinPAGE.Delete} \texttt{<pagename>}

Removes one page from the page list including all windows within it. Page names are case-sensitive.

\begin{quote}
\texttt{WinPage.Delete P000}\quad;\texttt{delete the first page}
\end{quote}

See also
- \texttt{WinPAGE}
- \texttt{WinPAGE.List}
- \texttt{Win}
- \texttt{WINPAGE.EXIST()}

\texttt{▲ 'Window System' in 'IDE User's Guide'}
WinPAGE.List  Display an overview of all pages and their windows

Format:  

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WinPAGE.List [ShowAlways]</td>
<td>Opens the WinPAGE.List window, listing all pages and their windows by name.</td>
</tr>
</tbody>
</table>

A  P000 and P001 are examples of default page names.
B  Three windows on page P001. Default window names are auto-incremented W001, W002, etc. To assign a user-defined name to a window, run WinPOS and then open the window.
C  ANALYZE and EDIT are examples of user-defined page names. To create a new page with a user-defined page name, use WinPAGE.Create <pagename>.

**ShowAlways**  
Expands all +/- buttons in the WinPAGE.List window and keeps them expanded. Use this option if you want to see at a glance on which page the individual windows are located.

Left, right, and double-clicking inside the WinPAGE.List window executes these actions:

- Single-clicking any text line selects a page and all windows on that page.
- Double-clicking an empty line creates a new page with an auto-incremented page name, P000, P001, P002, etc. Alternatively, right-click an empty line, and then select New Page.
- Right-clicking any text line opens the Pages popup menu:
  - **Rename** inserts the WinPAGE.REName command in the command line. Alternatively, double-click the page you want. You can now rename the selected page via the command line.
  - **Delete** deletes the selected page and all windows on that page right away.
  - **Show** and **Hide** toggle the window list of an individual page or of all pages.
  - **Show always** corresponds to the option ShowAlways.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WinResist.WinPAGE.List</td>
<td>open a resistant window to navigate between pages</td>
</tr>
</tbody>
</table>

See also

- WinPAGE
- WinPAGE.RESet
- WinPAGE.Create
- WinPAGE.Delete
- WinPAGE.REName
- Win
- WINPAGE.EXIST()

▲ 'Window System' in 'IDE User's Guide'

©1989-2018 Lauterbach GmbH
WinPAGE.REName

Format: `WinPAGE.REName <old_pagename> <new_pagename>`

Renames an existing page. Page names are case-sensitive.

```
WinPage.REName PI ANALYZER ; renames page PI to ANALYZER
```

See also

- WinPAGE
- WinPAGE.List
- Win
- WINPAGE.EXIST()

WinPAGE.RESet

Reset window system

Format: `WinPAGE.RESet`

All pages and windows are removed, including resistant windows. That is, windows with the pre-command `WinResist.<window>` or `WR.<window>` are also removed.

See also

- WinPAGE
- WinPAGE.List
- Win
- WinCLEAR
- Window System’ in ‘IDE User’s Guide’

WinPAGE.select

Select page

Format: `WinPAGE.select [pagename]`

If no parameters are set, the next page will be selected. Page names are case-sensitive.

See also

- WinPAGE
- WinPAGE.List
- Win
- WINPAGE.EXIST()
- Window System’ in ‘IDE User’s Guide’
Specify a window cut-out

This command is used to scroll or pan a window. If no window name is defined, the uppermost window will be modified. This allows to scroll a window by using PRACTICE. Usually, you pan and scroll a window with the mouse.

As of build no. 86141 (July 2017), the behavior of the command has changed: It now displays an error message in the TRACE32 message line if the specified windowname does not exist.

Format: \texttt{WinPAN} [\texttt{<x>}] [\texttt{<y>}] [\texttt{WinTOP} | \texttt{<windowname>}]\\

- \texttt{<x>}: Use positive values to pan to the right; negative values to pan to the left.
- \texttt{<y>}: Use positive values to scroll down; negative values to scroll up.
- \texttt{WinTOP} (or \texttt{TOP} as an alias): Scrolls or pans the uppermost window.
- \texttt{<windowname>}: Window names are case-sensitive. They are created with the \texttt{WinPOS} command.

See also

- WinPOS
- Win
- WINDOW.NAME()
- 'Window System' in 'IDE User's Guide'
WinPOS

Define window dimensions and window name

[Examples]  [Script in Demo Folder]

<table>
<thead>
<tr>
<th>Format:</th>
<th>WinPOS [&lt;pos&gt;] [&lt;size&gt;] [&lt;scale&gt;] [&lt;windowname&gt;] [&lt;state&gt;] [&lt;header&gt;]</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;state&gt;:</td>
<td>Normal</td>
</tr>
</tbody>
</table>

Determines the coordinates for the next window opened by a command. The window position can be specified as an integer value, floating point value or in percent of the total screen size. <header> allows to replace the default window header, which is the name of the command that generated the window, by a user-defined one.

NOTE: As of build 72592, the syntax of the WinPOS command was changed. If your script stops at a WinPOS command with percentage values, please check the syntax. The PRACTICE script below uses a WinPOS switch to illustrate the syntax change.

```
IF (VERSION.BUILD.BASE()>72592.)
(  ;as of build 72592, 3 commas are required as separators
  ;after percentage values
  WinPOS 50% 0% 50% 100% ,,,, WINDOWNAME
)
ELSE
(  ;before build 72592, only 2 commas were required
  WinPOS 50% 0% 50% 100% ,, WINDOWNAME
)
```

| <pos>         | • <left> = x-coordinate as a floating point or integer or percentage value.  
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• &lt;up&gt; = y-coordinate as a floating point or integer or percentage value.</td>
</tr>
</tbody>
</table>

| <size>        | Parameter Type:  
|---------------|------------------------------------------------------------------------|
|               | • <hsize> = width of a window as a floating point or integer or percentage value.  
|               | • <vsize> = height of a window as a floating point or integer or percentage value. |

| <scale>       | Parameter Type:  
|---------------|------------------------------------------------------------------------|
|               | • <hscale> = width of the scale area of a window.  
|               | • <vscale> = height of the scale area of a window. |
### Examples

<table>
<thead>
<tr>
<th>&lt;windowname&gt;</th>
<th>The &lt;windowname&gt; argument can be used to assign a user-defined name to a window. Usually WinPOS commands will be generated by a STOre command. Window names are case-sensitive.</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;header&gt;</td>
<td>Specify the user-defined window caption as a quoted string.</td>
</tr>
</tbody>
</table>

WinPOS ,,,,,, myName
Trace.List ;open a Trace.List window named myName

;changes the <up> position of the window that is opened next
WinPOS , 20% ,,,,,, myName2
WinPOS , 20. ,,,,,, myName2
WinPOS , 200.0e-1 ,,,,,, myName2
WinPOS , 20.0 ,,,,,, myName2
WinPOS , 0x14 ,,,,,, myName2

WinPOS 1. 1. 103. 20. 2. 0. myWin Normal "Intermixed Source/Assembly"
Data.ListMix

WinPOS 1. 1. 20. 20. 2. ,, DUMP
Data.dump 0x1000

WinPOS 1. 10.
TYPE ~~~\test.txt

; PRACTICE script generated by the STOre Win command
WinCLEAR
WinPOS 0.0 0.0 120. 36. 16. 1. W000
WinTABS 10. 10. 25. 62.
Data.List

WinPOS 0.0 40.5 58. 36. 5. 0. W001
Var.Frame /Locals /Caller

WinPOS 62.0 40.5 58. 36. 0. 0. W002
Var.Watch %SpotLight flags ast
Due to the WinPOS syntax change, you may encounter compatibility problems in PRACTICE scripts that (a) make heavy use of WinPOS commands and (b) need to be compatible with old and new TRACE32 software.

As of build 77665, TRACE32 provides a solution in the form of a PRACTICE helper script that allows you to bypass potential WinPOS compatibility problems. To preview the PRACTICE helper script, run this command:

```
B:CD.PSTEP ~/demo/practice/winpos.cmm
```

If you encounter WinPOS compatibility problems, we recommend the following solution:

1. Include the PRACTICE helper script in your own PRACTICE scripts (*.cmm), see ON CMD ... in the example below.
2. Rename all WinPOS commands to WinPOS2.
3. Separate all existing WinPOS2 arguments with commas (without spaces, see WinPOS2 below).
4. Replace each omitted WinPOS2 argument with a comma, too.

```plaintext
; the individual arguments can optionally be comma-separated
WinPOS 10., 20., 30, 40., 1., 2., myName3
Frame.view
```

See also

- WinPAN
- WinTAB
- Win
- WinTOP
- WinDEFaultSIZE
- WinRESIZE
- WinTABS
- WinTOP
- WINdow.EXIST()
- WINDOW.NAME()

▲ 'Window System' in 'IDE User's Guide'
▲ 'I/O Commands' in 'Training Script Language PRACTICE'
The **WinPrint** pre-command is used to generate a hardcopy or a file from one command. The numbers of columns and lines in the window are adapted to the possibilities of the printer. Printer selection can be executed by the **PRinTer** command.

Thus, the output can also be re-routed to a file. In the case of some commands, extended parameters are possible for printing more than one page.

```
WinPrint.Data.dump 0--0xffff
WinPrint.Analyzer.List (-1000.)--100. Address Data sYmbol
```

**PRinTer.FILE** provides an example of how to print the contents of TRACE32 windows to file in XML format.

### See also
- **WinPRT**
- **PRinTer.EXPORT**
- **WINDOW.NAME()**
- 'Printer Operations' in 'IDE User's Guide'
- 'Window System' in 'IDE User's Guide'
- 'Release Information' in 'Release History'
- 'Document your Results' in 'Training FIRE Basics'
- 'Document your Results' in 'Training ICE Basics'

### WinPRT

**Format:**  
WinPRT [<command>]

This command can be used to make a hardcopy of the uppermost window or the window that has the specified name. It is the same command as **Print** in the window manager menu. It can be used to make multi-page printouts controlled by PRACTICE.

As of build no. 86141 (July 2017), the behavior of the command has changed: It now displays an error message in the TRACE32 message line if the specified `<windowname>` does not exist.
WinTOP
(or **TOP** as an alias)

*<windowname>*

WinTOP
(or **TOP** as an alias)

*<windowname>*

WinTOP
(or **TOP** as an alias)

*<windowname>*

- Prints the uppermost window.
- Window names are case-sensitive. They are created with the **WinPOS** command.

**Example:**

```
PRinTer.TYPE PSP12
WinPOS ,, 80. 30. ,,, MYWIN
Analyzer.Chart.TASKState
LOCAL &page
&page=0
WHILE &page<10.
  (WinPRT MYWIN
   WinPAN 70. 0. MYWIN)
```

See also

- **WinPrint**
- **Win**
- **PRinTer**
- **PRinTer.HardCopy**
- **WINDOW.NAME()**
- 'Printer Operations' in 'IDE User's Guide'
- 'Window System' in 'IDE User's Guide'

---

**WinResist**

Generate a resistant window

Format: **WinResist.<command>**

This pre-command is used to create a resistant window. This window cannot be cleared by the command **WinCLEAR**. The window is displayed on all window pages and usually used for editing PRACTICE files. Resistant windows can be deleted manually by the mouse-based window functions or by the command **WinPAGE.RESet**.

```
WinResist.PEDIT test.cmm ; open PRACTICE script
```

See also

- **Win**
- **WinBack**
- **WinCLEAR**
- **WinFreeze**
- 'Window System' in 'IDE User's Guide'
WinRESIZE

New size for window

Format:  

\[
\text{WinRESIZE} \ [\text{<width>}] \ [\text{<height>}] \ [\text{WinTOP} \mid \text{<windowname>}] 
\]

Resizes the uppermost window or the window that has the specified \text{<windowname>}

As of build no. 86141 (July 2017), the behavior of the command has changed: It now displays an error message in the TRACE32 message line if the specified \text{<windowname>} does not exist.

\textbf{WinTOP}  
(or \text{TOP} as an alias)

Resizes the uppermost window.

\textbf{<windowname>}  

Window names are case-sensitive. Use \text{WinPOS} to assign a user-defined name and an initial size to a window.

In the example below, \text{WinPOS} is used to open a window with a user-defined size and name. \text{WinRESIZE} is used to re-apply the user-defined size if the named window is already open. In addition, the named window is displayed on top of all other windows.

```plaintext
; determine whether the named window is already open
IF WINdow.EXIST("myWin01")==FALSE()
   ; apply a user-defined size (height, width) and name to the window
   WinPOS , , 120. 20. , , myWin01
   Group.List ; Open the window
ELSE
   ; resize the named window by re-applying the initial size
   WinRESIZE 120. 20. myWin01
ENDC
; bring the named window to the top of the display hierarchy
WinTOP myWin01
```

See also

- \text{Win}
- \text{WinDEFaultSIZE}
- \text{WinPOS}
- \text{WinTOP}
- WINdow.EXIST()
- "Window System" in "IDE User's Guide"
WinSmall

Generate window with small font

Format: 

**WinSmall.<command>**

Pre-command for generating a window with small font.

See also

- Win
- WinLarge
- WinMid

▲ 'Window System’ in 'IDE User’s Guide'

WinTABS

Specify size of re-sizable columns

Format: 

**WinTABS <tabs> [ <tabs>...]**

TRACE32 PowerView display windows may contain fixed columns and re-sizable columns. If the mouse is positioned on the border of re-sizable column, the cursor changes to a re-size cursor (see screenshot below).

The command **WinTABS** is used the specify the size of re-sizable columns for the next display window that will be opened.

WinTABS 20. 5. 20. 40. ; specify size for the **code**, **label**, **mnemonic** and **comment** columns of ; the Data.List window

Data.List

WinTABS 50. 20. ; specify the size for the **tree** and ; **InternalBAR.Log** column of a ; Trace.STATistic.TREE window

Trace.STATistic.TREE

See also

- Win
- WinPOS

▲ 'Window System’ in 'IDE User’s Guide’
Brings the named window to the top of the display hierarchy. If the named window is not on the current window page, then the page of the window is selected and the window is moved to the top of the display hierarchy. To check whether a window with given window name exists, use the PRACTICE function WINdow.EXIST().

Format: **WinTOP [<windowname>]**

- **<windowname>** Window names are case-sensitive. A window name is created by using the WinPOS command followed by the command that opens the actual window.

**Example:** In this PRACTICE script, a custom dialog `my_dialog` is brought to the top of the display hierarchy, provided the dialog already exists. Else a new dialog with the window name `my_dialog` is created.

```plaintext
IF WINdow.EXIST(my_dialog)           ;if the window name exists,
(                                    ;bring the window to the top
  WinTOP my_dialog
) ELSE
(                                    ;if the window name does not exist,
  WinPOS ,,,, , my_dialog           ;assign the window name to this
  DIALOG.view                       ;custom dialog
  (                                 ;custom dialog
    HEADER "MyDialog"
    POS 0. 0. 30. 1.
    TEXT "A named dialog window"
    BUTTON "Close" "DIALOG.End"
  )
)
ENDDO
```

**See also**

- Win
- WinPOS
- WinRESIZE
- WINdow.EXIST()
Pre-command for generating a transparent window. These kinds of external windows will allow windows in the background to shimmer through.

Prerequisites:

- Windows 2000 and later.
- Available for the TRACE32 window modes FDI and MTI.
- If the TRACE32 window mode is MDI, then the WinTrans pre-command can only be used together with the WinExt pre-command:

```
WinExt.WinTrans.Register.view ; open a transparent Register.view window; while TRACE32 is in MDI window mode
```

See also

- Win
- 'Window System' in 'IDE User's Guide'
**ZERO**

**ZERO.offset**

**Set time reference**

Format: `ZERO.offset <time>`

Sets the global reference time. The global time is used to correlate different analyzers within one TRACE32 system, e.g. state analyzer and port analyzer. Usually this function will be used like the reference cursor function. Every analyzer has one reference cursor, but the absolute reference time is unique to the whole system.

```
ZERO 100.s ; move zero reference point by 100 s
```

See also
- ZERO.offset
- ZERO.RESet

```
; display a trace listing with the ti.zero column as the first column
Trace.List TIme.Zero DEFault /Track

;set the zero reference point to record no. -10000.
ZERO.offset Trace.RECORD.TIME(-10000.)

; go to the zero reference point in the trace listing
Trace.GOTO 0.s
```

**ZERO.RESet**

**Reset to original value**

Format: `ZERO.RESet`

Resets the offset to the global reference time.

See also
- ZERO.offset

```
```
Compress files to GZIP archive (with DEFLATE algorithm)

Format:  

```
ZIP <source> [<destination>]
```

The source file is compressed to a GZIP archive.

```
ZIP refl.ad ; pack file
```

See also

- PACK
- UNPACK
- UNZIP

▲ ‘File and Folder Operations’ in ‘IDE User's Guide’
The following help filters are available for the `HELP.FILTER` command group:

- Help Filters for TRACE32 Hardware/Software
- Help Filters for RTOS Debuggers
- Help Filters for Third-Party Integrations
- Help Filters for UEFI Debuggers
- Help Filters for Debug Back-Ends

### Help Filters for TRACE32 Hardware/Software

<table>
<thead>
<tr>
<th>Filter</th>
<th>TRACE32 Hardware/Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>bdm*</td>
<td>TRACE32 debugger</td>
</tr>
<tr>
<td></td>
<td>e.g. bdmarm, bdmsh4</td>
</tr>
<tr>
<td>esi</td>
<td>TRACE32 ERPOM simulator</td>
</tr>
<tr>
<td>fire*</td>
<td>TRACE32-FIRE</td>
</tr>
<tr>
<td></td>
<td>e.g. fire12, firexc166</td>
</tr>
<tr>
<td>fireport</td>
<td>TRACE32-FIRE port analyzer</td>
</tr>
<tr>
<td>gdb*</td>
<td>TRACE32 GDB Front-end</td>
</tr>
<tr>
<td></td>
<td>e.g. gdbarm, gdbi386</td>
</tr>
<tr>
<td>ice*</td>
<td>TRACE32-ICE</td>
</tr>
<tr>
<td></td>
<td>e.g. ice51, ice166</td>
</tr>
<tr>
<td>iceport</td>
<td>TRACE32-ICE port analyzer</td>
</tr>
<tr>
<td>icr*</td>
<td>TRACE32 real-time trace</td>
</tr>
<tr>
<td></td>
<td>e.g. icretm, icrsh4</td>
</tr>
<tr>
<td>icrstm</td>
<td>TRACE32 CombiProbe</td>
</tr>
<tr>
<td>mon*</td>
<td>TRACE32 ROM monitor</td>
</tr>
<tr>
<td></td>
<td>e.g. mon68k, mon166</td>
</tr>
<tr>
<td>nat386</td>
<td>Windows native process debugger</td>
</tr>
<tr>
<td>nexus*</td>
<td>TRACE32 NEXUS debugger</td>
</tr>
<tr>
<td></td>
<td>e.g. nexusppc, nexusmac</td>
</tr>
<tr>
<td>pdg*</td>
<td>TRACE32 pdg Front-end</td>
</tr>
<tr>
<td></td>
<td>e.g. pdgarm</td>
</tr>
<tr>
<td>pi</td>
<td>PowerIntegrator</td>
</tr>
<tr>
<td>pp</td>
<td>PowerProbe</td>
</tr>
<tr>
<td>sim*</td>
<td>TRACE32 instruction set simulator or TRACE32 Front-end</td>
</tr>
<tr>
<td></td>
<td>e.g. simarm, simppc</td>
</tr>
<tr>
<td>Filter</td>
<td>TRACE32 Hardware/Software</td>
</tr>
<tr>
<td>--------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>stg</td>
<td>Stimuli generator</td>
</tr>
<tr>
<td>time</td>
<td>TRACE32 timing analyzer</td>
</tr>
<tr>
<td>tp</td>
<td>Trigger probe</td>
</tr>
</tbody>
</table>

**Help Filters for RTOS Debuggers**

<table>
<thead>
<tr>
<th>Filter</th>
<th>RTOS Debuggers</th>
</tr>
</thead>
<tbody>
<tr>
<td>rtos*</td>
<td>RTOS Debugger</td>
</tr>
<tr>
<td>rtosamx</td>
<td>AMX</td>
</tr>
<tr>
<td>rtosartk</td>
<td>ARTK</td>
</tr>
<tr>
<td>rtosartx166</td>
<td>ARTX-166</td>
</tr>
<tr>
<td>rtosbios</td>
<td>DSP/BIOS</td>
</tr>
<tr>
<td>rtoschibios</td>
<td>RTOS Debugger for ChibiOS/RT</td>
</tr>
<tr>
<td>rtoschorus</td>
<td>Chorus Classic and Chorus Micro</td>
</tr>
<tr>
<td>rtoscmicro</td>
<td>Cmicro</td>
</tr>
<tr>
<td>rtoscmx</td>
<td>CMX and CMX-TINY+</td>
</tr>
<tr>
<td>rtosecos</td>
<td>eCos</td>
</tr>
<tr>
<td>rtosembos</td>
<td>embOS</td>
</tr>
<tr>
<td>rtosepoc</td>
<td>Symbian OS EKA1</td>
</tr>
<tr>
<td>rtosfamos</td>
<td>FAMOS</td>
</tr>
<tr>
<td>rtosfreertos</td>
<td>FreeRTOS</td>
</tr>
<tr>
<td>rtoshi7000</td>
<td>HI7000</td>
</tr>
<tr>
<td>rtoshios</td>
<td>HIOS</td>
</tr>
<tr>
<td>rtoslinux</td>
<td>Linux</td>
</tr>
<tr>
<td>rtoslynxs</td>
<td>LynxOS</td>
</tr>
<tr>
<td>rtosmqx</td>
<td>MQX</td>
</tr>
<tr>
<td>rtosmtos</td>
<td>MTOS-UX</td>
</tr>
<tr>
<td>rtosnetbsd</td>
<td>NetBSD</td>
</tr>
<tr>
<td>rtosnorti</td>
<td>NORTi</td>
</tr>
<tr>
<td>rtosnucleus</td>
<td>Nucleus PLUS</td>
</tr>
<tr>
<td>Filter</td>
<td>RTOS Debuggers</td>
</tr>
<tr>
<td>-------------</td>
<td>---------------------------------------</td>
</tr>
<tr>
<td>rtosokl4</td>
<td>OKL4</td>
</tr>
<tr>
<td>rtosorti</td>
<td>OSEK/ORTI</td>
</tr>
<tr>
<td>rtosorti</td>
<td>RTOS Debugger for OSEK/ORTI</td>
</tr>
<tr>
<td>rtosos21</td>
<td>OS21</td>
</tr>
<tr>
<td>rtosos9</td>
<td>OS-9</td>
</tr>
<tr>
<td>rtososeb</td>
<td>OSE Epsilon</td>
</tr>
<tr>
<td>rtososec</td>
<td>OSE Classic</td>
</tr>
<tr>
<td>rtososeck</td>
<td>OSEck</td>
</tr>
<tr>
<td>rtososed</td>
<td>OSE Delta</td>
</tr>
<tr>
<td>rtososee</td>
<td>OSE Epsilon</td>
</tr>
<tr>
<td>rtospikeos</td>
<td>PikeOS</td>
</tr>
<tr>
<td>rtosprkernel</td>
<td>PrKERNEL</td>
</tr>
<tr>
<td>rtospsos</td>
<td>pSOS+</td>
</tr>
<tr>
<td>rtospxros</td>
<td>PXROS</td>
</tr>
<tr>
<td>rtosqnx</td>
<td>QNX</td>
</tr>
<tr>
<td>rtosquadros</td>
<td>RTXC Quadros</td>
</tr>
<tr>
<td>rtosrealos</td>
<td>RTOS Debugger for REALOS</td>
</tr>
<tr>
<td>rtosrt7700</td>
<td>RTOS/7700</td>
</tr>
<tr>
<td>rtos rtc</td>
<td>RealTimeCraft</td>
</tr>
<tr>
<td>rtosrtems</td>
<td>RTEMS</td>
</tr>
<tr>
<td>rtosrtx166</td>
<td>RTX166 and RTX166 tiny</td>
</tr>
<tr>
<td>rtosrtx51</td>
<td>RTX51 and RTX51 tiny</td>
</tr>
<tr>
<td>rtosrtxarm</td>
<td>RTX-ARM</td>
</tr>
<tr>
<td>rtosrtxc</td>
<td>RTXC</td>
</tr>
<tr>
<td>rtosrubus</td>
<td>Rubus OS</td>
</tr>
<tr>
<td>rtossclipta</td>
<td>Sciopta</td>
</tr>
<tr>
<td>rtossmx</td>
<td>SMX</td>
</tr>
<tr>
<td>rtossymbian2</td>
<td>Symbian OS EKA2</td>
</tr>
<tr>
<td>rtossysbios</td>
<td>SYS/BIOS</td>
</tr>
<tr>
<td>rtosthreadx</td>
<td>ThreadX</td>
</tr>
<tr>
<td>rtosuc3cmp</td>
<td>MicroC3/Compact</td>
</tr>
</tbody>
</table>
### Filter | RTOS Debuggers
--- | ---
rtosuc3std | MicroC3/Standard
rtosuclinux | uClinux
rtosucos, rtosucos3 | MicroC/OS-II
 | MicroC/OS-III
rtosuiplus | uiPLUS
rtosuitron | uiTRON
rtosvdk | VDK
rtosvrt | VRTX32/68K, VRTX80, VRTXmc/68K, VRTXsa
rtosvrtx | VRTX32/68K, VRTX80, VRTXmc/68K, VRTXsa
rtosvxworks | Vx Works
rtoswince | Windows CE
rtoswindows | Windows Standard
rtoszeos | ZeOS

### Help Filters for Third-Party Integrations

<table>
<thead>
<tr>
<th>Filter</th>
<th>Third-Party Tool</th>
</tr>
</thead>
<tbody>
<tr>
<td>inteclipse</td>
<td>Eclipse</td>
</tr>
<tr>
<td>intcodeblock</td>
<td>CodeBlocks</td>
</tr>
<tr>
<td>intcw</td>
<td>CodeWright</td>
</tr>
<tr>
<td>inteasy</td>
<td>EasyCase</td>
</tr>
<tr>
<td>intexdi2</td>
<td>Windows CE Platform Builder</td>
</tr>
<tr>
<td>intlabview</td>
<td>LabView</td>
</tr>
<tr>
<td>intose</td>
<td>OSE Illuminator</td>
</tr>
<tr>
<td>intrhapsody</td>
<td>Rhapsody in MicroC</td>
</tr>
<tr>
<td>intrhapsodycpp</td>
<td>Rhapsody in C/C++</td>
</tr>
<tr>
<td>inttornado</td>
<td>Tornado I</td>
</tr>
<tr>
<td>intvbas</td>
<td>Visual Basic Interface</td>
</tr>
<tr>
<td>intxttools</td>
<td>X-Tools and X32</td>
</tr>
</tbody>
</table>
### Help Filters for UEFI Debuggers

<table>
<thead>
<tr>
<th>Filter</th>
<th>UEFI Debuggers</th>
</tr>
</thead>
<tbody>
<tr>
<td>uefibldk</td>
<td>UEFI BLDK Debugger</td>
</tr>
<tr>
<td>uefih2o</td>
<td>UEFI H2O Debugger</td>
</tr>
<tr>
<td>uefitiano</td>
<td>UEFI TianoCore Debugger</td>
</tr>
</tbody>
</table>

### Help Filters for Debug Back-Ends

<table>
<thead>
<tr>
<th>Filter</th>
<th>Debug Back-Ends</th>
</tr>
</thead>
<tbody>
<tr>
<td>back*</td>
<td>Debug Back-Ends</td>
</tr>
<tr>
<td>backgtl</td>
<td>GTL Debug Back-End</td>
</tr>
<tr>
<td>backxcp</td>
<td>XCP Debug Back-End</td>
</tr>
</tbody>
</table>