

PowerView Command Reference



Release 09.2023

PowerView Command Reference

TRACE32 Online Help

TRACE32 Directory

TRACE32 Index

TRACE32 Documents	
PowerView User Interface	
PowerView Command Reference	1
History	14
AREA	16
AREA	Message windows 16
AREA.CLEAR	Clear area 17
AREA.CLOSE	Close output file 17
AREA.Create	Create or modify message area 18
AREA.Delete	Delete message area 18
AREA.List	Display a detailed list off all message areas 19
AREA.OPEN	Open output file 21
AREA.PIPE	Redirect area to stdout 22
AREA.RESet	Reset areas 22
AREA.SAVE	Save AREA window contents to file 22
AREA.Select	Select area 23
AREA.STDERR	Redirect area to stderr 24
AREA.STDOUT	Redirect area to stdout 24
AREA.view	Display message area in AREA window 25
AutoSTOre	27
AutoSTOre	Save and restore settings (history, GUI, etc.) automatically 27
BITMAPEDIT	29
BITMAPEDIT	Bitmap editor for user-defined icons 29
ChDir	30
ChDir	Change directory 30
ClipSTOre	31
ClipSTOre	Store settings to clipboard 31
CmdPOS	32
CmdPOS	Controls the position of TRACE32 in MWI window mode 32
CommandLineKEYS	34
CommandLineKEYS	Special characters 34
ComPare	35

ComPare	Compare files	35
COPY		37
COPY	Copy files	37
DATE		38
DATE	Display date and time	38
DEL		39
DEL	Delete file	39
DIALOG		40
DIALOG	Custom dialogs	40
Dialog Definition Programming Commands		40
BAR	Progress bar	41
BOX	Define a decorative border	42
BUTTON	Raised button with an icon and text	43
CHECKBOX	Define a checkbox	45
CHOOSEBOX	Define a choose box	46
CLOSE	Catch window close	48
COMBOBOX	Define a combo box	49
DEFBUTTON	Define the default button	50
DEFCOMBOBOX	Define a default combo box	50
DEFEDIT	Define a default edit control	50
DEFHOTCOMBOBOX	Define a default hot combo box	50
DEFHOTEDIT	Define a default hot edit control	50
DEFMEDIT	Define a default multiline edit control	51
DLISTBOX	Define a draggable list box	51
DYNAMIC	Dynamic, single-line area	52
DYNCOMBOBOX	Define a dynamic combo box	53
DYNDEFCOMBOBOX	Define a default dynamic combo box	53
DYNDEFHOTCOMBOBOX	Define a dynamic default hot combo box	53
DYNHOTCOMBOBOX	Define a dynamic hot combo box	53
DYNLTEXT	Dynamic single-line text area in bold and large font size	54
DYNPULLDOWN	Define a dynamic pull-down list	55
DYNTXT	Dynamic, single-line text area in regular font size	57
EDIT	Define an edit control	58
HEADER	Define window header	59
HELP	Define a help icon	60
HOTEDIT	Define a hot edit control	61
HOTCOMBOBOX	Define a hot combo box	62
ICON	New icon in top left corner of dialog	62
INFOTEXT	Define a multiline info text box on a dialog	63
INIT	Initialize dialog	64
LINE	Define a decorative horizontal line	66
LISTBOX	Define a list box	67

LTEXT	Static, single-line text area in bold and large font size	68
LEDIT	Define an edit control in bold and large font	69
MEDIT	Define a multiline edit control	69
MLISTBOX	Define a multiline list box	69
NAME	Internal dialog name	70
POS	Define position and size	71
POSX	Define position and size on the x-axis	73
POSY	Define position and size on the y-axis	73
PULLDOWN	Define a static pull-down list	74
SPACE	Apply previous height to next dialog element	75
STATIC	Place an icon in a dialog	75
SUBROUTINE	Define subroutine for usage in dialog command blocks	76
TEXT	Static, single-line text area in regular font size	77
TEXTBUTTON	Flat button with text only	78
TREEBUTTON	Implements a +/- toggle button	79
UPDATE	Executes commands periodically	80
VLIN	Decorative vertical line	81
DIALOG.AREA	Adds an output area to a custom dialog	82
DIALOG.DIR	Display a folder picker dialog	83
DIALOG.Disable	Disable dialog elements	84
DIALOG.Enable	Enable dialog elements	85
DIALOG.END	Close the dialog window	85
DIALOG.EXecute	Execute a dialog button	85
DIALOG.File	Pass file name from OS file dialog to PRACTICE script	86
DIALOG.File.open	Display an OS file-open dialog	87
DIALOG.File.SAVE	Display an OS file-save dialog	88
DIALOG.File.SELECT	Display an OS file-select dialog	89
DIALOG.MESSAGE	Create dialog box with an information icon	90
DIALOG.NOYES	Create dialog box with NO and YES buttons	90
DIALOG.OK	Create dialog box with an exclamation mark	91
DIALOG.Program	Interactive programming	92
DIALOG.ReProgram	Dialog programming	94
DIALOG.SELect	Programmatically focus on this dialog	94
DIALOG.Set	Modify the value of a dialog element	95
DIALOG.SetDIR	Browse for folder	97
DIALOG.SetFile	Pass file name from OS file dialog to custom dialog	98
DIALOG.SetFile.open	OS file-open dialog > file name > EDIT element	98
DIALOG.SetFile.SAVE	OS file-save dialog > file name > EDIT element	100
DIALOG.SetFile.SELECT	OS file-select dialog > file name > EDIT element	100
DIALOG.STORAGE	Stored macros in the dialog context	101
DIALOG.STORAGE.define	Define macros stored in the dialog context	101
DIALOG.STORAGE.LOAD	Load macros stored in the dialog context	102
DIALOG.STORAGE.SAVE	Update macros stored in the dialog context	102

DIALOG.view	Show dialog window	102
DIALOG.YESNO	Create dialog box with YES and NO buttons	104
DIR		105
DIR	List subdirectories and files	105
DUMP		107
DUMP	Binary file dump	107
EDIT		109
EDIT	TRACE32 editor	109
Overview EDIT		109
EDIT.CLOSE	Close a text file	110
EDIT.ENCoding	Change the file encoding	111
EDIT.EXTErn	Use specified external ASCII editor to edit file	112
EDIT.file	Edit file	113
EDIT.Find	Perform find, replace and goto operations in TRACE32 editors	116
EDIT.FORMAT	Format file contents an editor window	118
EDIT.Goto	Go to specified line	119
EDIT.InsertText	Insert text	120
EDIT.List	List editor files	120
EDIT.LOAD	Load text files	121
EDIT.OPEN	Use TRACE32 editor to edit file	122
EDIT.QUIT	Discard modifications	123
EDIT.REDO	Redo the previously undone edit/edits	123
EDIT.Replace	Open dialog window on the Replace tab	124
EDIT.REVERT	Revert file	124
EDIT.SAVE	Save a text file	125
EDIT.SELect	Select text/code in an editor window	126
EDIT.UNDO	Undo the last edit/edits	127
ERROR		128
ERROR.RESet	Reset PRACTICE error	128
EVAL		129
Eval	Evaluate expression	129
FIND		131
FIND	Search in file	131
FramePOS		132
FramePOS	Controls the position of TRACE32 in MDI window mode	132
HELP		135
HELP	Online help	135
HELP.Bookmark	Show help bookmark list	136
HELP.Bookmark.ADD	Files on bookmark list	137
HELP.Bookmark.ADD.file	Add file to bookmark list	137
HELP.Bookmark.ADD.Find	Add file to bookmark list	138

HELP.Bookmark.ADD.Index	Add file to bookmark list	139
HELP.Bookmark.DELeTe	Delete from bookmark list	139
HELP.Bookmark.show	Show help bookmark list	140
HELP.checkUPDATE	Automatic update check for new help-files	140
HELP.command	Command related support	140
HELP.FILTER	Filters for online help	141
HELP.FILTER.Add	Add a filter to the help filter list	142
HELP.FILTER.Delete	Delete filter from help filter list	142
HELP.FILTER.List	List all help filters	143
HELP.FILTER.RESet	Reset help filter system	143
HELP.FILTER.set	Activate/deactivate help filters for online help	144
HELP.Find	Perform a full-text search in online help	144
HELP.Index	Search in indexed terms, commands, and functions	147
HELP.OPEN	Open PDF documentation for command or function	149
HELP.PDF	Open PDF file	150
HELP.PICK	Context-sensitive help	150
HELP.PRinT	Print help files	151
HELP.PRinT.PRinTsel	Print selected files	151
HELP.PRinT.SELect	Select files to print	151
HELP.PRinT.show	Show print help files	152
HELP.PRinT.UNSELect	Unselect all print files	152
HELP.Topics	Show the structure of the online help system	153
HELP.TREE	Display command tree	154
HISTory		155
HISTory	Command history of last executed commands	155
HISTory.eXecute	Execute command history	156
HISTory.SAVE	Store command history log	156
HISTory.Set	History settings	157
HISTory.SIZE	Command history and file history	158
HISTory.SIZE.cmd	Define log size of command history	158
HISTory.SIZE.FILE	Define number of recently used files in 'File' menu	159
HISTory.type	Display command history log of last executed commands	159
IFCONFIG		160
IFCONFIG	Ethernet or USB communication	160
IFCONFIG.PROfile	Display operation profiles	160
IFCONFIG.state	Interface configuration	162
IFCONFIG.TEST	Test interface function and speed	164
InterCom		165
InterCom	Data exchange between different TRACE32 PowerView instances	165
InterCom.ENABLE	User-defined InterCom name, auto-assigned port number	166
InterCom.Evaluate	Evaluate function via InterCom system	168
InterCom.execute	Execute command via InterCom system	169

InterCom.executeNoWait	Execute command via InterCom system	171
InterCom.NAME	Assign user-defined InterCom name	171
InterCom.PING	Test InterCom system	173
InterCom.PipeCLOSE	Close named pipe	173
InterCom.PipeOPEN	Open named pipe	174
InterCom.PipeREAD	Read from named pipe	174
InterCom.PipeWRITE	Write to named pipe	175
InterCom.PORT	Assign user-defined InterCom UDP port number	175
InterCom.WAIT	Wait for remote InterCom system	177
LICENSE		178
LICENSE	Manage TRACE32 licenses	178
LICENSE.List	Display all license information	178
LICENSE.REQest	Request a license	179
LICENSE.state	Display the currently used maintenance contract	180
LICENSE.UPDATE	Update the maintenance contract	181
LOG		182
LOG	Log TRACE32 commands and PRACTICE script calls	182
LOG.CLOSE	Close command log	183
LOG.DO	Log calls of PRACTICE scripts	183
LOG.OFF	Switch off command log	184
LOG.ON	Switch on command log	185
LOG.OPEN	Open command log file	185
LOG.toAREA	Log commands by writing them to an AREA window	187
LOG.type	Display command log	191
LS		191
LS	Display directory	191
MENU		192
MENU	Customize the user interface TRACE32 PowerView	192
MENU.AddMenu	Add one standard menu item	192
MENU.AddTool	Add a button to the main toolbar	193
MENU.Delete	Delete nested menu	194
MENU.Delete.NAME	Delete specified menu	194
MENU.PENdIng	Menu files waiting for compilation	195
MENU.PENdIng.List	List menu files waiting for compilation	195
MENU.PENdIng.RESet	Clear list of pending menu files	195
MENU.Program	Interactive programming	196
MENU.ReProgram	Menu programming	197
MENU.RESet	Default configuration	200
Programming Commands		201
ADD	Add definition to existing menu	201
ADdHERE	Define hook	201
AFTER	Place a new menu item or separator after the named menu item	202

BEFORE	Place a new menu item or separator before the named menu item	202
BUTTONS	Add user-defined local buttons to a window	203
DEFAULT	Define default item	204
DELETE	Delete a certain item	204
ELSE	Conditional compile	204
ENABLE	Conditional enable	205
HELP	Define a help item	206
IF	Conditional compile	206
MENU	Menu definition	207
MENUITEM	Item definition	209
NAME	Define an internal menu name	209
PERMENU	Menu or submenu created from peripheral file (*.per)	210
POPUP	Popup definition	211
REPLACE	Replace the following item	212
SEPARATOR	Separator definition	212
SUBROUTINE	Define menu subroutine	213
TEAROFF	Define tearoff menu	214
TOOLBAR	Toolbar definition	214
TOOLITEM	Item definition	214
WAIT	Wait with menu file compilation until system is ready	217
WIDTH	Increase/decrease button width	218
MKDIR		219
MKDIR	Create new directory	219
MKTEMP	Create file or directory with unique name	220
MV		223
MV	Rename file	223
OS		224
OS	Execute host commands	224
Overview OS		224
OS.Area	Re-route host command output to AREA window	227
OS.Command	Execute a host command	228
OS.Hidden	Execute a host command in silent mode	230
OS.OPEN	Open file in default application	231
OS.screen	Call up the shell or execute host command	233
OS.SetENV	Set operating system environment variables	234
OS.Window	Re-route host command output to the OS.Window	235
PACK		236
PACK	Compress files (with LZW algorithm)	236
PATCH		237
PATCH	Binary file patching	237
PATH		238

PATH	Define search paths for files used by TRACE32 commands	238
PATH	Search path	239
PATH.Delete	Delete search path	239
PATH.DOWN	Define search path at end of list	240
PATH.List	List search path	241
PATH.RESet	Reset search path	241
PATH.Set	Define search path	242
PATH.UP	Define search path at top of list	243
PRinTer		244
PRinTer	Print and export window contents	244
PRinTer.Area	Re-route printer output to AREA window in specified format	245
PRinTer.ClipBoard	Re-route printer output to clipboard in specified format	246
PRinTer.CLOSE	Close file after multiple printer outputs	246
PRinTer.CONFIG	Print-out configuration	247
PRinTer.CONFIG.HEADER	Print window title	247
PRinTer.CONFIG.OFFSET	Specify print-out borders	247
PRinTer.CONFIG.SIZE	Specify print-out size	248
PRinTer.EXPORT	Export formatted printer output to file	249
PRinTer.FILE	Re-route printer output to a file in specified file format	253
PRinTer.FileType	Select file format	256
PRinTer.HardCopy	Make a hardcopy of the screen	257
PRinTer.OFFSET	Specify print-out borders	257
PRinTer.OPEN	Re-route multiple printer outputs to the same file	258
PRinTer.PRINT	Print to opened printer file	260
PRinTer.select	Select printer	261
PRinTer.SIZE	Specify print-out size	262
PWD		263
PWD	Change directory	263
PYthon		264
PYthon.EDIT	Open Python script in editor	264
PYthon.INSTALL	Install RCL module and Python interpreter	264
PYthon.RUN	Run Python script in dedicated window	265
QUIT		266
QUIT	Return to operating system	266
REN		267
REN	Rename file	267
RM, RMDIR		268
RM	Delete file	268
RMDIR	Remove directory	268
SCHreeNShot		269
SCHreeNShot	Save a screenshot of a window to a file	269

SETUP		271
SETUP	Setup commands	271
SETUP.ASCIITEXT	Configure ASCII text display	272
SETUP.BAKfile	Enable backup file creation	274
SETUP.COLOR	Change colors	275
SETUP.DEVNAME	Set logical device name	276
SETUP.DropCoMmanD	Set command for files dropped into command line	277
SETUP.EDITEXT	Define an external editor	278
SETUP.EDITOR	TRACE32 editor configuration	280
SETUP.EDITOR.AutoSuggest	Show input suggestions while typing	281
SETUP.EDITOR.BAKfile	Make backup copy when file is saved	282
SETUP.EDITOR.HighLight	Control syntax highlighting	282
SETUP.EDITOR.Indentation	Select indentation method	283
SETUP.EDITOR.IndentSize	Set indentation size	284
SETUP.EDITOR.IndentWithTabs	Use tabulator for indentation	285
SETUP.EDITOR.Mode	Show visible whitespace or ASCII view	285
SETUP.EDITOR.SaveChangesPrompt	Save file if edit window closed	286
SETUP.EDITOR.SmartBackspace	Backspace maintains indentation	287
SETUP.EDITOR.SmartCursor	Control cursor movement	287
SETUP.EDITOR.SmartFormat	Automatic formatting	288
SETUP.EDITOR.state	Show editor configuration dialog	289
SETUP.EDITOR.TabSize	Set tabulator size	290
SETUP.EDITOR.TrailingWhitespace	Remove trailing whitespace	290
SETUP.EDITOR.TYPE	Set editor implementation	291
SETUP.EXTension	Set default file name extensions	292
SETUP.FASTRESPONSE	Optimize for fast response times	292
SETUP.FILETYPE	File type configuration	293
SETUP.FILETYPE.DropCoMmanD	Set command for dropped files	293
SETUP.FILETYPE.ENCoding	Set encoding mode	294
SETUP.FILETYPE.EXTension	Set default file name extensions	296
SETUP.HOLDDIR	Configure working directory	299
SETUP.ICONS	Display icons in popup menus	299
SETUP.InterComACKTIMEOUT	Sets the InterCom acknowledge timeout	300
SETUP.PDEBUG	PRACTICE debug configuration settings dialog	301
SETUP.PDEBUG.BlockClose	Block window closing commands	302
SETUP.PDEBUG.BlockPosition	Block window positioning commands	302
SETUP.PDEBUG.MacroRESet	Reset PRACTICE macros after ending script	302
SETUP.PDEBUG.RESet	Reset settings to default values	303
SETUP.PDEBUG.ScriptParams	Set PRACTICE debug script parameters	303
SETUP.PDEBUG.TermScripts	Terminate all pending PRACTICE scripts	303
SETUP.PDEBUG.WindowExternal	Open debug window as external window	304
SETUP.PDEBUG.WindowOnTop	Keep debug window on top	304
SETUP.PDFViewer	Context-sensitive help via your favorite PDF viewer	305

SETUP.PDFViewer.EXECutable	Path and executable of your PDF viewer	306
SETUP.PDFViewer.OPEN	Open a PDF of the help system	306
SETUP.PDFViewer.PRinT	Print PDF via HELP window	307
SETUP.PDFViewer.RESet	Reset the settings in SETUP.PDFViewer dialog	307
SETUP.PDFViewer.TEMPorary	Help configuration for demo purposes	308
SETUP.PDFViewer.TEMPorary.EXECutable	PDF viewer for demo purposes	308
SETUP.PDFViewer.TEMPorary.OPEN	Open a PDF of the help system	308
SETUP.PDFViewer.TEMPorary.PRinT	Print PDF via HELP window	309
SETUP.PDFViewer.TEMPorary.RESet	Reset demo-help configuration	309
SETUP.PYthon.EXECutable	Defines path to python interpreter	309
SETUP.QUITDO	Define quit PRACTICE script file	310
SETUP.RADIX	Radix mode	311
SETUP.RANDOM	Set seed for RANDOM() function	312
SETUP.ReDraw	Update whole screen	312
SETUP.RESOLVEDIR	Resolve symbolic links	313
SETUP.SOUND	Set sound generator mode	313
SETUP.STOPMESSAGE	Print message when STOP command is executed	313
SETUP.STOre	Configure output of the STOre commands	315
SETUP.TabSize	Configure tab width	316
SETUP.TIMEFORM	Select scientific time format	317
SETUP.UpdateRATE	Update rate for windows	318
SETUP.WARNSTOP	Configure PRACTICE stops	318
SETUP.XSLTSTYLESHEET	Reference to XSLT stylesheet for XML files	319
SHA1SUM		320
SHA1SUM	Calculate SHA1 checksum of a file	320
SILENT		321
SILENT	Suppress informational messages in AREA window	321
SOFTKEYS		322
SOFTKEYS	Toggle the buttons on the softkey bar	322
STATUSBAR		323
STATUSBAR	Toggle state line	323
STOre		324
STOre	Store settings as PRACTICE script	324
SUBTITLE		325
SUBTITLE	Define a window subtitle for AMP debugging	325
TAR		327
TAR	Pack files into an archive	327
TIMEOUT		329
TIMEOUT	Specify timeout for TRACE32 command	329
TITLE		330

TITLE	Define a main window title for a TRACE32 PowerView GUI	330
TOOLBAR		331
TOOLBAR	Toggle toolbar	331
TYPE		332
TYPE	Display text file	332
UNARchive		333
UNARchive	Linux and Microsoft libraries	333
UNARchive.extract	Extract files from Linux library and Microsoft library	333
UNARchive.Show	Extract files from library and list them in window	334
UNARchive.Table	Display table of contents of library	334
UNPACK		335
UNPACK	Expand files (with LZW algorithm)	335
UNZIP		336
UNZIP	Expand GZIP archive file (with DEFLATE algorithm)	336
VERSION		337
VERSION	TRACE32 version information	337
VERSION.ENVIRONMENT	Display environment settings	337
VERSION.HARDWARE	Display hardware versions	338
VERSION.SOFTWARE	Display software versions	338
VERSION.ThirdPartyLicenses	Display third party license information	339
VERSION.view	Display window with version info	340
WELCOME		341
WELCOME	Welcome to TRACE32	341
WELCOME.CONFIG	Configure search paths for PRACTICE demo scripts	341
WELCOME.CONFIG.ADDDIR	Add a new script search path	342
WELCOME.CONFIG.FILTER	Set the script search filter	342
WELCOME.CONFIG.ReMoveDIR	Remove a script search path	342
WELCOME.CONFIG.RESet	Reset the script search configuration	342
WELCOME.CONFIG.state	Open the welcome config window	343
WELCOME.SCRIPTS	Open the script search window	344
WELCOME.STARTUP	Open the welcome window if not disabled	344
WELCOME.view	Open the welcome window	345
Window		346
Win	Window handling (size, position, font size, etc.)	346
WinBack	Generate background window	347
WinCLEAR	Erase windows	347
WinDEFaultSIZE	Apply a user-defined default size to windows	349
WinDuplicate	Allows to open an existing window again	350
WinExt	Generate external window	351
WinFIND	Search for text in window	351
WinFreeze	Generate frozen window	353

WinLarge	Generate window with large font	354
WinMid	Generate window with regular font	355
WinOverlay	Pile up windows on top of each other	355
WinPAGE	Window pages	356
WinPAGE.Create	Create and select page	356
WinPAGE.Delete	Delete page	357
WinPAGE.List	Display an overview of all pages and their windows	358
WinPAGE.REName	Rename page	359
WinPAGE.RESet	Reset window system	359
WinPAGE.select	Select page	359
WinPAN	Specify a window cut-out	360
WinPOS	Define window dimensions and window name	361
WinPrint	Print address or record range of a window	364
WinPRT	Hardcopy of window	365
WinResist	Generate a resistant window	366
WinRESIZE	New size for window	367
WinSmall	Generate window with small font	368
WinTABS	Specify widths of re-sizable columns	368
WinTOP	Bring window to top	369
WinTrans	Generate transparent window	370
ZERO		371
ZERO.offset	Set time reference	371
ZERO.RESet	Reset to original value	372
ZIP		372
ZIP	Compress files to GZIP archive (with DEFLATE algorithm)	372
Appendix A - Help Filters		373

History

- 13-Jul-2023 New command [CommandLineKEYS](#).
- 26-Jun-2023 ENCoding ANSI is replaced by **WINCP** using [EDIT.ENCoding](#) and [SETUP.FILETYPE.ENCoding](#) commands.
- 04-Jan-2023 New command [DIALOG.NOYES](#).
- 15-Sept-2022 Added second parameter to [SETUP.ASCIITEXT](#).
- 04-Aug-2022 New command [EDIT.ENCoding](#).
- 03-Jun-2022 Marked [SETUP.DropCoMmanD](#) command as deprecated and replaced by new command [SETUP.FILETYPE.DropCoMmanD](#).
- 26-May-2022 New command group: [SETUP.FILETYPE](#):
New commands [SETUP.FILETYPE.ENCoding](#) and [SETUP.FILETYPE.EXTension](#).
Command [SETUP.EXTension](#) was renamed to [SETUP.FILETYPE.EXTension](#).
- 27-Apr-2022 New command [PRinTer.CONFIG.HEADER](#).
Command [PRinTer.OFFSET](#) was renamed to [PRinTer.CONFIG.OFFSET](#).
Command [PRinTer.SIZE](#) was renamed to [PRinTer.CONFIG.SIZE](#).
- 28-Mar-2022 New commands: [SETUP.PDEBUG.BlockClose](#), [SETUP.PDEBUG.BlockPosition](#),
[SETUP.PDEBUG.MacroRESet](#), and [SETUP.PDEBUG.RESet](#).
- 28-Mar-2022 New commands: [SETUP.PDEBUG.ScriptParams](#), [SETUP.PDEBUG.TermScripts](#),
[SETUP.PDEBUG.WindowExternal](#), and [SETUP.PDEBUG.WindowOnTop](#).
- 17-Mar-2022 New command: [SETUP.PDEBUG](#).
- 31-Jan-2022 Added debugger time absolute to [AREA.view](#) window.
- 10-Jan-2022 New command: [EDIT.InsertText](#).
- 03-Jan-2022 New option **/PDEBUG** for the command [HISTory.Set](#).
- 06-Dec-2021 New command group [DIALOG.STORAGE](#).
New dialog programming commands: [INIT](#) and [SUBROUTINE](#).
- 06-Oct-2021 New command: [VERSION.ThirdPartyLicenses](#).

07-Jul-2021 New command: [MENU.Delete.NAME](#), and the menu command [NAME](#).

08-Jun-2021 Removed command: ABORT.

31-May-2021 New command: [OS.SetENV](#).

28-Jan-2021 New PCL file format for command [PRinTer.FILE](#).
Updated command [ComPare](#).

22-Jan-2021 New command: [PYthon.INSTALL](#).

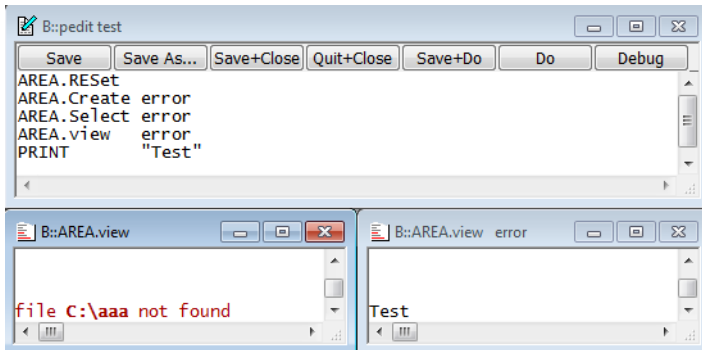
Message areas are the IN/OUT windows for error texts or print commands. They work like a standard scrolling terminal. All asynchronous error messages, which appear in the [message line](#), are written to the default message area (named **A000**), which can be displayed in the [AREA.view A000](#) window.

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 send 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:

```
WinPOS  ,,,,,,  myAreaWin  ;define a window name for an AREA window
AREA.view A000                ;and display the default message area A000
                                ;in that AREA window

PRinTer.EXPORT.ASCIIE C:\area.txt ;define file format and name
WinPAN      0 -999.  myAreaWin  ;scroll back to the first line of
                                ;the area window (for windows with
                                ;fewer than 1000. lines)
WinPRT      myAreaWin /ALL      ;/ALL prints all lines from the
                                ;visible top of the window to the end
```

See also

- [AREA.CLEAR](#)
- [AREA.CLOSE](#)
- [AREA.Create](#)
- [AREA.Delete](#)
- [AREA.List](#)
- [AREA.OPEN](#)
- [AREA.PIPE](#)
- [AREA.RESet](#)
- [AREA.SAVE](#)
- [AREA.Select](#)
- [AREA.STDERR](#)
- [AREA.STDOUT](#)

- [AREA.view](#)
- [ENTER](#)
- [LOG.toAREA](#)
- [SILENT](#)
- [AREA.EXIST\(\)](#)
- [AREA.NAME\(\)](#)
- [AREA.SELECTed\(\)](#)
- ▲ ['AREA Functions' in 'PowerView Function Reference'](#)
- ▲ ['Message Windows' in 'PowerView User's Guide'](#)

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](#)
- [AREA.Delete](#)
- [AREA.OPEN](#)
- [AREA.RESet](#)
- [PRINT](#)
- ▲ ['Message Windows' in 'PowerView User's Guide'](#)
- ▲ ['Release Information' in 'Legacy Release History'](#)

AREA.CLOSE

Close output file

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

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

<area_name> Without an *<area_name>*, all **AREA** output files will be closed.

Example: For an example, see [AREA.OPEN](#).

See also

- [AREA.CLEAR](#)
- [AREA.Create](#)
- [AREA](#)
- [AREA.OPEN](#)
- [AREA.RESet](#)
- [AREA.Select](#)
- ▲ ['Message Windows' in 'PowerView User's Guide'](#)

Format: **AREA.Create** [*<area_name>*] [*<columns>*] [*<lines>*]

Creates a new message area or modifies the number of columns and lines of an existing one. You may create up to 19 additional message areas.

<i><area_name></i>	The AREA name must not contain the following characters: * \ / ' " ; , & The AREA name is case sensitive.
without <i><area_name></i>	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.

Example:

```
AREA.Create A000 60. 100.      ; change number of columns and lines of
                               ; the default area

AREA.RESet                    ; init area system
AREA.Create XMESAGE 20. 20.   ; create new area named "XMESAGE"
AREA.view XMESAGE             ; open window for area "XMESAGE"
AREA.Select XMESAGE           ; 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 'PowerView User's Guide'
- ▲ 'Release Information' in 'Legacy Release History'
- ▲ 'I/O Commands' in 'Training Script Language PRACTICE'

Format: **AREA.Delete** *<area_name>*

Deletes the specified message area, which has previously been created with [AREA.Create](#), and closes the associated [AREA](#) window. You cannot delete the default message area **A000**.

- If there is no message area of the given name, then **AREA.Delete** will not show any error.
- If there are multiple **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](#).

Example: This script is for demo purposes only. To try this script, simply copy it to a `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

- [AREA](#)
- [AREA.CLEAR](#)
- ▲ ['Release Information' in 'Legacy Release History'](#)

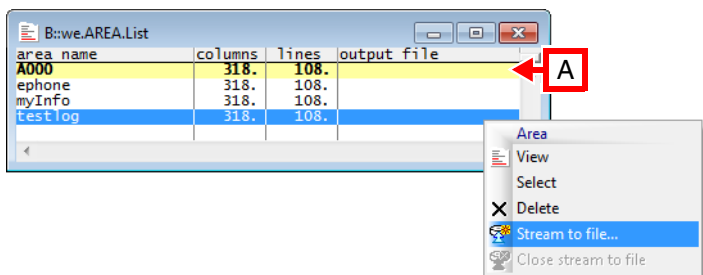
AREA.List

Display a detailed list off all message areas

[\[Example\]](#)

Format: **AREA.List**

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.



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 'Legacy Release History'](#)

Format: **AREA.OPEN** [*<area_name>*] *<file>* /*<option>*

<option>: **Create** | **Append** | **NoFileCache**

The outputs to the **AREA** window are saved in a file. The file can be closed with the **AREA.CLOSE** command.

<i><area_name></i>	<ul style="list-style-type: none"> Specify a user-defined <i><area_name></i>. Area names are created with the AREA.Create command. If the <i><area_name></i> is omitted, then AREA.OPEN refers to the default message AREA window named A000.
<i><file></i>	If the file with the specified <i><file></i> already exists, the file will be overwritten by default (same effect as option /Create).
<i><option></i>	The options are only available if you specify an <i><area_name></i> , 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.

Examples:

```
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.CLEAR](#)
[■ AREA.CLOSE](#)
[■ AREA.List](#)
[■ AREA.SAVE](#)
[■ AREA.Select](#)

▲ 'Message Windows' in 'PowerView User's Guide'

▲ 'Release Information' in 'Legacy Release History'

Format: **AREA.PIPE** [<pipe> | <area_name> [<file>]]

Redirects AREA to named pipe.

See also

■ [AREA](#)

AREA.RESet

Reset areas

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 'PowerView User's Guide'

▲ 'I/O Commands' in 'Training Script Language PRACTICE'

AREA.SAVE

Save AREA window contents to file

Format: **AREA.SAVE** [<area_name>] <file>

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

Example:

```
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 'PowerView User's Guide'](#)

AREA.Select

Select area

Format: **AREA.Select** [*<area_name>*]

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**.

Example:

```
AREA.RESet                ; init   area system
AREA.Create XMESSAGE 20. 20. ; create new area named "XMESSAGE"
AREA.view XMESSAGE        ; display 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](#)
- [AREA.CLOSE](#)
- [AREA.Create](#)
- [AREA.OPEN](#)
- [PRINT](#)
- ▲ ['Message Windows' in 'PowerView User's Guide'](#)
- ▲ ['I/O Commands' in 'Training Script Language PRACTICE'](#)

Format: **AREA.STDERR** [*<area_name>*]

Redirects AREA to stderr.

See also

■ [AREA](#)

Format: **AREA.STDOUT** [*<area_name>*]

Redirects AREA to stdout.

See also

■ [AREA](#)

Format: **AREA.view** [<area_name>]

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.

Example 1:

```
; initialize the area system
AREA.RESet

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

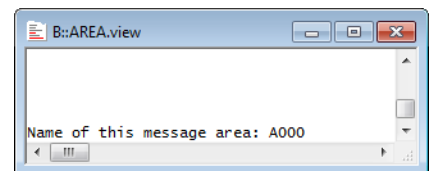
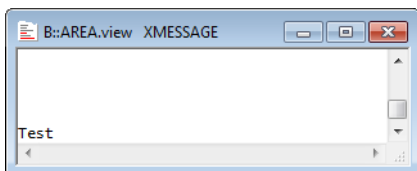
; create a new, user-defined message area named 'XMESSAGE'
AREA.Create XMESSAGE 20. 20.

; 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

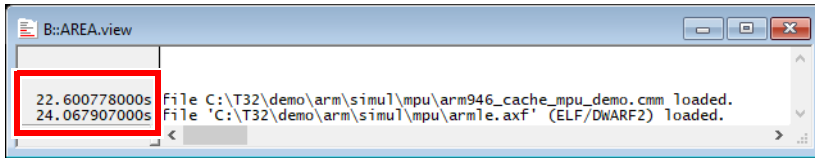
; print a string constant to the message area 'XMESSAGE' (see AREA win.)
PRINT "Test"

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



Starting from TRACE32 release 02.2022, **AREA** windows include the *debugger time absolute* (see **ZERO** command). The *debugger time absolute* is started with the first **SYStem.Up**.

Timestamps can be displayed by [scale area](#) of **AREA.view** window.



See also

- [AREA](#)
- [PRINT](#)
- ▲ ['Message Windows' in 'PowerView User's Guide'](#)
- ▲ ['I/O Commands' in 'Training Script Language PRACTICE'](#)

Format: **AutoSTOre** <file> [<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 <file>. The script is 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 <file>.

The **AutoSTOre** command should be used only once per TRACE32 session. Usually it is used within the PRACTICE script file `autostore.cmm` (which you should not edit), but you can also use it again in the PRACTICE script files `system-settings.cmm` (in the TRACE32 system directory) or `user-settings.cmm` (in the user settings directory, on Windows `%APPDATA%\TRACE32` or `~/trace32` otherwise).

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** at the end of a TRACE32 session.

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

<p><file> or ,</p>	<p>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 OS.ID() function and the string <code>store.cmm</code>.</p>
<p><item>, <option>, and <device_specific_settings></p>	<p>For a detailed description of <item>, <option>, and <device_specific_settings>, refer to the STOre command.</p>

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: 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**).

```
AutoSTOre , HISTory BookMark HELP
```

See also

- [ClipSTOre](#) ■ [SETUP.STOre](#) ■ [STOre](#)
- ▲ ['Window System' in 'PowerView User's Guide'](#)

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

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**. Insert the **placeholder [] for the bitmap** before you open the file in the **BITMAPEDIT** editor. For step-by-step procedures, see **"Icons"** in PowerView User's Guide, page 113 (ide_user.pdf).

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

```
MENU.ReProgram ; e.g. file addmybutton.cmm in  
( ; directory ~/demo/menu/  
  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.
```

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.

```
Format:          ChDir [<path>]

Format:          ChDir.DO <file> [<parlist>]
```

Changes or displays the current working directory. 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).

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
```

```
ChDir.Data.LOAD.Elf *
```

NOTE: If ChDir is used in front of a command, like ChDir.DO <file>, then the search paths defined with **PATH** are omitted.

See also

■ [DO](#)

■ [MKDIR](#)

■ [PWD](#)

□ [OS.DIR\(\)](#)

▲ 'File and Folder Operations' in 'PowerView User's Guide'

▲ 'Release Information' in 'Legacy Release History'

Format:	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 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
```

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](#)
- [SETUP.STOre](#)
- [STOre](#)
- ▲ ['Window System' in 'PowerView User's Guide'](#)

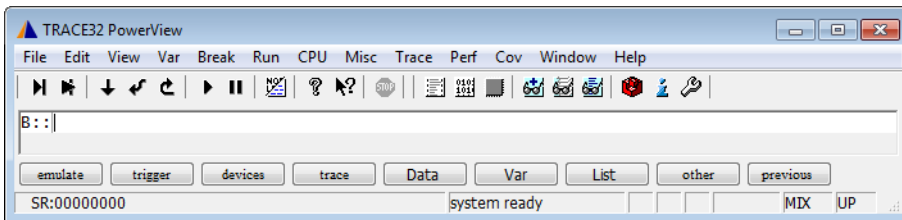
Format: **CmdPOS** <left> <up> <hsize> <vsize> [<item>] [<colormode>]

<item>: **Normal | Iconic | Maximized**

<colormode>: **Auto | DEFault | <colorindex>**

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 <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.



The work area is hidden.

The other TRACE32 windows float outside the main window.

- For more information about the user interface, see [“Graphical User Interface - Window Modes”](#) in PowerView User’s Guide, page 11 (ide_user.pdf).
- For an overview of the eight colors for cores, open the [SETUP.COLOR](#) window.

<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 main window size in cursor width or percentage (only valid for Normal)
<vsize>	Vertical main window size in cursor height or percentage (only valid for Normal)
Normal	The TRACE32 main window is positioned at the given x- and y-coordinate with the chosen horizontal and vertical size.
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 <code>CORE</code> variable within the configuration file (config.t32). If <code>CORE</code> is not set, then the default coloring is used.
DEFAult	Set default colors for toolbar and MWI background.
<code><colorindex></code>	Integer number between 0 and 7 to select a fixed background color for toolbar and MWI background.

Examples:

```
CmdPOS 10. 10. 70. 30. normal           ; Shows the TRACE32 main window
                                         ; including the work area
```

```
CmdPOS , , , 0. , normal               ; Hides the work area but shows the
                                         ; menubar, toolbar and command line
```

```
CmdPOS , , , , Iconic                  ; Minimized TRACE32 to an icon on
                                         ; the taskbar
```

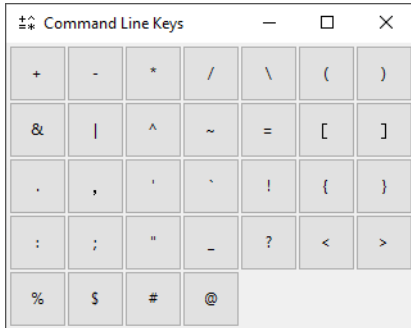
```
CmdPOS , , , , Auto                    ; CORE dependent toolbar color
```

See also

- [FramePOS](#)
- [SETUP.COLOR](#)
- [CORE.SHOWACTIVE](#)
- ▲ ['PowerView - Screen Display' in 'PowerView User's Guide'](#)
- ▲ ['Commands' in 'PowerView User's Guide'](#)
- ▲ ['Release Information' in 'Legacy Release History'](#)

Format: **CommandLineKEYS**

Opens a window to assist typing special characters into the command line.



Format: **ComPare** <file1> <file2> [/<option>]

<option>: **Case** | **IgnoreSpace** | **IgnoreCRLF**

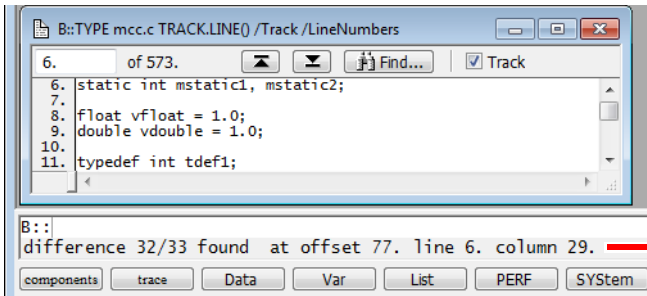
Compares two files on a byte-by-byte level. The **ComPare** command stops at the first difference. The different bytes are displayed, together with the position counted in bytes, in lines and columns. The result will be found in the **FOUND()** function. By comparing test results to reference files,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

- Case** Observe case sensitivity, i.e. upper and lower case characters are not the same.
- IgnoreSpace** Ignore any differences in white-spaces when comparing files. That means that the following characters are ignored: blank, tab, line-feed, carriage-return. The first found difference is reported for the first file parameter <file1>. If using this option the printed line and column result is influenced by the order of the given file parameters.
- IgnoreCRLF** Ignore any differences in the line endings. That means that the following characters are ignored: line-feed, carriage-return. The first found difference is reported for the first file parameter <file1>. If using this option the printed line and column result is influenced by the order of the given file parameters.

NOTE: The options can be combined since TRACE32 build. 130739 (R.2021.02).

Examples:

```
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\(\)](#)
- [□ TRACK.LINE\(\)](#)
- [▲ 'File and Folder Operations' in 'PowerView User's Guide'](#)
- [▲ 'Release Information' in 'Legacy Release History'](#)

COPY

COPY

Copy files

Format: **COPY** <source> <destination>

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

Examples:

```
COPY ~/per68302.t32 per68302.per
```

```
COPY text1.txt text1.old
```

DATE

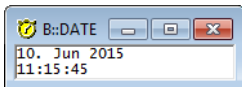
For architectures that do not have the **CLOCK** command group, **CLOCK** is an alias for **DATE**.

DATE

Display date and time

Format: **DATE**

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



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

Example:

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

See also

- [CLOCK](#)
- [CONVert.TIMEMSTOINT\(\)](#)
- [CONVert.TIMESTOINT\(\)](#)
- [CONVert.TIMEUSTOINT\(\)](#)
- [DATE.DATE\(\)](#)
- [DATE.TIME\(\)](#)
- ▲ ['DATE Functions' in 'PowerView Function Reference'](#)

DEL

DEL

Delete file

Format: **DEL** <*file*>

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

Example:

```
DEL "c:/t32/test.bak"
```

See also

■ [RM](#)

▲ ['File and Folder Operations' in 'PowerView 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`
and
- `~/demo/analyzer/trigger`

For information about dialog syntax, file types, built-in icons, return values, and PRACTICE macros inside dialog definitions, see [“Dialog Programming”](#) in PowerView User’s Guide, page 79 (`ide_user.pdf`).

For reference information, screen shots, and source code examples of the various dialog elements, see [“Dialog Definition Programming Commands”](#) in this manual.

See also

■ DIALOG.AREA	■ DIALOG.DIR	■ DIALOG.Disable	■ DIALOG.Enable
■ DIALOG.END	■ DIALOG.EXecute	■ DIALOG.File	■ DIALOG.MESSAGE
■ DIALOG.NOYES	■ DIALOG.OK	■ DIALOG.Program	■ DIALOG.ReProgram
■ DIALOG.SELect	■ DIALOG.Set	■ DIALOG.SetDIR	■ DIALOG.SetFile
■ DIALOG.STORAGE	■ DIALOG.view	■ DIALOG.YESNO	■ END

▲ [‘DIALOG Functions’](#) in [‘PowerView Function Reference’](#)

▲ [‘Release Information’](#) in [‘Legacy Release History’](#)

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.

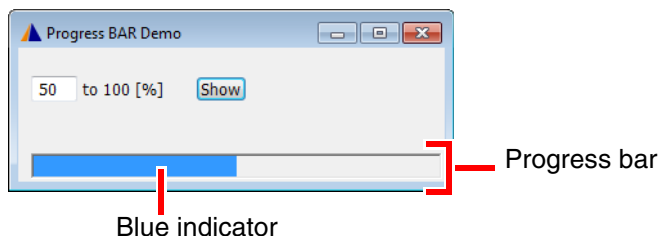
Commands which define a dialog element can have a label in front of the command. This label can be used to access the value of the dialog element.

The initial position and size of a custom dialog box on the screen can be governed with the [WinPOS](#) command.

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

Format: **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**.



Example 1: To reproduce the screenshot above, use this script.

```

LOCAL &IndicatorLength

DIALOG.view
(
  HEADER "Progress BAR Demo"

  ;define position (x, y) and horizontal width of the EDIT field
  POS 1. 1. 4.5
  myLabel:      EDIT "0" ""

  ;define descriptonal TEXT field
  POS 6. 1. 11.
  TEXT "to 100 [%]"

  ;define action button
  POS 17. 1. 5.
  DEFBUTTON "Show"
  (
    ;this action is executed when the 'Show' button is pressed
    &IndicatorLength=DIALOG.STRING(myLabel)+"."
    DIALOG.Set myProgressBar &IndicatorLength
  )
  ;define progress bar with a width of 40. units
  POS 1. 4. 40. 1.
  myProgressBar: BAR
)
STOP
DIALOG.END
ENDDO

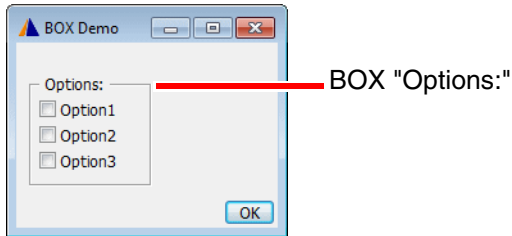
```

Example 2: An advanced demo script is included in your TRACE32 installation. To access the script, run this command:

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

Format: **BOX** ["<text>"]

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



Example: To reproduce the screenshot above, use this script.

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

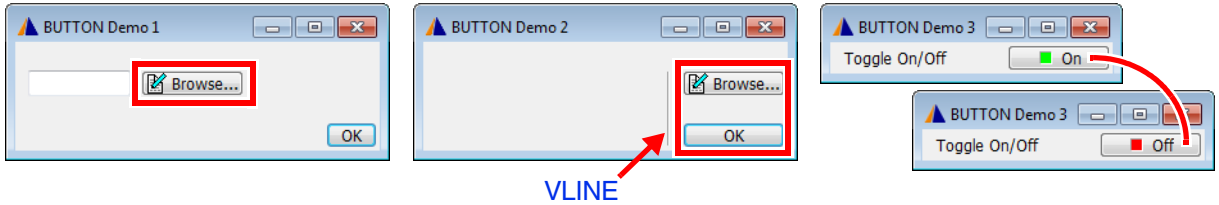
  POS 1. 1. 12. 5.
  BOX "Options:"

  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
```

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:

```
WinResist.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:

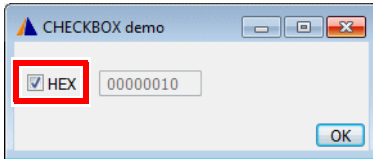
- **VLINE** has a height of **3.** units. To prevent the **BUTTON** from receiving the same height, we need to set the height of **POS** to **1.**
- The comma for **POS** and **DEFBUTTON** means that the value of the previous **POS** argument is used, i.e. height=1.

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

```
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
```

Format: **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.



Example 1:

```
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
```

Example 2: An advanced demo script is included in your TRACE32 installation. To access the script, run this command:

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

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

`<label>`: `<group_name>.<subname>`:

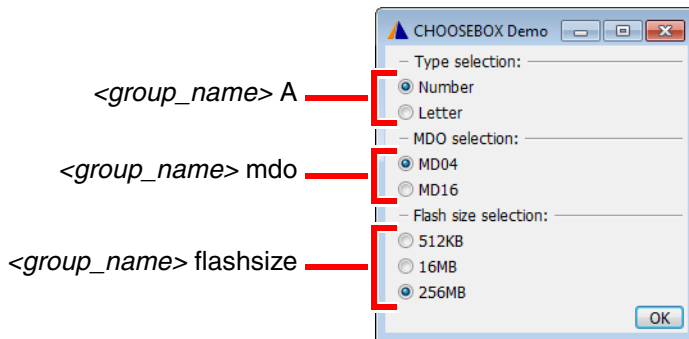
Defines a choose box item (radio button type). Normally a choose box is an element of a set/group of buttons, from which only one button can be active at any time.

NOTE: All choose boxes of one group must share the same label prolog / group name (e.g. "mdo.").

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.

Example 1: For the source code of this screenshot, refer to the script on the next page.



Example 2: 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
```

Example:

```
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
```

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.

Example:

```
LOCAL &label

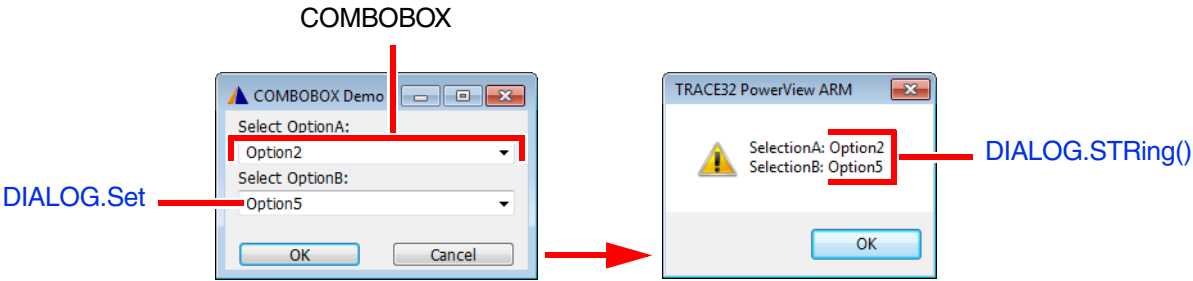
    DIALOG.view
    (
        POS 1. 1. 10.
LAB:    EDIT "" ""
        POS 1. 3. 5.
        DEFBUTTON "OK" "JUMPTO okclose"
        CLOSE "JUMPTO winclose"
    )
    STOP

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


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. You can set the current list item using the **DIALOG.Set** command. Use **DIALOG.STRING()** to retrieve the active list item.

Example:



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
```

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.

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.

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.

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.

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.

Format: **DEFMEDIT**" <initial_text>" [<command>]

Defines an **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: **DLISTBOX**" <list_items>" [<command>]

Defines a **LISTBOX** control where the list items can be rearranged by drag and drop.

Example:

DLISTBOX

Click and drag a list item.

DIALOG.STRING2() returns the current sequence of list items.

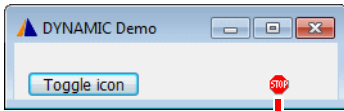
DIALOG.STRING() returns the selected list item.

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

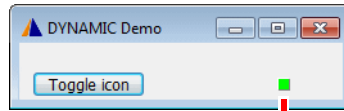
```
B::PSTEP ~/demo/practice/dialogs/dialog_dlistbox.cmm
```

Format: **DYNAMIC** "<initial_text>"

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



[:stop] icon



[:colorlime] icon

Example 1: To reproduce the screenshots above, use this script.

```

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

```

Example 2: 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
```

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

Defines a dynamic combo box which does not have the input focus when the dialog is opened.

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

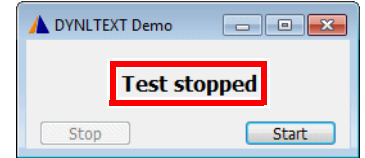
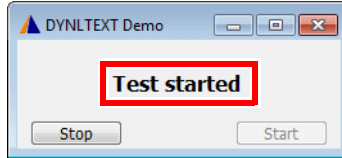
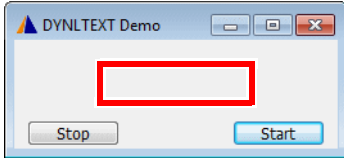
Define a dynamic hot combo box

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

Defines a dynamic hot combo box which does not have the input focus when the dialog is opened.

Format: **DYNLTEXT "<initial_text>"**

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.



Example: To reproduce the screenshots above, use this script.

```
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
```

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

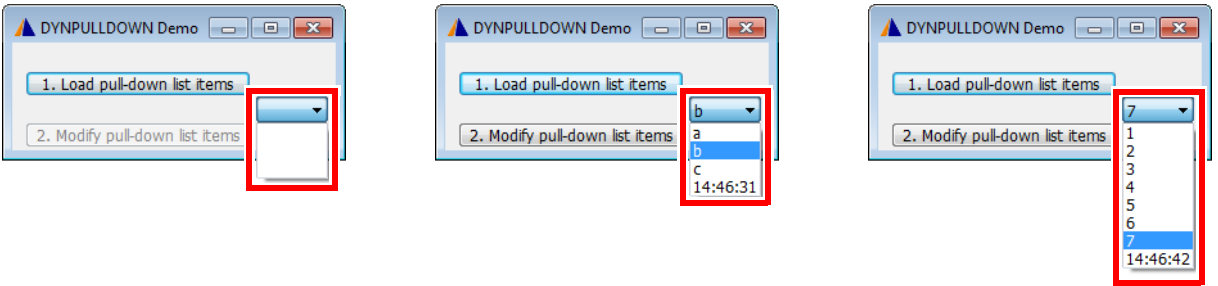
Defines a pull-down list that can be dynamically modified using **DIALOG.Set** while the dialog is open.

<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 DIALOG.STRING() function.
<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 PRACTICE script. For an example, see PULLDOWN .

Example: 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 **DIALOG.Set**, and not by **DYNPULLDOWN**. To reproduce this example, see source code below.



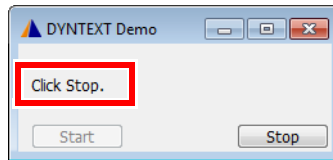
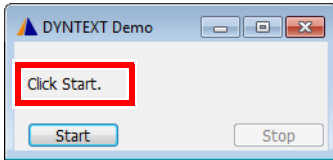
Source code for the above example:

```
DIALOG.view
(
HEADER "DYNPULLDOWN Demo"
POS 31. 2. 7.
; the pull-down list is initialized with three empty lines,
; one comma for each line
myEntries: DYNPULLDOWN ",,," ""
POS 1. 1. 25.
BTN1: DEFBUTTON "1. Load pull-down list entries"
( ; "b" is the value that is displayed first.
DIALOG.Set myEntries "b" "a,b,c,"+CLOCK.TIME()
DIALOG.Enable BTN2
)
POS 1. 3. 25.
BTN2: BUTTON "2. Modify pull-down list entries"
( ; "7" is the value that is displayed first.
DIALOG.Set myEntries "7" "1,2,3,4,5,6,7,"+CLOCK.TIME()
)
)
DIALOG.Disable BTN2 ;Deactivate the 2nd button temporarily
STOP
DIALOG.END
ENDDO
```

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.

Format: **DYNTEXT "<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.



Example: To reproduce the screenshots above, use this script.

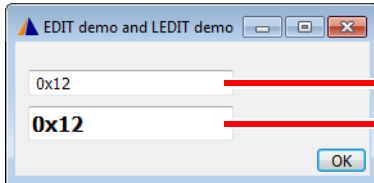
```
DIALOG.view
(
  HEADER "DYNTEXT Demo"

  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
```

Format: **EDIT** "<initial_text>" [<command>]

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.



```

LOCAL &Val1 ;initialize a PRACTICE macro

DIALOG.view
(
  HEADER "EDIT demo and LEDIT demo"

  POS 1. 1. 20.
  ;the Data.dump window opens when the cursor leaves the EDIT control
  ;after data entry
  myInput:      EDIT "0x12" "Data.dump"

  POS 1. 2.5 20.
  myInputB:     LEDIT "0x12"
  ( ;get the user input when the cursor leaves the LEDIT control
    &Val1=DIALOG.STRING(myInputB)
  )

  POS 29. 4. 5.
  DEFBUTTON "OK"
  (
    CONTinue
    DIALOG.END
    ;if "&Val1" is not empty, then show the user input in a message box
    IF "&Val1"!=" "
      DIALOG.MESSAGE "You have just entered: &Val1"
  )
)
STOP
PRINT "&Val1" ;print the user input to the message line
ENDDO

```

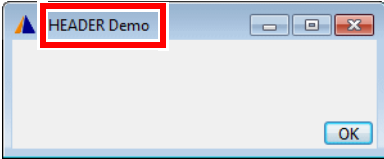
A more complex demo script is included in your TRACE32 installation. To access the script, run this command:

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

The **HOTEDIT** control executes the <command> for each character *while* you are typing.

Format: **HEADER "<text>"**

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.

```
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.

```
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
```

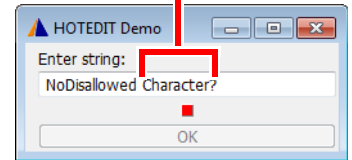
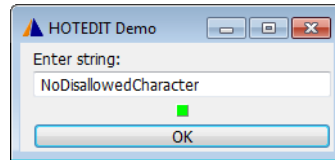
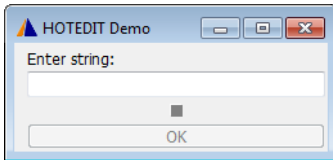
Format: **HELP** <*name*>

Format: **HOTEDIT** "<initial_text>" [<command>]

Defines an **EDIT** control. While a normal **EDIT** control executes <command> when the control loses input focus, **HOTEDIT** executes <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 <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: `B::CD.PSTEP ~/demo/practice/dialogs/dialog_hotedit.cmm`

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

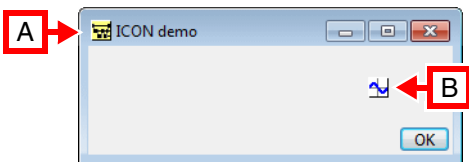
Defines a **COMBOBOX** control. While a normal **COMBOBOX** control executes <command> when the control loses input focus, **HOTCOMBOBOX** executes <command> whenever the control's text or selection changes.

ICON

New icon in top left corner of dialog

Format: **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**.

Example:

```
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"

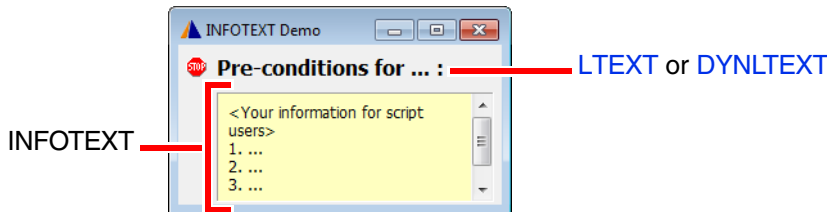
)
STOP
DIALOG.END
ENDDO
```

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

Format:	INFOTEXT "<msg_text>" [<background>] [<border_style>] [] [<scrollbar>] [<padding>]
<back ground>:	G Ray W hite L ightGray D arkGray S ticker
<border_style>:	N oBorder S imple S unken R aised
:	V ariable1 F ixed1 F ixed2 F ixed3 F ixed4
<scrollbar>:	H Scroll
<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.



<msg_text>	Max. length 2048 characters.
Default settings	If you omit all formatting options, then INFOTEXT is formatted with G Ray, N oBorder, V ariable1, and 0 by default.
H Scroll	<ul style="list-style-type: none"> If HScroll is included, the INFOTEXT box displays a horizontal scrollbar, and the automatic word wrap is turned <i>off</i>. If HScroll is omitted, the horizontal scrollbar is hidden, and the automatic word wrap is turned <i>on</i>. Your message text automatically adjusts to the width of the INFOTEXT box.

(all other formatting options)

A 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:

```
CD.PSTEP ~/demo/practice/dialogs/dialog_infotext.cmm
```

Example:

```
LOCAL &addTxt ;declare local macro

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

INIT

Initialize dialog

[build 142541 - DVD 02/2022]

Format: **INIT** ["<command>"]

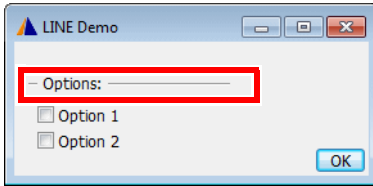
Executes a command or command block when the dialog window is opened. If the command string is omitted, the next line must begin with an open bracket to include a PRACTICE script. Use this block to initialize dialog elements, or to [set up and initialize dialog storage macros](#).

Example:

```
DIALOG.view
(
cb1: CHECKBOX "Option" ""
  INIT
  (
    ;checkbox cb1 is checked when dialog opens
    DIALOG.Set cb1 TRUE()
  )
)
```

Format: **LINE "<text>"**

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



Example:

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

  POS 1. 1. 20.
  LINE "Options:"

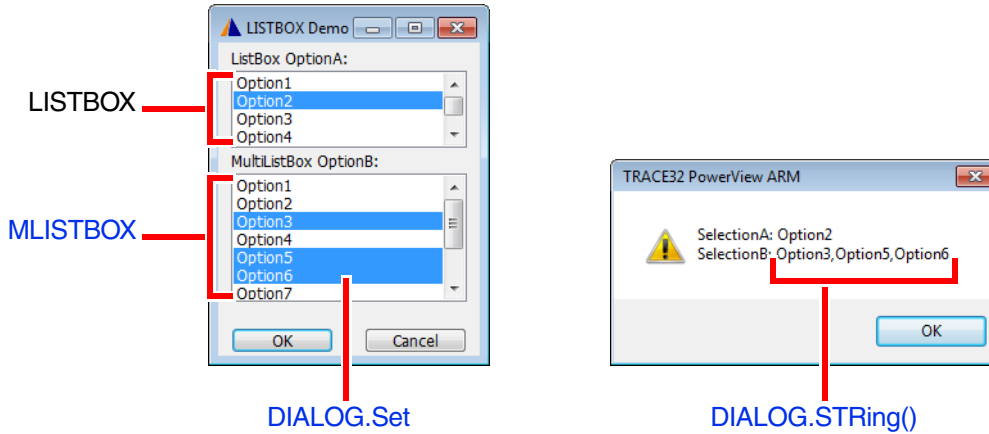
  POS 2. 2.25 10. 1.
  Option1: CHECKBOX "Option 1" ""
  Option2: CHECKBOX "Option 2" ""

  POS 29. 5. 5.
  DEFBUTTON "OK" "CONTinue"
)
STOP
DIALOG.END
ENDDO
```

Format: **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.

Example:

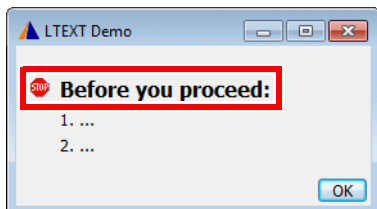


The demo script for the above example is included in your TRACE32 installation. To access the script, run this command:

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

Format: **LTEXT "<text>"**

Defines a 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.



Example: To reproduce the screenshot above, use this script.

```
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
```

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:

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.

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

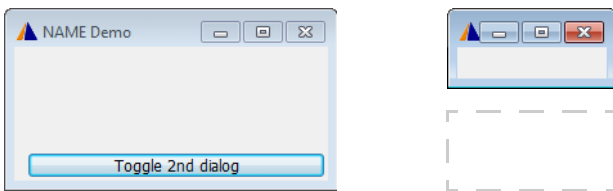
```
B::CD.PSTEP ~/demo/practice/dialogs/dialog_listbox.cmm
```

Format: **NAME "<text>"**

Defines an internal name for a dialog. Empty names are not allowed. 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 to bring a particular dialog to the front when it is hidden behind a lot of other open dialogs and windows.

Using the NAME element in a dialog will overwrite a previous name defined with the command **WinPOS**.

Example: 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.



```
DIALOG.view
(
  NAME "myDlg"
  HEADER "NAME Demo"

  POS 1. 4. 29.
  DEFBUTTON "Toggle 2nd dialog"
  (
    IF WINDOW.EXIST("myDlg2")==FALSE()
      GOSUB NextDialog
    ELSE
      DIALOG.SELECT myDlg2
      DIALOG.END
  )
)
STOP
DIALOG.END
ENDDO
;-----
NextDialog:
DIALOG
(
  NAME "myDlg2"
  HEADER "NAME Demo 2"
)
STOP
DIALOG.END
ENDDO
```

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.

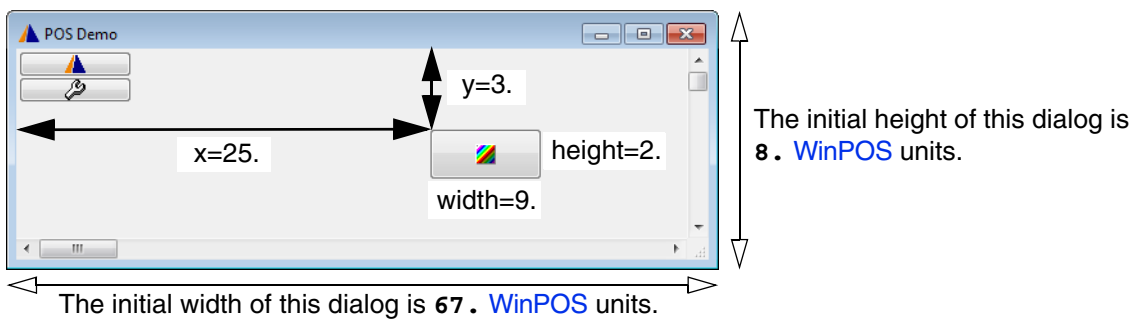
<x>	Max. <x> is 16383 . 5 units.
<y>	Max. <y> is 8191 . 75 units.
<width>	Max. <width> of an element is 16383 . 5 units.
<height>	Max. <height> of an element is 8191 . 75 units.
,	Value of the previous POS argument is used.
<no_argument>	Value of the previous POS argument is used, starting from right to left. In this example, the <height> and <width> of the previous POS are used for the unspecified <height> and <width>:
<pre> ; <x> <y> <width> <height> POS 3. 7. </pre>	

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 script 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.



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

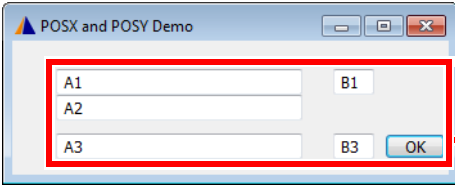
```
WinPOS , , 67. 8. ;limit the initial size of this large custom dialog
DIALOG.view
(
  HEADER "POS Demo"
  ;No POS command => default width is 9. and default height is 1. unit
  BUTTON "[:t32]" "PRINT ""This is a demo."" "
  ;No POS command => next element is advanced by 1. unit on the y-axis
  BUTTON "[:config]" ""

  ; <x> <y> <button_width> <button_height>
  POS 25. 3. , 2.
  BUTTON "[:colors]" ""

  ; <x> <y> <button_width> <no_argument>
  POS 500. 100. 10.
  BUTTON "OK" "CONTinue"
)
```


Format: **POSX** *<increment>* *<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**.



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.

Example:

```

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" ""
    ;
    POSY      1.5   <no_arguments>
myBox5: EDIT "B3" ""

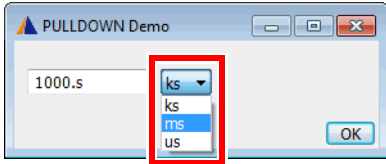
    POSX  1.           6.   1.
    DEFBUTTON "OK" "CONTinue"
)
STOP
DIALOG.MESSAGE DIALOG.STRING(myBox1) ;get value of EDIT box by label
DIALOG.END
    
```

Format: **POSY** *<increment>* *<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**.

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

Defines a static pull-down list.



<list_items>

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.

<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 PRACTICE script.

Example 1:

```
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.
  DEFBUTTON "OK" "CONTinue"
)
STOP
DIALOG.END
ENDDO
```

Example 2: 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
```

Format: **SPACE**

Applies the *<height>* of the previous **POS**, **POSX**, or **POSY** command to the next dialog element.

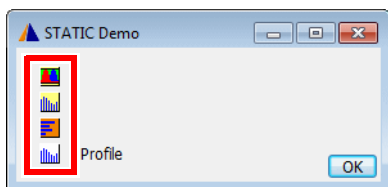
STATIC

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.



Example:

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

  ; x y width height
  POS 2. 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"`

Format: **SUBROUTINE**

Defines a subroutine of the DIALOG window. The subroutine is available for usage in all command blocks of the DIALOG program. The SUBROUTINE keyword must be succeeded by a command block.

Example:

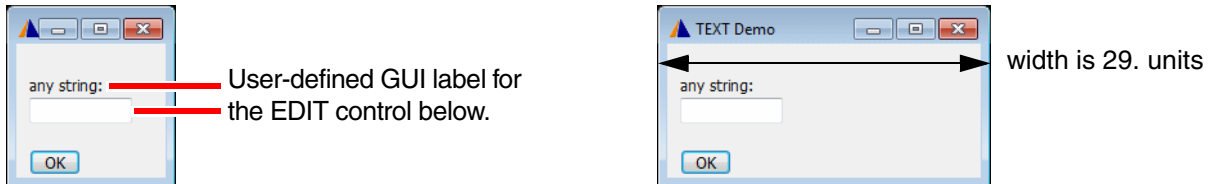
```
DIALOG.view
(
  BUTTON "Add 1"      "GOSUB Calc 1"
  BUTTON "Subtract 1" "GOSUB Calc -1"
nu: EDIT  "0"        ""

  INIT
  (
    DIALOG.STORAGE.define &value
    &value=0
    DIALOG.Disable nu
  )
  SUBROUTINE Calc
  (
    LOCAL &diff
    ENTRY &diff
    &value=&value+&diff
    DIALOG.Set nu FORMAT.DECIMAL(1.,&value)
  )
)
ENDDO
```

Format: **TEXT** "<text>"

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

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



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 "" ""

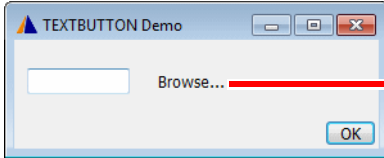
  POS 1. 4. 5.
  DEFBUTTON "OK" "CONTinue"
)
STOP
DIALOG.END
ENDDO
```

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

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

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.



Flat button with text only

Example:

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

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

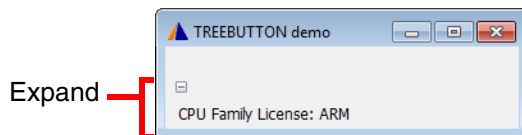
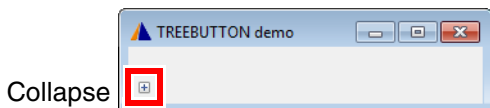
  POS 12. 1. 10.
  TEXTBUTTON "Browse..."
  (
    DIALOG.SetFile myLabel  ~/demo/practice/dialogs/*.cmm
  )

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

Format: **TREEBUTTON ""** [*<command>*]

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.



Example:

```

LOCAL &expand
&expand=0

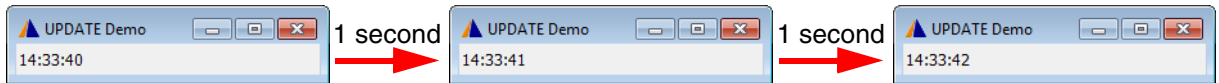
DIALOG.view
(
    NAME "myDemoDlg"
    HEADER "TREEBUTTON demo"

    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.
    DYNTEXT "CPU Family License: "+LICENSE.FAMILY(LICENSE.getINDEX())
)
WinRESIZE 35. 2. myDemoDlg ;Initial dialog size, collapsed
STOP
DIALOG.END
ENDDO
    
```

Format: **UPDATE** ["<command_string>"] [<update_interval>]

Executes commands periodically. The default update interval is one second. The <update_interval> cannot be interrupted. It is recommended that you comment out the **UPDATE** line before debugging such a PRACTICE script.

Example 1: Here, the **DIALOG.Set** command is parametrized with the **DATE.Time()** function to implement a timer on a dialog.



```
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 DATE.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 DATE.TIME()" 1.0s
)
STOP
DIALOG.END
ENDDO
```

Example 2: 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
```

Remember that the <update_interval> of **UPDATE** cannot be interrupted.

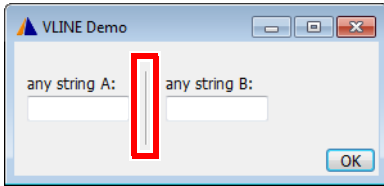
An alternative to **UPDATE** is **ON TIME**. To access the demo script, run this command:

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

ON TIME can be interrupted.

Format: **VLINE ""**

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



Example:

```

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

```

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

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



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

```

;*.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

```

```

;*.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: "+DATE.TIME()
    ;...<your_code>
)
;move button 6 units on the x axis
POSX 6.
StopBTN: BUTTON "Stop"
(
    PRINT "    Stopped at: "+DATE.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: The entire **DIALOG.AREA** block is embedded in the same *.cmm file, where the named **AREA** is created. This 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 'PowerView 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 must exist. The directory name can contain wildcard characters. The result of the selection is returned like the result value of a subroutine.

Example:

```
LOCAL &directoryname
DIALOG.DIR c:\t32                    ;use c:\t32 as start folder

;select the folder you want in the folder picker dialog

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 'PowerView User's Guide'](#)

Format: **DIALOG.Disable** <label>

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



<label>

User-defined label identifying a dialog element.

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 'PowerView User's Guide'

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](#).

<label>

User-defined label identifying a dialog element.

See also

■ [DIALOG](#)

■ [DIALOG.Disable](#)

■ [DIALOG.Set](#)

■ [DIALOG.view](#)

▲ ['Dialog Programming' in 'PowerView 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 'PowerView 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.

<label>

User-defined label identifying a dialog element.

See also

■ [DIALOG](#)

■ [DIALOG.view](#)

▲ ['Dialog Programming' in 'PowerView 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.

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

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.SetFile](#)

■ [DIALOG.File.SAVE](#)
■ [DIALOG.view](#)

■ [DIALOG.File.SELECT](#)

■ [DIALOG](#)

▲ 'Dialog Programming' in 'PowerView User's Guide'

Format: **DIALOG.File.open** <file>

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.)

Example: This script opens an OS file-open dialog with the title **Open my text file**. After you have selected a *.txt file, the dialog closes, and the script reads and prints the first line of the *.txt file it to the TRACE32 [message line](#). To try this script, copy it to a test.cmm file, and then run it in TRACE32 (See [“How to...”](#)).

```
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](#)

Format: **DIALOG.File.SAVE** <file>
 DIALOG.FileW <file> (deprecated)

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.

Example: This script opens an OS file-save dialog with the title **Save my text file**. After you have entered a file name and clicked **Save**, the dialog closes and the *.txt file is created. The script now writes "Hello World" to the newly-created *.txt file. To try this script, copy it to a test.cmm file, and then run it in TRACE32 (See ["How to..."](#)).

```
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 file name.

See also

- [■ DIALOG.File](#)
- [■ DIALOG.SetFile.SAVE](#)

Format: **DIALOG.File.SELECT** <file>

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](#)

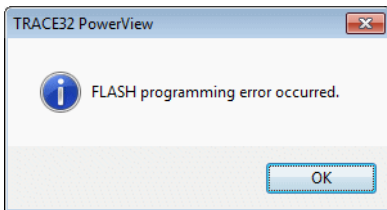
Format: **DIALOG.MESSAGE** <message>

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

Example:

```
; 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 'PowerView User's Guide'
- ▲ 'Release Information' in 'Legacy Release History'

Format: **DIALOG.NOYES** <message>

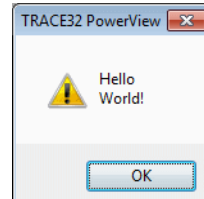
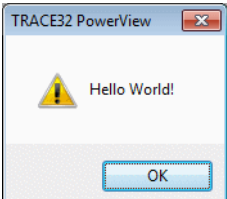
Similarly to the [DIALOG.YESNO](#) command, however the default button is No.

See also

- [DIALOG](#)
- [DIALOG.view](#)
- [DIALOG.YESNO](#)

Format: **DIALOG.OK** <message>

Creates a standard 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: **+CONVert.CHAR(0x0D) +**

```

DIALOG.OK "Hello"+CONVert.CHAR(0x0D)+"World!"           ; is carriage return
                                                    ; '\r'

DIALOG.OK "Hello"+CONVert.CHAR(0x0A)+"World!"           ; is line feed '\n'

DIALOG.OK "Hello"+CONVert.CHAR(0x0D0A)+"World!"         ; is carriage return
                                                    ; + line feed '\r\n'

DIALOG.OK "Hello" "World!" ; an empty space also creates a line break
    
```

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"\  

+CONVert.CHAR(0x0d) \  

    "and then switch ON the target board."

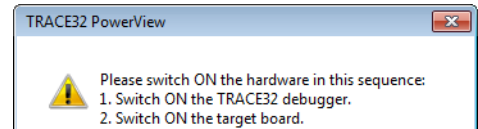
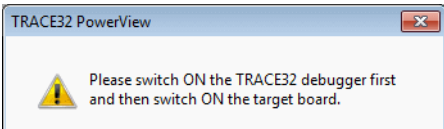
    DIALOG.OK "Please switch ON the hardware \  

    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](#)
- [FORMAT.Decimal\(\)](#)

DIALOG.Program

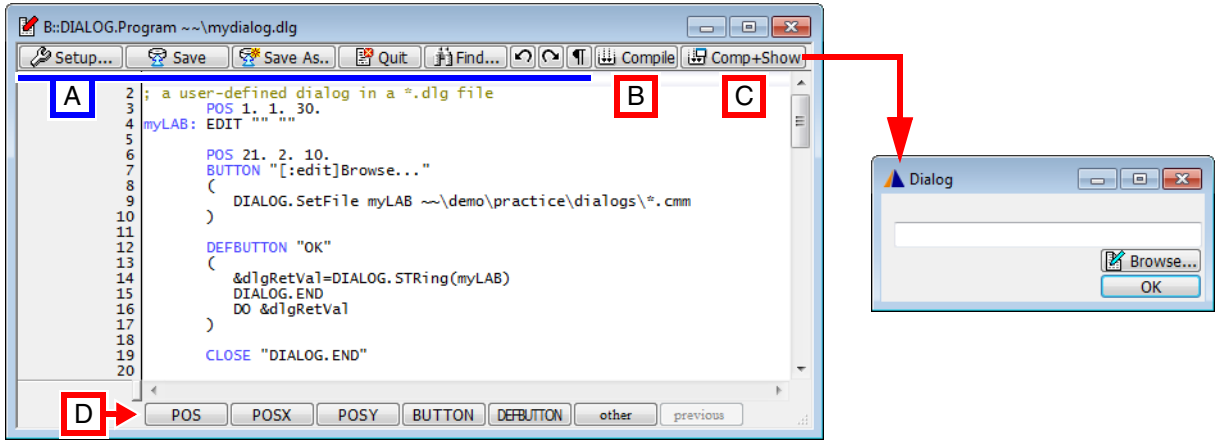
Interactive programming

Format: **DIALOG.Program** [*<file>*] [*<line>*] [*!<option>*]

<option>: **AutoSave | NoSave**

Opens the **DIALOG.Program** editor window, where you can create and edit dialog definition files for your own dialogs.

The editor provides syntax highlighting, configurable auto-indentation, and an online syntax check. The input is guided by softkeys. The [syntax for the dialog definition file](#) is described in the introduction to the **DIALOG** command group. You can view your dialogs with the **DIALOG.view** command.



Buttons common to all TRACE32 editors:

A For button descriptions, see [EDIT.file](#).

Buttons specific to this editor:

- B Compile** performs a syntax check and, if an error is found, displays an error message.
- C Compile+Show** performs a syntax check and, if the code is error free, displays the dialog.
- D** Commands for dialog programming. For descriptions and examples, refer to the [DIALOG](#) command group.

<i><file></i>	The default extension for <i><file></i> is *.dlg .
<i><line></i> , <i><option></i>	For description of the arguments, see EDIT.file .

See also

- [DIALOG](#)
 - [DIALOG.ReProgram](#)
 - [DIALOG.view](#)
 - [SETUP.EDITOR](#)
- ▲ 'Dialog Programming' in 'PowerView User's Guide'
 - ▲ 'Text Editors' in 'PowerView User's Guide'
 - ▲ 'Release Information' in 'Legacy Release History'

Format: **DIALOG.ReProgram** [*<file>*]

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.

```
DIALOG.ReProgram mydialog.dlg   ; opens dialog window

; <your_code>...
```

See also

- [DIALOG](#) ■ [DIALOG.Program](#) ■ [DIALOG.view](#)
- ▲ 'Dialog Programming' in 'PowerView User's Guide'
- ▲ 'Release Information' in 'Legacy Release History'

DIALOG.SELect

Programmatically focus on this dialog

Format: **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

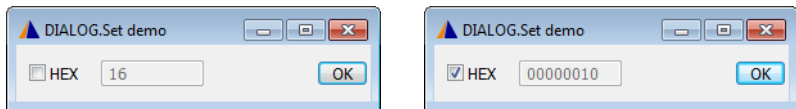
- [DIALOG](#) ■ [DIALOG.view](#)
- ▲ 'Dialog Programming' in 'PowerView User's Guide'

Format: **DIALOG.Set** <label> <value>

Dynamically changes the value or state of a dialog element while the dialog remains open.

<label>	User-defined label identifying a dialog element.
<value>	The value you want to dynamically assign to the dialog element. Type: <ul style="list-style-type: none"> • 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:

```

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: This script shows how you can set and toggle the state of **CHECKBOX** and **CHOOSEBOX** using **DIALOG.Set**. Simply copy the script to a `test.cmm` file, and then step through the script (See “**How to...**”).

```
DIALOG.view          ;examples for boolean elements
(
    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()
DIALOG.Set CHECK FALSE()
DIALOG.Set CHECK "ON"
DIALOG.Set CHECK "OFF"

;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.

```
DIALOG.view          ;examples for string elements
(
    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 'PowerView User's Guide'

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

```

DIALOG.view
(
    POS 1. 1. 20.
myLAB:  EDIT " " "

    POS 11. 2. 10.
    BUTTON "[:coloryellow]Folder"
    (
        DIALOG.SetDIR myLAB ~/demo/
    )

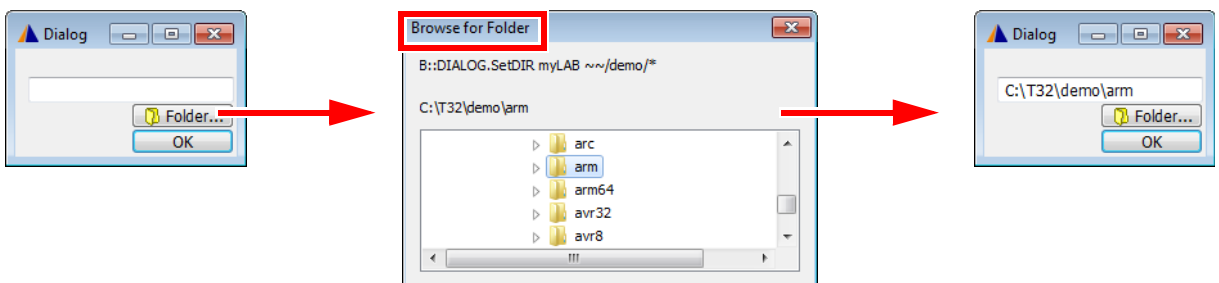
    DEFBUTTON "OK" "CONTinue"
)
STOP

&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

```



See also

■ [DIALOG.SetFile](#) ■ [DIALOG](#) ■ [DIALOG.DIR](#) ■ [DIALOG.view](#)

▲ 'Dialog Programming' in 'PowerView User's Guide'

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.

Dialog Type:	File open DIALOG.SetFile.open	File save DIALOG.SetFile.SAVE	File select DIALOG.SetFile.SELECT
Default button	Open	Save	OK
Existing file was chosen	accept	ask user if file should be replaced	accept
Non-existing file was chosen	reject (file must exist)	accept	accept

See also

- [DIALOG.SetFile.open](#) ■ [DIALOG.SetFile.SAVE](#) ■ [DIALOG.SetFile.SELECT](#) ■ [DIALOG.SetDIR](#)
 - [DIALOG](#) ■ [DIALOG.File](#) ■ [DIALOG.view](#)
- ▲ 'Dialog Programming' in 'PowerView User's Guide'

DIALOG.SetFile.open

OS file-open dialog > file name > EDIT element

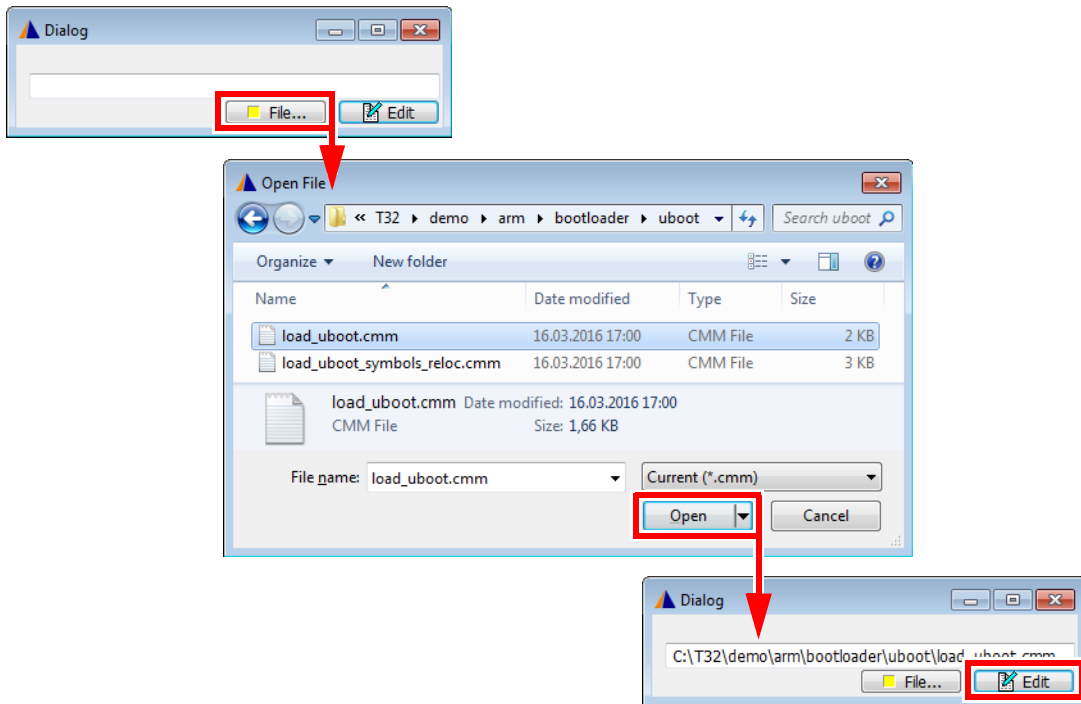
[\[Example\]](#)

Format: **DIALOG.SetFile.open** <label> <file>

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 run, simply copy and paste the PRACTICE script example into the TRACE32 command line.

```
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/*.cmm"
    )

    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](#)

Format: **DIALOG.SetFile.SAVE** <label> <file>
 DIALOG.SetFileW <label> <file> (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](#)

Format: **DIALOG.SetFile.SELECT** <label> <file>

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](#)

See also

■ [DIALOG](#)■ [DIALOG.view](#)**DIALOG.STORAGE.define**

Define macros stored in the dialog context

[build 142541 - DVD 02/2022]

Format: **DIALOG.STORAGE.define** <macro> [<macro> ...]

Defines the macros that will be available in the context of all dialog command blocks. This macros can be used to store data that is not stored in one of the dialog elements. Use in the [INIT block of the dialog program](#).

Example: The macro &starttime is defined to be part of the dialog context. It can be read and written in any of the dialog's command blocks.

```
DIALOG.view
(
  POS 0. 0. 40.
  STATIC "PLEASE READ:"
  STATIC "Important Information"
  BUTTON "Close" "GOSUB OnBtnClose"

  INIT
  (
    ;define macro &starttime to be part of dialog context
    DIALOG.STORAGE.define &starttime
    &starttime=OS.TIMER()
  )
  SUBROUTINE OnBtnClose
  (
    LOCAL &time
    ;macro &starttime is available in this subroutine
    &time=(OS.TIMER()-&starttime)/1000.
    PRINT "The dialog was opened for &time seconds"
    DIALOG.END
  )
)
ENDDO
```

Format: **DIALOG.STORAGE.LOAD**

Loads the macros stored in the context of the current dialog to the current PRACTICE stack. Used internally by PowerView in DIALOG programming.

Format: **DIALOG.STORAGE.LOAD**

Updates the macros stored in the context of the current dialog with the values of the macros on the current PRACTICE stack. Used internally by PowerView in DIALOG programming.

Format: **DIALOG.view** [*<file>*]

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

<file>

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

Example: The PRACTICE script file calls the dialog file, which contains an embedded PRACTICE script.

PRACTICE script file (*.cmm)

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

Dialog file (*.dlg)

```
;content of getfile.dlg  
  
POS 1. 1. 10.  
LAB: EDIT " " "  
POS 11. 1. 5.  
BUTTON "File"  
( ;embedded PRACTICE script  
  DIALOG.SetFile LAB *.cmm  
)  
POS 1. 3. 5.  
DEFBUTTON "OK" "DIALOG.END"  
CLOSE "DIALOG.END"
```

See also

- | | | | |
|------------------------------------|----------------------------------|----------------------------------|----------------------------------|
| ■ DIALOG | ■ DIALOG.AREA | ■ DIALOG.DIR | ■ DIALOG.Disable |
| ■ DIALOG.Enable | ■ DIALOG.END | ■ DIALOG.EXecute | ■ DIALOG.File |
| ■ DIALOG.MESSAGE | ■ DIALOG.NOYES | ■ DIALOG.OK | ■ DIALOG.Program |
| ■ DIALOG.ReProgram | ■ DIALOG.SELect | ■ DIALOG.Set | ■ DIALOG.SetDIR |
| ■ DIALOG.SetFile | ■ DIALOG.STORAGE | ■ DIALOG.YESNO | |

▲ ['Dialog Programming' in 'PowerView User's Guide'](#)

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.

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

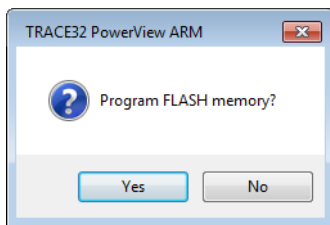
With **SETUP.QUITDO** you can define a PRACTICE script which will be executed before TRACE32 quits.

Example:

```
LOCAL &result

DIALOG.YESNO "Program FLASH memory?"
ENTRY &result
IF &result==FALSE()
    ENDDO

PRINT "User clicked Yes."
;... <your_code>
```



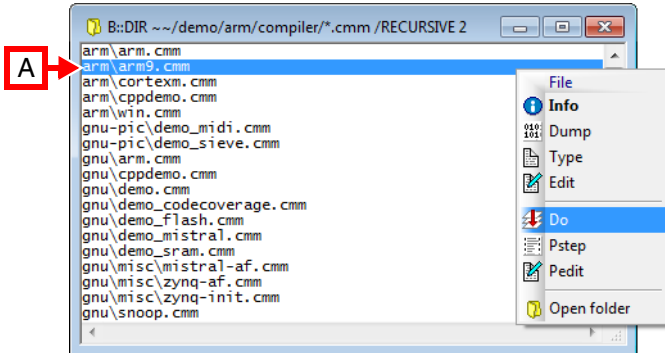
For information about line breaks and the line continuation character, see [DIALOG.OK](#).

See also

- [DIALOG](#)
 - [DIALOG.MESSAGE](#)
 - [DIALOG.NOYES](#)
 - [DIALOG.OK](#)
 - [DIALOG.view](#)
 - [FORMAT.Decimal\(\)](#)
- ▲ 'Dialog Programming' in 'PowerView User's Guide'
▲ 'Release Information' in 'Legacy Release History'
▲ '/O Commands' in 'Training Script Language PRACTICE'

Format: **DIR** [*<path>*] [/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.



A 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 <i><path></i> , this option includes the subdirectories and their files in the listing. If <i><depth></i> 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 TRACE32 [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 built-in 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.EDITTEXT](#)

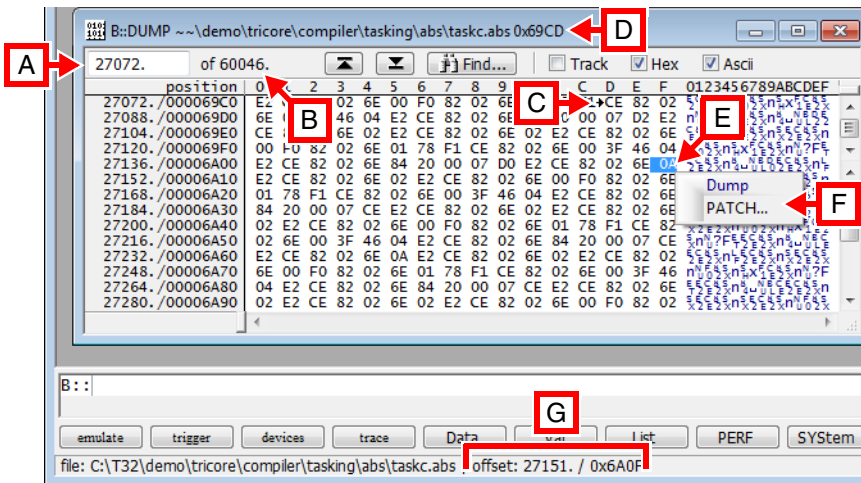
▲ ['File and Folder Operations' in 'PowerView User's Guide'](#)

▲ ['Release Information' in 'Legacy Release History'](#)

Format: **DUMP** [*<file>*] [*<offset>*] [*<option>* ...]

<option>: **NoHex** | **NoAscii** | **COLumns** *<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.



- A** Scroll to file offset.
- B** File size in bytes.
- C** A small black arrow indicates the content at the file offset.
- D** File offset entered in the command line.1
- E** Current selection.
- F** Right-click for popup menu.
- G** Offset of current selection in decimal and hex.

<i><offset></i>	File offset can be specified in decimal or hex.
NoHex	Hex display is switched off.
NoAscii	ASCII display is switched off.

<p>COLumns <columns></p> <p>WIDTH (deprecated)</p>	<p>Determines how many <columns> are displayed in the window.</p> <ul style="list-style-type: none"> To use the TRACE32 default setting, omit option and parameter. When you now resize the window width, the number of columns adjusts to the window width. COLumns without the <columns> parameter: The number of columns remains fixed when you resize the window width.
<p>Track</p>	<p>With Track enabled, the DUMP window tracks the selections you are making in the TYPE window.</p> <p>Prerequisite: The same file is open in both windows.</p>

Example 1:

```
;display file in hex and ASCII, start at file offset 1000 (hex)
DUMP mcc.abs 0x1000
```

Example 2:

```
Data.LOAD.Ieee mcc.abs

;let's now assume that the following error is displayed in the TRACE32
;message line: ERROR ENTRY NEAR OFFSET 1234. IN FILE mcc.abs

;display the file which caused the error
```

See also

■ [PATCH](#)

■ [TYPE](#)

■ [Data.dump](#)

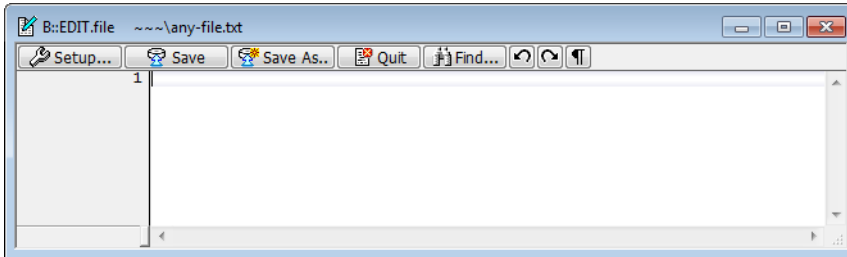
▲ ['File and Folder Operations' in 'PowerView User's Guide'](#)

See also

- | | | | |
|----------------|-----------------|---------------|-------------------|
| ■ EDIT.CLOSE | ■ EDIT.ENCoding | ■ EDIT.EXTErn | ■ EDIT.file |
| ■ EDIT.Find | ■ EDIT.FORMAT | ■ EDIT.Goto | ■ EDIT.InsertText |
| ■ EDIT.List | ■ EDIT.LOAD | ■ EDIT.OPEN | ■ EDIT.QUIT |
| ■ EDIT.REDO | ■ EDIT.Replace | ■ EDIT.REVERT | ■ EDIT.SAVE |
| ■ EDIT.SELect | ■ EDIT.UNDO | ■ PATCH | ■ SETUP.EDITEXT |
| ■ SETUP.EDITOR | | | |
- ▲ 'File and Folder Operations' in 'PowerView User's Guide'
 - ▲ 'Text Editors' in 'PowerView User's Guide'
 - ▲ 'Text Editors' in 'PowerView User's Guide'
 - ▲ 'Release Information' in 'Legacy Release History'

Overview EDIT

Using the commands in the **EDIT** command group, you can perform the basic editing operations that are common to all [TRACE32 editors](#), such as multiple undo/redo, find, replace, and goto operations.



For information about the editor feature highlights and a list of the editors for the various TRACE32-specific file types, please refer to "[PowerView User's Guide](#)" (ide_user.pdf).

NOTE:

Always use the editor pertaining to the respective TRACE32 file type. A special-purpose editor additionally provides commands that are specific to the file type you want to edit.

For example, use the PRACTICE script editor **PEDIT** to create and edit PRACTICE scripts (*.cmm). As opposed to the general-purpose editor **EDIT**, the special-purpose editor **PEDIT** additionally allows you to debug and execute your script.

The commands **EDIT.file** and **EDIT.OPEN** are specific to the general-purpose editor. We recommend that you use these two **EDIT.*** commands only if you want to create and edit text files, e.g. *.txt, *.log, *.dat, etc.

What is the difference between the commands...?

EDIT.file	EDIT.OPEN
<ul style="list-style-type: none">• Opens the file in the <i>general-purpose editor</i> of TRACE32.- or -• Opens the file in an <i>external editor</i> provided you have configured an external editor for use in TRACE32 with SETUP.EDITEXT.	<ul style="list-style-type: none">• Always opens the file in the <i>general-purpose editor</i> of TRACE32.• Regardless of whether you have configured an <i>external editor</i>.

What is the difference between the commands ...?

EDIT.CLOSE	EDIT.QUIT
<ul style="list-style-type: none">• Saves the file.• Removes the file from the editor buffer.	<ul style="list-style-type: none">• Discards unsaved changes.• Removes the file from the editor buffer.

What is the difference between the commands ...?

EDIT.LOAD	EDIT.REVERT
Reloads a file from the host system. <ul style="list-style-type: none">• Affects only the file specified by name.- or -• Affects all files if no file name is specified.	Reverts all changes performed since the last load or save operation. <ul style="list-style-type: none">• Affects only the active editor window and the file displayed in that window.• You cannot specify a file name at all.

EDIT.CLOSE

Close a text file

Format: **EDIT.CLOSE** [*<file>*]

Saves the *<file>* and removes it 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 saved and closed.

NOTE: For a comparison of the commands **EDIT.CLOSE** and **EDIT.QUIT**, refer to the introduction to the **EDIT** command group.

Examples:

```
EDIT.CLOSE test.txt          ; close one file
```

```
EDIT.CLOSE                   ; close all files opened by an EDIT command
```

See also

■ [EDIT](#)

■ [EDIT.file](#)

■ [EDIT.QUIT](#)

EDIT.ENCoding

Change the file encoding

[build 150073 - DVD 09/2022]

Format: **EDIT.ENCoding** <encoding>

<encoding>: **WINCP | UTF-8 | UTF-8-BOM**

Allows to define the file encoding of the currently edited file. The new encoding becomes active when the file is saved.

WINCP

The file will be saved encoded in the Windows code-page for non-unicode programs. This option is only available for Windows.

UTF-8

The file will be saved in UTF-8 encoding. If PowerView saves or modifies the file, it will add the UTF-8 BOM at the beginning of the file, only if the original file also included a BOM.

UTF-8-BOM

The file will be saved in UTF-8 encoding. If PowerView saves or modifies a file, it will add the UTF-8 BOM at the beginning of the file.

See also

■ [EDIT](#)

■ [EDIT.file](#)

Format: **EDIT.EXtern** *<file>* [*<line>*]

Opens a file in an external editor - but only on condition you have configured an external editor with the command **SETUP.EDITTEXT OFF**.

NOTE: Alternatively, you can configure an external editor with **SETUP.EDITTEXT ON**. In this case, you need to use the **EDIT.file** command to open the file in the external editor.

Examples:

```
EDIT.EXtern my.txt                   ; opens the file my.txt at line 1
```

```
EDIT.EXtern main.c 123.             ; opens the file main.c at line 123
```

See also

■ [EDIT](#)

■ [EDIT.file](#)

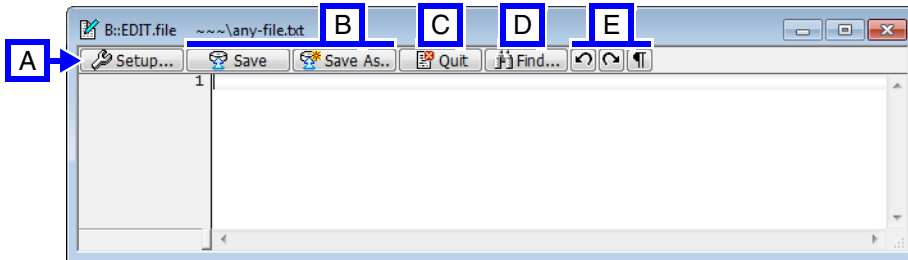
■ [SETUP.EDITTEXT](#)

Format: **EDIT.file** [*<file>*] [*<line>*] [*/<option>*]

<option>: **AutoSave** | **NoSave**

Depending on your TRACE32 configuration, **EDIT.file** opens the specified file either in the general-purpose editor of TRACE32 or in an external editor. The external editor is called if you have configured an external editor with the command **SETUP.EDITTEXT ON**.

The general-purpose editor is primarily used to create and edit text files, e.g. *.txt, *.log, *.dat, etc.



Buttons common to all TRACE32 editors:

- A** You can configure the editor settings, such as auto-indentation with spaces or tabs, using command group **SETUP.EDITOR**.
- B** Save file / Save file with a new name (**EDIT.SAVE**).
- C** Close window. A message box prompts you to save unsaved changes. The file is removed from the editor buffer.
- D** Open the **EDIT.Find** dialog for find, replace, and goto operations.
- E** Undo (**EDIT.UNDO**). Redo (**EDIT.REDO**). Toggle Show All (**SETUP.EDITOR.Mode switch**).

NOTE: For a comparison of the commands **EDIT.file** and **EDIT.OPEN**, refer to the introduction to the **EDIT** command group.

(no option)	Regular file-open operation. For an explanation of why the options AutoSave and NoSave are now deactivated and how you can re-activate them, see below .
<i><line></i>	The insertion point is placed into the specified line.

AutoSave	Opens a file which is saved automatically as soon as you click outside the editor window.
NoSave	<p>Opens a file in the <i>TRACE32 read-only mode</i>. This means:</p> <ul style="list-style-type: none"> • The file cannot be saved to disk because the Save button in the TRACE32 editor window is disabled. • However, you can edit and save the file in an external editor while the same file is still open in a TRACE32 editor. • You can load the file you have modified in the external editor back into the TRACE32 editor window: <ol style="list-style-type: none"> 1. Click inside the TRACE32 editor window. 2. Confirm with Yes when TRACE32 displays a message box, prompting you to reload the modified file; else click No.

Why are the softkey buttons for the options AutoSave and NoSave deactivated?

TRACE32 grays out the softkeys of both options **AutoSave** and **NoSave** in the following scenario:

- You are closing the *editor window* of a file via the **x** button. In your next step, you want to re-open the window and include one of the two options - while the file is still in the *editor buffer*. This would result in a conflicting option settings for the file in question. To prevent that, TRACE32 deactivates both options.
- By clicking the **x** button of an editor window, you are closing only the editor window. The file itself remains in the editor buffer until *you* remove the file from the editor buffer. To view the files in the editor buffer, use the **EDIT.List** window.

How do I remove a file from the editor buffer?

Click the **Quit** button in the editor window. This removes the file from the editor buffer + closes the editor window. Alternatively, use the command **EDIT.QUIT** *<file>* or **EDIT.CLOSE** *<file>* to remove the file from the editor buffer.

This reactivates the options **AutoSave** or **NoSave**. You can now (re-)open the same file and apply one of the options - if you want to.

Examples

Example 1: This script line performs a regular file-open operation for the specified file. The softkey buttons of the options **AutoSave** and **NoSave** are deactivated.

```
EDIT.file ~/my-todos.txt
```

```
;to re-activate the softkey buttons for the options AutoSave and NoSave,  
;you need to remove the file from the editor buffer like this:  
EDIT.CLOSE ~/my-todos.txt
```

For information about why the options are deactivated in a regular file-open operation, see [EDIT.file](#).

Example 2: The file that is opened in this script is saved automatically whenever you click outside the editor window.

```
;let's open the file with the AutoSave option.  
EDIT.file ~/my-todos.txt /AutoSave
```

See also

- | | | | |
|----------------|----------------|-----------------|-------------------|
| ■ EDIT | ■ EDIT.CLOSE | ■ EDIT.ENCoding | ■ EDIT.EXTerN |
| ■ EDIT.Find | ■ EDIT.FORMAT | ■ EDIT.Goto | ■ EDIT.InsertText |
| ■ EDIT.List | ■ EDIT.LOAD | ■ EDIT.OPEN | ■ EDIT.QUIT |
| ■ EDIT.REDO | ■ EDIT.Replace | ■ EDIT.REVERT | ■ EDIT.SAVE |
| ■ EDIT.SELect | ■ EDIT.UNDO | ■ PEDIT | ■ SETUP.EDITEXT |
| ■ SETUP.EDITOR | ■ TYPE | | |

This command is only available if [SETUP.EDITOR.TYPE](#) is set to **PowerView**.

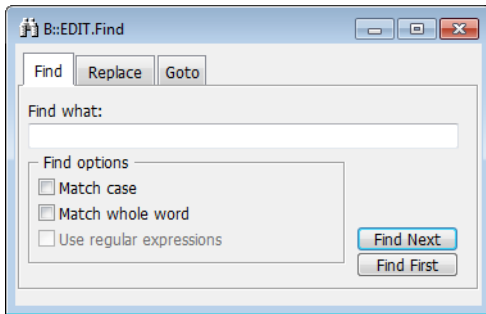
[\[Example\]](#)

Format: **EDIT.Find** [*<string>*] [*/<option>*]

<option>: **Case** | **Word** | **REGEX** *<expression>* | **Replace** *<string>* | **ReplaceAll** *<string>*
 | **FOCUS** | **HANDLE** *<handle>*



With arguments: The command performs a find, replace or goto operation in the selected TRACE32 editor window.

Without arguments: The command opens the **EDIT.Find** dialog window, which is available in all TRACE32 editors.



If you have opened multiple editor windows, you need to control in which editor window you want to perform the find, replace or goto operation. To do this, you can:

- Click inside the editor window you want, and then execute the command.
- Work with user-defined window names. For more information, see [WinPOS](#) and [WinTOP](#).
- Use the **HANDLE** option.

FOCUS	<p>With FOCUS: After the find, replace or goto operation, the focus is placed on the editor window and is clearly visible to the user:</p>  <p>Without FOCUS: The window caption remains dimmed.</p> 
HANDLE	<p>File handle. The numerical value of the file handle identifies the editor window and the file in which you want to perform a find, replace or goto operation.</p> <p>You can find the file handle in the handle column of the EDIT.List window.</p>

Example:

```
;open a file in the PEDIT window. This window has the focus because we  
;have just opened it.
```

```
PEDIT ~~~/my_script.cmm
```

```
;for demo purposes, let's correct the camel casing of the ENDDO command  
EDIT.Find "enddo" /Case /Word /ReplaceAll "ENDDO" /FOCUS
```

See also

■ [EDIT](#)

■ [EDIT.file](#)

■ [EDIT.Goto](#)

■ [EDIT.Replace](#)

This command is only available if **SETUP.EDITOR.TYPE** is set to **PowerView**.

Format:	EDIT.FORMAT [<i><line></i> <i><range></i>] /<options>
<i><range></i> :	<i><start_line></i> -- <i><end_line></i>
<options>:	CamelCase NoIndent

Formats the file contents as per the format settings in the **SETUP.EDITOR.state** window. Without arguments, the entire file is formatted. Alternatively, right-click a selection in an editor window, and then select **Format Selection** from the popup menu.

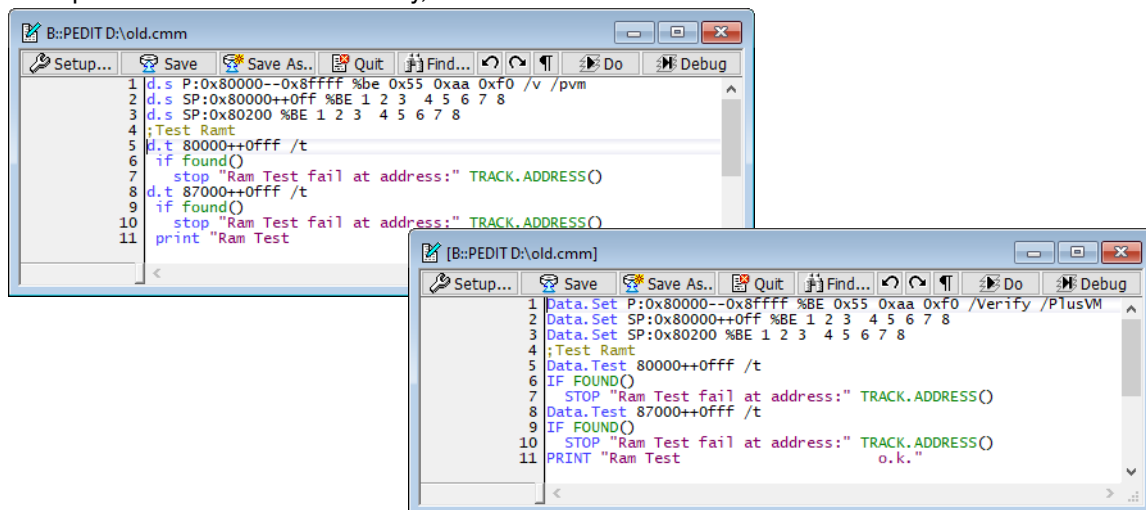
Available for **PEDIT**, **MENU.Program** and **PER.Program**.

<i><line></i>	Formats the entire file starting at the specified <i><line></i> .
<i><range></i>	Formats only the specified range of <i><start_line></i> -- <i><end_line></i> .

Following options are available:

Beautify	Convert all PRACTICE commands, functions, keywords and options into their official camel-cased form. See example below.
CamelCase (obsolete)	
NoIndent	Do not change indentation, intended for use with option /Beautify to keep non-standard indentation.

Example for EDIT.FORMAT /Beautify, before and after



The image shows two screenshots of a debugger window titled "B::PEDIT D:\old.cmm". The top screenshot shows the original code with inconsistent spacing and indentation. The bottom screenshot shows the same code after being formatted, with consistent indentation and spacing.

```
1 |d.s P:0x80000--0x8ffff %be 0x55 0xaa 0xf0 /v /pvm
2 |d.s SP:0x80000++0fff %BE 1 2 3 4 5 6 7 8
3 |d.s SP:0x80200 %BE 1 2 3 4 5 6 7 8
4 |Test Ramt
5 |d.t 80000++0fff /t
6 |if found()
7 |stop "Ram Test fail at address:" TRACK.ADDRESS()
8 |d.t 87000++0fff /t
9 |if found()
10|stop "Ram Test fail at address:" TRACK.ADDRESS()
11|print "Ram Test"
```

```
1 |Data.Set P:0x80000--0x8ffff %BE 0x55 0xaa 0xf0 /Verify /PlusVM
2 |Data.Set SP:0x80000++0fff %BE 1 2 3 4 5 6 7 8
3 |Data.Set SP:0x80200 %BE 1 2 3 4 5 6 7 8
4 |Test Ramt
5 |Data.Test 80000++0fff /t
6 |IF FOUND()
7 |STOP "Ram Test fail at address:" TRACK.ADDRESS()
8 |Data.Test 87000++0fff /t
9 |IF FOUND()
10|STOP "Ram Test fail at address:" TRACK.ADDRESS()
11|PRINT "Ram Test o.k."
```

See also

- [EDIT](#)
- [EDIT.file](#)
- [SETUP.EDITOR](#)

EDIT.Goto

Go to specified line

This command is only available if [SETUP.EDITOR.TYPE](#) is set to **PowerView**.

Format: **EDIT.Goto** [*<line>* *<column>* [*/<option>*]]

<option>: **FOCUS | HANDLE**

With arguments: The command performs a goto operation to the specified line in a TRACE32 editor window.

Without arguments: The command opens the dialog window for find, replace and goto operations, bringing the **Goto** tab to the front.

<option>

For a description of the options, see [EDIT.Find](#).

See also

- [EDIT](#)
- [EDIT.file](#)
- [EDIT.Find](#)
- [EDIT.Replace](#)

Format:	EDIT.InsertText <string> [/<option>]
<option>:	FOCUS NewLine HANDLE <handle>

This command is used to insert text into a document opened in the editor.

PowerView editor	The text is inserted at the current cursor position. If there is a text selection, then the selected text is replaced with the text specified with this command.
Native editor	The text is inserted at the end of the document.

<option>	For a description of the options, see EDIT.Find .
NewLine	Adds a new line after the inserted text.

See also

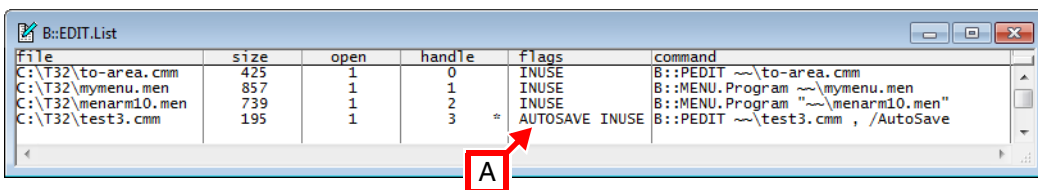
- [EDIT](#)
- [EDIT.file](#)

EDIT.List

List editor files

Format:	EDIT.List
---------	------------------

Lists all editor files that are in the editor buffer of TRACE32. Double-clicking a file name in this list opens the file for editing in the respective editor window; see **command** column.



A The **AutoSave** option saves a file automatically as soon as you click outside the editor window. For more information about how to use the option and its counterpart **NoSave**, see [EDIT.file](#).

See also

- [EDIT](#)
- [EDIT.file](#)

Format: **EDIT.LOAD** [*<file>*]

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.

NOTE: For a comparison of the commands **EDIT.LOAD** and **EDIT.REVERT**, refer to the introduction to the [EDIT](#) command group.

Example:

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

See also

■ [EDIT](#)

■ [EDIT.file](#)

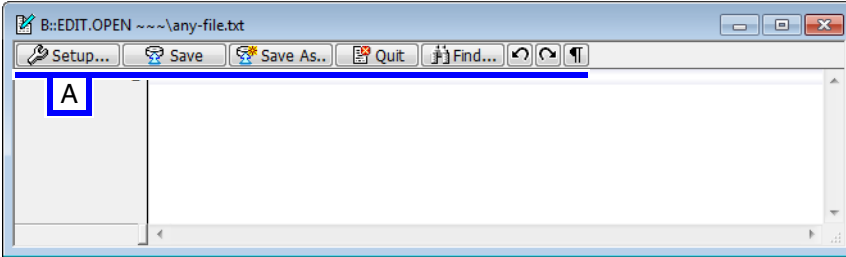
■ [EDIT.REVERT](#)

■ [EDIT.SAVE](#)

Format: **EDIT.OPEN** [*<file>*] [*<line>*] [*/<option>*]

<option>: **AutoSave** | **NoSave**

Always opens the specified file in the general-purpose editor of TRACE32 - regardless of whether you have configured an external editor with the command **SETUP.EDITTEXT ON** *<cmdline>*.



A For descriptions of the buttons that are common to all TRACE32 editors, see [EDIT.file](#).

<i><line></i> , <i><option></i>	For description of the arguments, see EDIT.file .
---	---

NOTE: For a comparison of the commands **EDIT.OPEN** and **EDIT.file**, refer to the introduction to the [EDIT](#) command group.

Examples:

```
; open file config.t32 for editing
EDIT.OPEN config.t32
```

```
; cursor is placed into line 50.
EDIT.OPEN C:\T32_MPC\menp4xxx.men 50.
```

```
; "*" allows to use the file browser to select the file
EDIT.OPEN *.c
```

See also

- [EDIT](#)
- [EDIT.file](#)
- [SETUP.EDITTEXT](#)
- [SETUP.EDITOR](#)

Format: **EDIT.QUIT** [*<file>*]

Removes the specified file from the editor buffer. All unsaved changes of the files are discarded.

If no file name is defined, all opened files within the editor buffer will be removed from the editor buffer.

NOTE: For a comparison of the commands **EDIT.CLOSE** and **EDIT.QUIT**, refer to the introduction to the **EDIT** command group.

Examples:

```
EDIT.QUIT test.txt                   ; don't save file test.txt
```

```
EDIT.QUIT                           ; ignore all changes in all text files
```

See also

■ [EDIT](#)

■ [EDIT.CLOSE](#)

■ [EDIT.file](#)

EDIT.REDO

Redo the previously undone edit/edits

This command is only available if [SETUP.EDITOR.TYPE](#) is set to **PowerView**.

Format: **EDIT.REDO** [**ALL** | *<count>*]

<option>: **FOCUS** | **HANDLE**

Redoes the previously undone edit in a TRACE32 editor.

ALL	Redoes all previously undone edits.
<i><count></i>	Specify the number of redo operations.
<i><option></i>	For a description of the options, see EDIT.Find .

See also

■ [EDIT](#)

■ [EDIT.file](#)

■ [EDIT.UNDO](#)

This command is only available if [SETUP.EDITOR.TYPE](#) is set to **PowerView**.

Format:	EDIT.Replace
---------	---------------------

Opens the dialog window for find, replace and goto operations, bringing the **Replace** tab to the front.

See also

■ [EDIT](#)

■ [EDIT.file](#)

■ [EDIT.Find](#)

■ [EDIT.Goto](#)

This command is only available if [SETUP.EDITOR.TYPE](#) is set to **PowerView**.

Format:	EDIT.REVERT [/<option>]
<option>:	FOCUS HANDLE

Reverts a file by loading the last saved file version from the host system back into the active editor window.

<option>	For a description of the options, see EDIT.Find .
----------	---

NOTE:	For a comparison of the commands EDIT.LOAD and EDIT.REVERT , refer to the introduction to the EDIT command group.
--------------	---

See also

■ [EDIT](#)

■ [EDIT.file](#)

■ [EDIT.LOAD](#)

EDIT.SELect

Select text/code in an editor window

This command is only available if [SETUP.EDITOR.TYPE](#) is set to **PowerView**.

Format:	EDIT.SELect <i><start></i> [<i><end></i>] [<i>!<option></i>]
<i><start></i> :	<i><line></i> [<i><column></i>]
<i><end></i> :	<i><line></i> [<i><column></i>]
<i><option></i> :	FOCUS HANDLE

Selects text/code in an editor window.

<option>

For a description of the options, see [EDIT.Find](#).

Examples:

```
EDIT.SELect 9. /FOCUS ;select entire line 9
EDIT.SELect 9. 3. /FOCUS ;select line 9 starting in column 3
EDIT.SELect 9. 4. 12. /FOCUS ;select from line 9, column 3
;up to and including line 12
EDIT.SELect 20. ,, 25. ,, /FOCUS ;select line 20 up to and including
;line 25.
```

See also

■ [EDIT](#)

■ [EDIT.file](#)

This command is only available if [SETUP.EDITOR.TYPE](#) is set to **PowerView**.

Format:	EDIT.UNDO [ALL <i><count></i>] [<i>!<option></i>]
<i><option></i> :	FOCUS HANDLE

Undoes the last edit in a TRACE32 editor.

ALL	Undoes all edits.
<i><count></i>	Specify how many edits you want to undo.
<i><option></i>	For a description of the options, see EDIT.Find .

See also

■ [EDIT](#)

■ [EDIT.file](#)

■ [EDIT.REDO](#)

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 'PowerView Function Reference'

Format: **Eval** <expression>

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

Example 1: 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

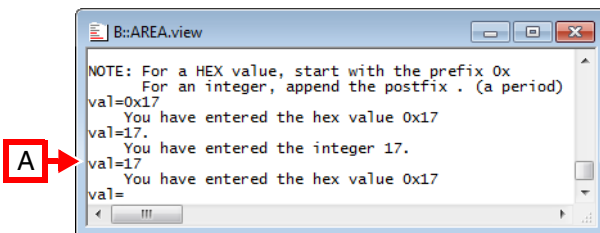
AREA.view                   ;open an AREA window

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
    PRINT "val="            ;your text for the user prompt
    ENTER &a                ;generate a user prompt in the AREA window
                             ;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.

Example 2:

```
Eval Register(pc)==1000           ; evaluate expression
                                   ...
IF (EVAL()!=0)                    ; use in other command
```

Example 3:

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

See also

- [Var.Eval](#)
- [EVAL.FLOAT\(\)](#)
- [EVAL.TYPE\(\)](#)
- ▲ ['Release Information' in 'Legacy Release History'](#)
- [EVAL\(\)](#)
- [EVAL.PARAM\(\)](#)
- [EVAL.ADDRESS\(\)](#)
- [EVAL.STRING\(\)](#)
- [EVAL.BOOLEAN\(\)](#)
- [EVAL.Time\(\)](#)

Format: **FIND** <file> [<offset> | <range>] [<items>] [/<options>]

<options>: **Back**
NoFind
NoCase

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: This script searches for a string in a file and, if the string is found, opens 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](#)
- [Data.GOTO](#)
- [TRACK.LINE\(\)](#)
- [TYPE](#)
- [Data.GREP](#)
- [WinFIND](#)
- [FOUND\(\)](#)
- [Data.Find](#)
- [TRACK.COLUMN\(\)](#)

- ▲ 'FOUND Functions' in 'PowerView Function Reference'
- ▲ 'File and Folder Operations' in 'PowerView User's Guide'

Format: **FramePOS** <left> <up> <hsize> <vsize> [<state>] [<colormode>]

<state>: **Normal** | **Iconic** | **Maximized** | **Top**

<colormode>: **Auto** | **DEFault** | <colorindex>

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.

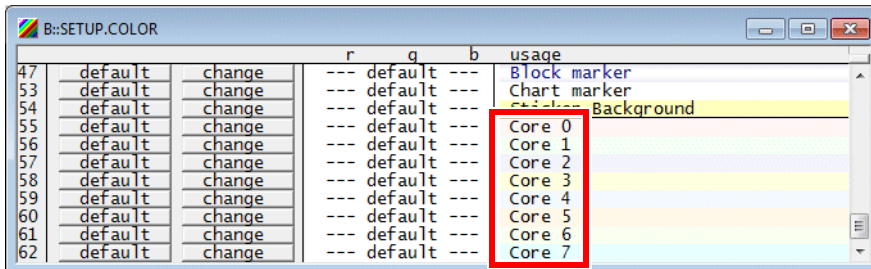
For more information about the user interface, see [“Graphical User Interface - Window Modes”](#) in PowerView User’s Guide, page 11 (ide_user.pdf).

<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	<p>The TRACE32 window is activated and positioned above all other top-level windows.</p> <p>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.</p>
Auto	<p>Color TRACE32 instance (MDI parent window) dependent on the CORE=<number> parameter in the config file.</p> <p>If the CORE parameter is not used, no coloring is done.</p> <p>This option is recommended for AMP systems.</p>
DEFault	<p>Set default colors for toolbar and MDI background.</p>
<colorindex>	<p>TRACE32 instance (MDI parent window) is colored as defined for the Cores 0 to 7 in the SETUP.COLOR window (see screenshot below).</p>

CORE parameter in config file:

```
PBI=
USB
CORE=2
```



Examples:

```
FramePOS , , , , Auto ; color TRACE32 instance dependent
; on the CORE parameter in the
; config file
; CORE=2 -> color of Core 1
```

```
FramePOS , , , , 1. ; color TRACE32 instance as
; specified for Core 1.
```

```
FramePOS 12.286 2.4167 90. 70. ; Position and size of the TRACE32
; main window specified by fixed
; values
```

```
FramePOS 33% 0% 33% 75% ; Position and size of the TRACE32
; main window specified by
; percentage
```

```
FramePOS , , , , Auto ; color TRACE32 instance (MDI
; parent window) dependent on
; the CORE=<number> parameter in
; the config file
; recommended for AMP systems
```

See also

■ [CmdPOS](#)

■ [SETUP.COLOR](#)

■ [WinExt](#)

■ [CORE.SHOWACTIVE](#)

▲ ['PowerView - Screen Display' in 'PowerView User's Guide'](#)

▲ ['Commands' in 'PowerView User's Guide'](#)

▲ ['Release Information' in 'Legacy 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](#)” in PowerView User’s Guide, page 90 (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

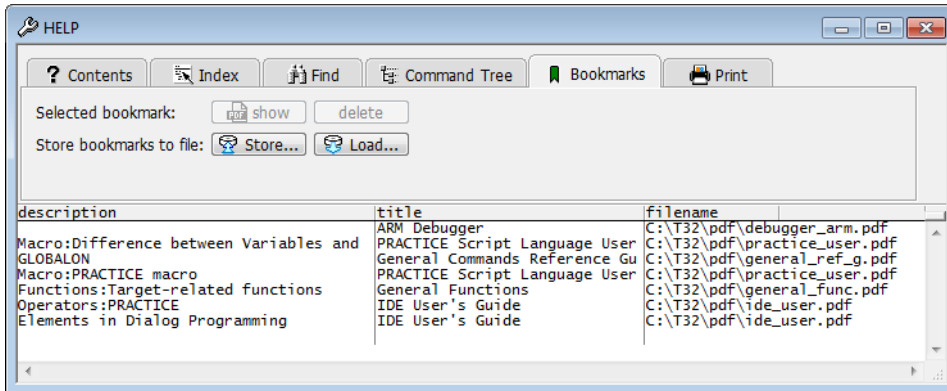
- | | | | |
|-----------------------------------|------------------------------------|--------------------------------|-------------------------------|
| ■ HELP.Bookmark | ■ HELP.checkUPDATE | ■ HELP.command | ■ HELP.FILTER |
| ■ HELP.Find | ■ HELP.Index | ■ HELP.OPEN | ■ HELP.PDF |
| ■ HELP.PICK | ■ HELP.PRinT | ■ HELP.Topics | ■ HELP.TREE |
| ■ SETUP.PDFViewer | | | |

▲ ‘HELP System’ in ‘PowerView User’s Guide’

▲ ‘Release Information’ in ‘Legacy 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 that is to modify your PRACTICE start-up script (*.cmm), so that help bookmarks are stored automatically.

See [“Store and Load Help Bookmarks Automatically”](#) in PowerView User’s Guide, page 93 (ide_user.pdf).

See also

■ [HELP.Bookmark.ADD](#)

■ [HELP.Bookmark.DELeTe](#)

■ [HELP.Bookmark.show](#)

■ [HELP](#)

See also

- [HELP.Bookmark.ADD.file](#)
- [HELP.Bookmark.ADD.Find](#)
- [HELP.Bookmark.ADD.Index](#)
- [HELP.Bookmark](#)
- [HELP.Bookmark.show](#)

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

Examples:

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

HELP.Bookmark.ADD CPUdata.pdf ; Additionally, start the search on
"Register Values" /Page 5 ; 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 'Legacy Release History'](#)

Format: **HELP.Bookmark.ADD.Index** *<file>* *<index>* [*<description>* *<title>*]

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:

Example:

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

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

See also

■ [HELP.Bookmark.ADD](#)

▲ ['Release Information' in 'Legacy Release History'](#)

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](#)

Format: **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.

Example:

```
HELP           ;Displays the table of contents

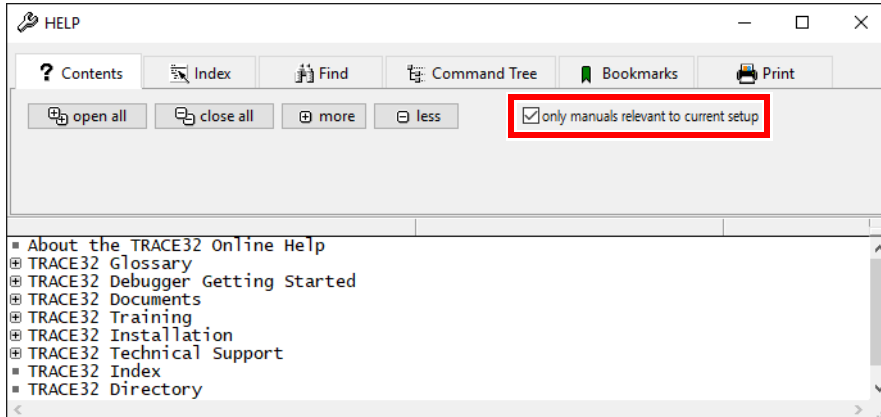
HELP Register ;Displays information about the Register command
```

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](#)

The online help presents only the information/manuals relevant to the current setup by default.



TRACE32 uses help filters for this purpose. Most filters are set automatically, depending on the connected TRACE32 hardware, the selected CPU and other settings. Some help filters have to be set by the command **HELP.FILTER.ADD**. These include manuals for integration with third-party tools, OS Awareness manuals and other TRACE32 debug extensions.

The command **HELP.FILTER.List** provides a list of all active help filters.

You can activate/deactivate the help filters, add or delete help filters using the commands listed in the **See also** block below. The help filters are listed in “**Appendix A - Help Filters**”, page 373.

See also

- [HELP.FILTER.Add](#) ■ [HELP.FILTER.Delete](#) ■ [HELP.FILTER.List](#) ■ [HELP.FILTER.RESet](#)
- [HELP.FILTER.set](#) ■ [HELP](#)
- ▲ 'Appendix A - Help Filters' in 'PowerView Command Reference'
- ▲ 'HELP System' in 'PowerView User's Guide'

Format: **HELP.FILTER.Add** *<help_filter>*

Adds filter *<help_filter>* to the help filter list. Adding the filter causes the accompanying manual to appear in the online help.

Example:

```
; show Linux debugging manuals in the online help
HELP.FILTER.Add rtoslinux

; show manuals that provide details about the integration for Simulink
; in the online help
HELP.FILTER.Add intsimulink
```

See also

■ [HELP.FILTER](#)

■ [HELP.FILTER.set](#)

Format: **HELP.FILTER.Delete** *<help_filter>*

Deletes the help filter *<help_filter>* from the help filter list. Removing the filter results in the accompanying manual no longer being displayed in the online help.

Example:

```
;Active help filters:   bdmarm;rtoslinux

HELP.FILTER.Delete    rtoslinux

;Remaining help filter: bdmarm;
```

See also

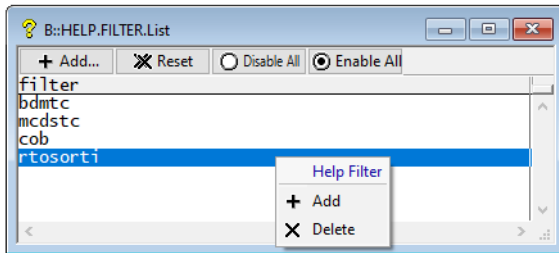
■ [HELP.FILTER](#)

■ [HELP.FILTER.set](#)

▲ ['Release Information' in 'Legacy Release History'](#)

Format: **HELP.FILTER.List**

List all active help filters.



See also

■ [HELP.FILTER](#)

■ [HELP.FILTER.set](#)

HELP.FILTER.RESet

Reset help filter system

Format: **HELP.FILTER.RESet**

Deletes all help filters and unchecks the check box **only manuals relevant to current setup**. All TRACE32 manuals are displayed.

See also

■ [HELP.FILTER](#)

■ [HELP.FILTER.set](#)

Format: **HELP.FILTER.set** [ON | OFF]

HELP.FILTER.set without argument toggles the help filter

ON
(default)

Activates the help filters, i.e. the check box **only manuals relevant to current setup** is checked. The online help presents only the information/manuals relevant to the current setup.

OFF

Deactivates the help filters, i.e. the check box **only manuals relevant to current setup** is unchecked. All TRACE32 manuals are displayed.

Example:

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

See also

■ [HELP.FILTER](#)

■ [HELP.FILTER.Add](#)

■ [HELP.FILTER.Delete](#)

■ [HELP.FILTER.List](#)

■ [HELP.FILTER.RESet](#)

Format: **HELP.Find** <string> [/<options>]

<options>: **Case** | **Similar**

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 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

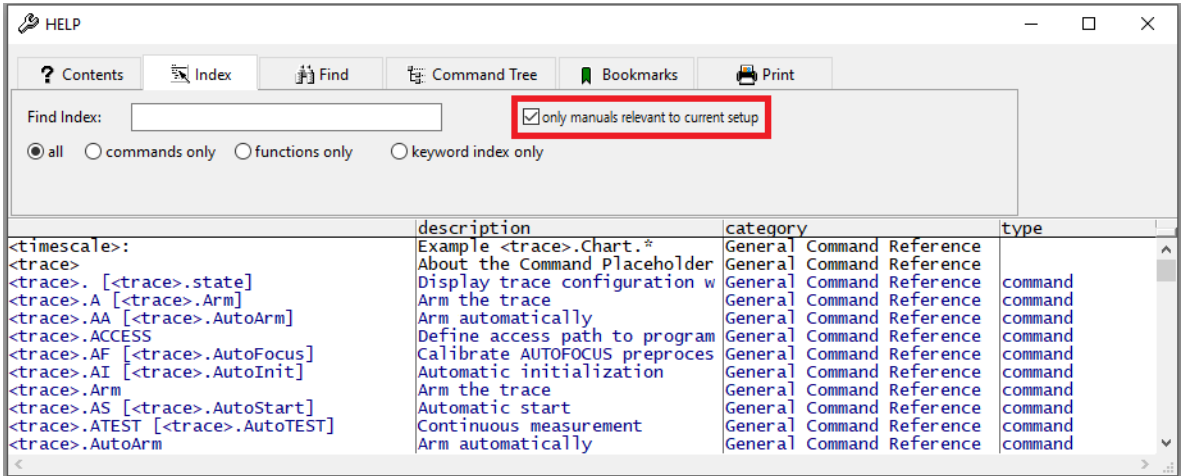
■ [HELP](#)

▲ ['Release Information' in 'Legacy Release History'](#)

Format: **HELP.Index** [*<string>*] [**All** | **Command** | **Function** | **Short**]

Opens the **HELP.Index** window, displaying a complete alphabetic list of the TRACE32 commands, functions, and other indexed terms.

Please be aware, that online help presents only the information/manuals relevant to the current setup by default. If you want to perform your search through all document uncheck first the check box **only manuals relevant to current setup** first.



The help index contains the full and short forms of the commands and functions. For more information about short and full forms, see “[Long Form and Short Form of Commands and Functions](#)” in PowerView User’s Guide, page 29 (ide_user.pdf).

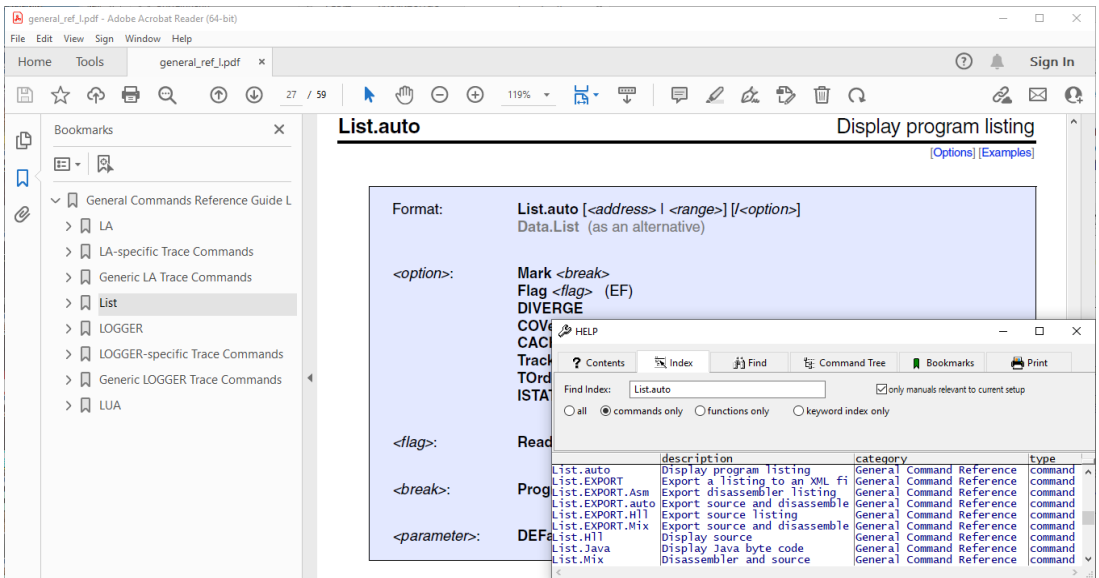
- All** Show the complete index list.
- Command** Show only commands in the index list.
- Function** Show only functions in the index list.
- Short** Show only a keyword index file (no commands, no functions). The Short option refers to the **keyword index only** radio button.

The colors of the index entry show the type:

- blue text** 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 'PowerView User's Guide'](#)

Format: **HELP.OPEN** *<string>* [*/<options>*]

<options>: **Function**

Opens the PDF documentation for the specified command or function. The **HELP** window is not opened.

See also

■ [HELP](#)

Format: **HELP.PDF** <file>

Opens a PDF file in a PDF viewer.

Example:

```
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 'PowerView 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 'PowerView User's Guide'](#)

See also

- [HELP.PRinT.PRinTsel](#)
- [HELP.PRinT.SElect](#)
- [HELP.PRinT.show](#)
- [HELP.PRinT.UNSElect](#)
- [HELP](#)

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](#)

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](#)

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](#)

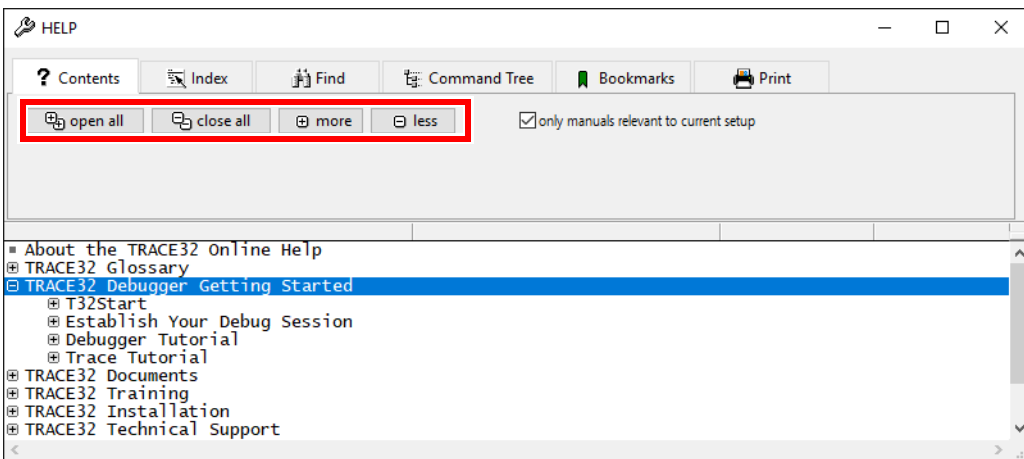
Format: **HELP.Topics** [/Close | /Open]

Shows the structure of the online help system..

Close Close all open tree branches.

Open Open all tree branches.

The online help structure lists all help pdfs. Please be aware, that online help presents only the information/manuals relevant to the current setup by default. If you want a list of all pdfs uncheck first the check box **only manuals relevant to current setup** first.



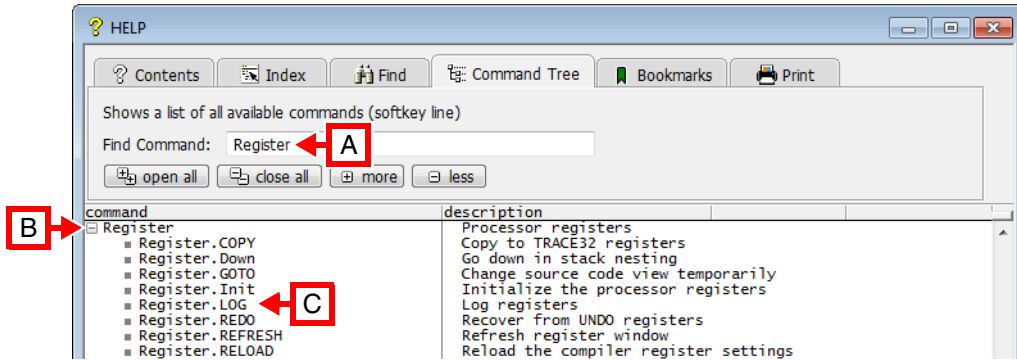
See also

■ [HELP](#)

▲ ['HELP System' in 'PowerView User's Guide'](#)

Format: **HELP.TREE <command> [/OPEN | /CLOSE]**

Lists all available commands for your hardware configuration in alphabetical order.



- A** To search for a specific command, type some letters in the **Find Command** input field.
- B** The list automatically scrolls to the first command matching your input.
- C** Double-clicking the desired command opens the appropriate help file in a PDF viewer.

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.

Close	Close all open tree branches.
Open	Open all tree branches.

Examples:

```
HELP.TREE "Register"           ; opens the HELP.TREE window, showing the
                               ; Register command and its subcommands
HELP.TREE /Open                ; open all help tree branches
HELP.TREE /Close               ; close all help tree branches
```

See also

- [HELP](#)
- ▲ ['PowerView - Screen Display' in 'PowerView User's Guide'](#)

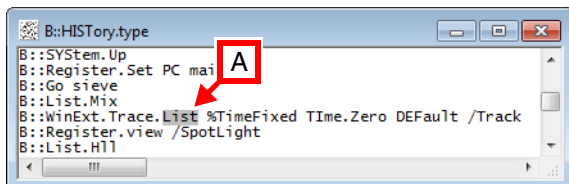
The last executed commands **you have typed at the TRACE32 command line** are stored in a history buffer, which can be displayed in the **HISTory.type** window. The history buffer also records command inputs that contain syntax errors and are thus not executed.

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

To recall the last executed commands from the history buffer, you have the following options:

- Press the **up-arrow** key in the command line to recall the most recent command in the command line. Press the **up-arrow** key repeatedly to recall older history entries.
- Press the **down-arrow** key to return to the most recent command in the history buffer.
- Type any string in the TRACE32 command line, and then press the **up-arrow** key or **down-arrow** key to perform a search in the history for lines matching the keyword.
- Use the **HISTory.type** window.

Example: Let's assume you have typed `list` at the command line and are pressing the **up-arrow** key. Only history entries containing the search item `list` will be suggested in the command line. In the **HISTory.type** window, the current match for `list` will additionally be highlighted [**A**].



See also

-
- [HISTory.eXecute](#)
 - [HISTory.SAVE](#)
 - [HISTory.Set](#)
 - [HISTory.SIZE](#)
 - [HISTory.type](#)
 - [LOG](#)
- ▲ 'Create a PRACTICE Script' in 'Training Script Language PRACTICE'

Format: **HISTory.eXecute**

Executes all commands in the history list.

See also

- [HISTory](#)
- [HISTory.SAVE](#)
- [HISTory.type](#)
- ▲ ['Commands' in 'PowerView User's Guide'](#)

HISTory.SAVE

Store command history log

Format: **HISTory.SAVE** [*file*]

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

Example:

```
;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](#)
- [HISTory.type](#)
- ▲ ['Commands' in 'PowerView User's Guide'](#)
- ▲ ['Create a PRACTICE Script' in 'Training Script Language PRACTICE'](#)

Format:	HISTory.Set <i><item></i> <i><string></i> [<i><string></i> <i><string></i>]
<i><item></i> :	CMD FILE HLL ADDRESS RANGE TraceFIND WelcomeScripts PDEBUG

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

CMD <i><string></i>	Adds an entry to the TRACE32 command history (which can be viewed with the command HISTory.type)
FILE <i><str1></i> <i><str2></i> <i><str3></i>	Adds an entry to the list of recently used files in the FILE menu. <ul style="list-style-type: none"> <i><str1></i> is the name of the icon shown left of the entry. <i><str2></i> is the name of the command to be executed when clicking on the entry. <i><str3></i> is the working directory in which the command is executed.
HLL <i><string></i>	Adds an entry to the list of recently used HLL expressions e.g. in the Var.Break.Set window.
ADDRESS <i><string></i>	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 <i><address></i> in connection with a command.
RANGE <i><string></i>	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 <i><range></i> in connection with a command.
TraceFIND <i><string></i>	Adds an entry to the list of recently searched items in the expert search of a trace recording.
WelcomeScripts <i><string></i>	Adds an entry to the list of recently searched items in the Search for scripts window (see WELCOME.SCRIPTS).
PDEBUG	Allows automated storage of PRACTICE debug script parameters in history list (see SETUP.PDEBUG.ScriptParams).

See also

■ [HISTory.SAVE](#)

■ [HISTory](#)

■ [HISTory.type](#)

▲ ['Commands' in 'PowerView 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](#)[■ HISTory.type](#)[▲ 'Commands' in 'PowerView 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).

Examples:

```
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](#)

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](#)

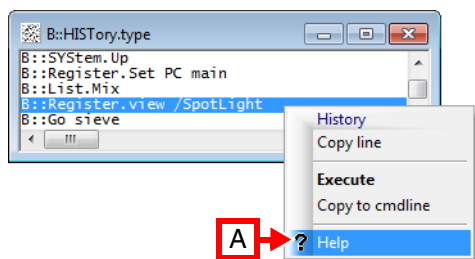
HISTory.type

Display command history log of last executed commands

Format: **HISTory.type**

Displays the command history buffer 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.



A Displays the online help for the selected command.

See also

■ [HISTory](#)

■ [HISTory.eExecute](#)

■ [HISTory.SAVE](#)

■ [HISTory.Set](#)

■ [HISTory.SIZE](#)

▲ 'Commands' in 'PowerView 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

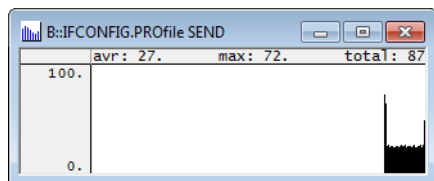
- [IFCONFIG.PROfile](#)
- [IFCONFIG.state](#)
- [IFCONFIG.TEST](#)
- [IFCONFIG.DEVICENAME\(\)](#)
- [IFCONFIG.ETHernetADDRESS\(\)](#)
- [IFCONFIG.IPADDRESS\(\)](#)
- [IFTEST.LATENCY\(\)](#)
- ▲ 'IFCONFIG and IFTEST Functions' in 'PowerView Function Reference'
- ▲ 'Starting a TRACE32 PowerView Instance' in 'Training Basic Debugging'
- ▲ 'Starting a TRACE32 PowerView Instance' in 'Training Basic SMP Debugging'
- ▲ 'Starting a TRACE32 PowerView Instance' in 'Training Basic SMP Debugging for Intel® x86/x64'

```
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 | SIMINST
```

Display a time profile about the usage of resources.

Window with time profile about the usage of resources.



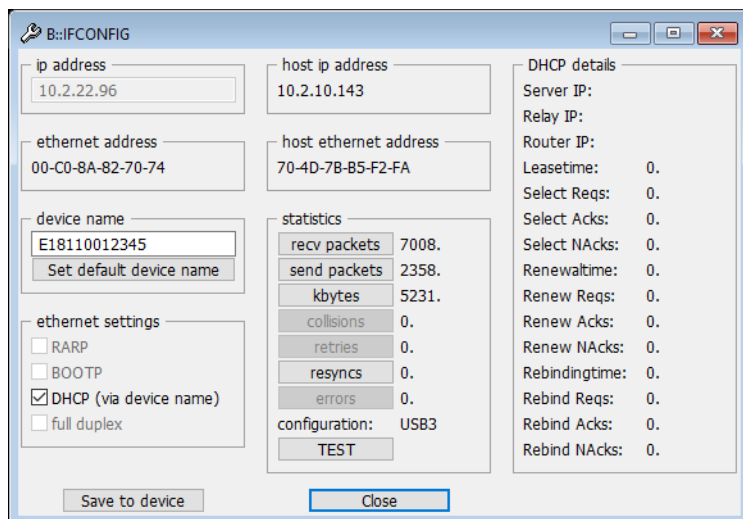
COL	Collisions when sending packets on Ethernet.
DPACKETS	Debug access packets.
ERROR	Communication errors.
FILECACHEHITS	File cache hits. (CHITS)
FILECACHEMISSES	File cache misses. (CMISSES)
KBYTE	Communication utilization in KBytes, all packets received and sent.
MAINTHREAD	Main thread utilization.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

See also

-
- [IFCONFIG](#) ■ [IFCONFIG.state](#) ■ [IFCONFIG.TEST](#)
 - ▲ ['Interface' in 'PowerView User's Guide'](#)

Format: **IFCONFIG.state**

Opens the **IFCONFIG.state** dialog used for configuring Ethernet / USB connections.



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.

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** block 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.

Starting with the PowerDebug X50, the network mask needs to be specified in CIDR notation to allow routed access to the PowerDebug device. For example, to specify the IP address 198.51.100.42 with a subnet mask of 255.255.255.192, enter 198.51.100.42/26 into this field.

gateway

Gateway, also known as default route.

ethernet address

Displays the ethernet address of the debug interface (read-only)

device name	<p>The device name is used to address a certain debug module.</p> <p>If the debug module is connected via USB, the device name can be used to address a specific debug module if multiple debug modules are connected to the PC. A firmware update may be required to enable this feature.</p> <p>If the debug module is connected via Ethernet, the device name is used to retrieve the IP address via DHCP.</p>
Set default device name	Click to set the device name back to the default device name. The default device name is the serial number of the debug module.
RARP	Use the Reverse Address Resolution Protocol (RARP) to retrieve the IP address. This option is greyed out if not supported by the sued PowerDebug module.
BOOTP	Use the Bootstrap Protocol (BOOTP) to retrieve the IP address. This option is greyed out if not supported by the sued PowerDebug module.
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.TEST](#)
 - [HOSTIP\(\)](#)
 - [IFCONFIG.ETHernetADDRESS\(\)](#)
 - [IFTEST.LATENCY\(\)](#)
 - ▲ ['IFCONFIG and IFTEST Functions' in 'PowerView Function Reference'](#)
 - ▲ ['Interface' in 'PowerView User's Guide'](#)
 - [IFCONFIG.PROfile](#)
 - [HOSTID\(\)](#)
 - [IFCONFIG.DEVICENAME\(\)](#)
 - [IFCONFIG.IPADDRESS\(\)](#)

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.

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 [<i><warp></i>]	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

Example:

```
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
```



See also

- [IFCONFIG](#)
- [IFCONFIG.PROfile](#)
- [IFCONFIG.state](#)
- [IFTEST.DOWNLOAD\(\)](#)
- [IFTEST.LATENCY\(\)](#)
- [IFTEST.UPLOAD\(\)](#)
- ▲ ['Interface' in 'PowerView User's Guide'](#)

InterCom Data exchange between different TRACE32 PowerView instances

The **InterCom** system allows the exchange of data between different TRACE32 systems. The exchange is based on UDP. The destination system is defined by a port number of a UDP port used by this TRACE32 system. This requires an entry in the 'config.t32' file of any participating TRACE32 system:

```
IC=NETASSIST
PORT=20001
NAME=firstInstance

...
```

NOTE: If multiple TRACE32 systems are used on one host, the port numbers must differ!

A good way to familiarize yourself with the **InterCom** command group is to start with the example given in [InterCom.ENABLE](#).

See also

- | | |
|--|--|
| ■ InterCom.ENABLE | ■ InterCom.Evaluate |
| ■ InterCom.execute | ■ InterCom.executeNoWait |
| ■ InterCom.NAME | ■ InterCom.PING |
| ■ InterCom.PipeCLOSE | ■ InterCom.PipeOPEN |
| ■ InterCom.PipeREAD | ■ InterCom.PipeWRITE |
| ■ InterCom.PORT | ■ InterCom.WAIT |
| ■ SETUP.InterComACKTIMEOUT | ■ SYnch |
| ■ TargetSystem | □ InterCom.PING() |
| □ InterCom.PODPORT() | □ InterCom.PODPORTNUMBER() |
| □ InterCom.PORT() | |
- ▲ ['InterCom Functions' in 'PowerView Function Reference'](#)
▲ ['InterCom' in 'PowerView User's Guide'](#)

Format: **InterCom.ENABLE** *<intercom_name>* [*!<option>*]

<option>: **INSTance** *<instance>* | **UseCore** *<core>*

Assigns a user-defined InterCom name to the current TRACE32 PowerView instance, and TRACE32 *automatically* chooses and assigns the next free InterCom UDP port number.

If the InterCom name was already set in the config file, this command overrides the initial InterCom name from the config file.

To view or return the current InterCom name and UDP port number, open the [TargetSystem.state](#) window or use the functions [InterCom.NAME\(\)](#) and [InterCom.PORT\(\)](#).

NOTE: To assign a user-defined InterCom name and a user-defined UDP port number, use the commands [InterCom.NAME](#) and [InterCom.PORT](#).

<p>INSTance <i><instance></i></p>	<p>Changes the InterCom name of a remote TRACE32 instance specified by <i><instance></i>.</p> <p>Alternatively, double-click the desired InterCom name in the ic name column of the TargetSystem.state window. For an illustrated example, see InterCom.NAME.</p>
<p>UseCore <i><core></i></p>	<p>Changes the InterCom name of that instance where the UseCore <i><core></i> index matches the <code>CORE=<core></code> index in the config file.</p>

Example: The TRACE32 PowerView instance named `firstInst` starts another instance named `secondInst` for the purpose of debugging two cores of an AMP system.

```
;shut down previous debug session
InterCom.execute ALL WinCLEAR
InterCom.execute ALL SYStem.Down

;assign the user-defined InterCom name 'firstInst' to the instance
;executing this PRACTICE script
InterCom.ENABLE firstInst

;select the 1st CortexA9MPCore core of OMAP4430 for this instance
SYStem.CPU OMAP4430
CORE.ASSIGN 1.
SYStem.CONFIG.CORE 1. 1.

;open a 2nd TRACE32 PowerView instance and assign the user-defined
;InterCom name 'secondInst'
TargetSystem.NewInstance secondInst /ONCE

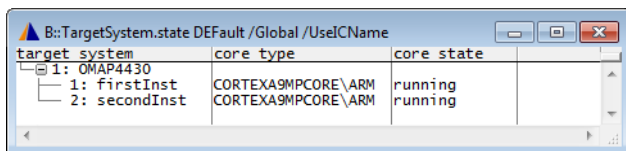
;select the 2nd CortexA9MPCore core of OMAP4430 for the 2nd instance
InterCom.execute secondInst SYStem.CPU OMAP4430
InterCom.execute secondInst CORE.ASSIGN 2.
InterCom.execute secondInst SYStem.CONFIG.CORE 2. 1.

;display a status overview of the AMP system
TargetSystem.state DEFault /Global /UseICName

;connect to the AMP system
SYStem.Up
InterCom.execute OTHERS SYStem.Up

;<your_code> ... e.g. load your application program with
;InterCom.execute <instance_name> Data.LOAD...

InterCom.execute ALL Go
```



target_system	core type	core state
1: firstInst	CORTEXA9MPCORE\ARM	running
2: secondInst	CORTEXA9MPCORE\ARM	running

See also

- [InterCom](#)
- [InterCom.execute](#)
- [SUBTITLE](#)
- [TITLE](#)
- [SYnch.Connect](#)
- [TargetSystem.NewInstance](#)
- ▲ ['Release Information' in 'Legacy Release History'](#)

Format: **InterCom.Evaluate** *<instance>* [*<function>*]

<instances>: *<intercom_name>* | **SELF** | **ALL** | **OTHERS** | [*<host>*:]*<port>*

Retrieves the result of a function executed on the remote system. Once retrieved, the result can be accessed by using the (local) **EVAL()** function. If no function is specified, the result of the remote **EVAL()** function will be retrieved.

<instance>

For parameter descriptions, see [InterCom.execute](#).
Group names, such as `cluster1.*`, are *not* allowed!

Example: This script reads the value of the register DEC of the TRACE32 PowerView instance named `secondInstance`

```
InterCom.Evaluate secondInstance Register(DEC)

&remote_register_value=EVAL()

PRINT "DEC=" EVAL()
```

See also

- [InterCom](#)
- [InterCom.execute](#)
- [EVAL\(\)](#)
- ▲ ['InterCom' in 'PowerView User's Guide'](#)
- ▲ ['Release Information' in 'Legacy Release History'](#)

Format: **InterCom.execute** <instances> <command>

<instances>: <intercom_name> | **SELF** | **ALL** | **OTHERS** | <name_pattern> | [**<host>:**]<port>

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.

<host>:<port>	<p>Name of the host and the port number.</p> <p>InterCom.execute localhost:<port_number> <command> can execute a command on any TRACE32 instance, even on a TRACE32 instance that is connected to another debugger hardware.</p> <p>Examples:</p> <pre>InterCom.execute 10000 PRINT "Hello world!" InterCom.execute 127.0.0.1:10000 PRINT "Hello world!" InterCom.execute localhost:10000 PRINT "Hello world" InterCom.execute stel:10000 PRINT "Hello world!"</pre>
----------------------------------	--

The following arguments work only in AMP debug scenarios:

<intercom_name>	InterCom name of a TRACE32 instance. Names can be assigned to TRACE32 instances with the InterCom.NAME command.
<name_pattern>	The InterCom.execute command supports the use of the wildcards * and ? in InterCom names. See example 3 .
ALL	All known TRACE32 instances.
OTHERS	ALL except SELF .
SELF	This TRACE32 instance.

NOTE: When executing a PRACTICE script (*.cmm) on the remote TRACE32 PowerView instance using **InterCom.execute <...> DO <file>** the local TRACE32 PowerView instance 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 1: This script shuts down the previous AMP debug session.

```
InterCom.execute ALL WinCLEAR      ;close all windows of the previous AMP
                                   ;debug session
InterCom.execute ALL SYStem.Down
```

Example 2: In this script, two commands are executed on the remote TRACE32 PowerView instance.

```
InterCom.execute localhost:20002 Register.RESet
InterCom.execute localhost:20002 Go.direct
```

Example 3: This script executes the **SYStem.Attach** command on all TRACE32 PowerView instances whose InterCom names start with `cluster1`.

```
InterCom.execute cluster1.* SYStem.Attach
```

See also

-
- [InterCom](#)
 - [InterCom.ENABLE](#)
 - [InterCom.Evaluate](#)
 - [InterCom.executeNoWait](#)
 - [InterCom.NAME](#)
 - [InterCom.PING](#)
 - [InterCom.PipeCLOSE](#)
 - [InterCom.PipeOPEN](#)
 - [InterCom.PipeREAD](#)
 - [InterCom.PipeWRITE](#)
 - [InterCom.PORT](#)
 - [InterCom.WAIT](#)
- ▲ ['InterCom' in 'PowerView User's Guide'](#)
 - ▲ ['Release Information' in 'Legacy Release History'](#)

Format: **InterCom.executeNoWait** *<instances>* *<command>*

<instances>: *<intercom_name>* | **SELF** | **ALL** | **OTHERS** | *<name_pattern>* | [*<host>*:]*<port>*

Executes TRACE32 commands on the remote TRACE32 PowerView instance. The commands will be executed immediately, and the local system will *not wait* until the remote TRACE32 PowerView instance 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.

<instances>

For parameter descriptions, see [InterCom.execute](#).

Example: Execute commands on the remote TRACE32 PowerView instance named `secondInstance`

```
InterCom.executeNoWait secondInstance Data.LOAD.Elf bigfile.elf
```

See also

- [InterCom](#)
- [InterCom.execute](#)
- ▲ 'Release Information' in 'Legacy Release History'

InterCom.NAME

Assign user-defined InterCom name

[\[Example\]](#)

Format: **InterCom.NAME** *<intercom_name>* [*<option>*]

<option>: **INSTance** *<instance>* | **UseCore** *<core>*

Assigns a user-defined InterCom name to the current TRACE32 PowerView instance. If the InterCom name was already set in the config file, this command overrides the initial InterCom name from the config file.

To view or return the current InterCom name, open the [TargetSystem.state](#) window or use the [InterCom.NAME\(\)](#) function.

NOTE: **InterCom.NAME** *<intercom_name>* does not change the current InterCom UDP port number, in contrast to [InterCom.ENABLE](#) *<intercom_name>*.

<option>

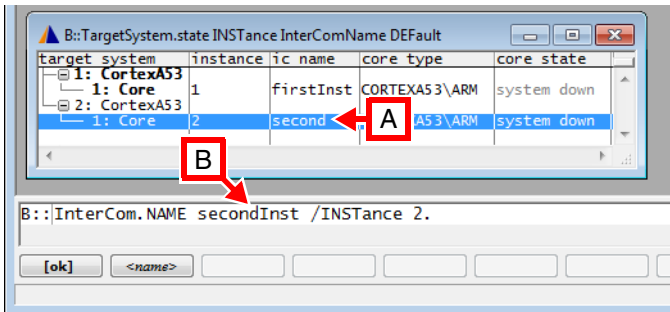
For a description of the options, see [InterCom.ENABLE](#).

Example: The following example is for demo purposes only. It shows how to assign a new InterCom name to a TRACE32 instance. See also screenshot below.

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

;returns: firstInst
PRINT InterCom.NAME()

;displays the name in the 'ic name' column
TargetSystem.state INSTANCE InterComName DEFault
```



- A** You can rename an instance by double-clicking a name in the **ic name** column.
- B** Double-clicking inserts the command **InterCom.NAME** into the TRACE32 command line. Simply enter a new name before the **/Instance** option, e.g. `secondInst`.

See also

- [InterCom](#)
- [InterCom.execute](#)
- [SUBTITLE](#)
- [TITLE](#)
- [SYnch.Connect](#)
- [InterCom.NAME\(\)](#)
- ▲ 'Release Information' in 'Legacy Release History'

Format: **InterCom.PING** <intercom_name> | [<host>:]<port> [/<option>]

<option>: **Large**

Sends one test message through the InterCom system to another TRACE32 PowerView instance. If everything works, the other instance will display the message 'PING received' and the sending TRACE32 PowerView instance will display the message 'PING response received'.

<host>:<port>	For description and examples, see InterCom.execute .
<intercom_name>	InterCom name of a TRACE32 instance. Names can be assigned to TRACE32 instances with the InterCom.NAME command.
Large	Sends a large data packet to test the throughput.

Example: This script checks the connection of the TRACE32 system with the InterCom UDP port 20002 on host node 'ste':

```
InterCom.PING ste:20002
```

See also

- [InterCom](#)
- [InterCom.execute](#)
- [InterCom.PING\(\)](#)
- ▲ ['InterCom' in 'PowerView User's Guide'](#)

InterCom.PipeCLOSE

Close named pipe

Format: **InterCom.PipeCLOSE** #<file_number>

Closes the named pipe.

See also

- [InterCom](#)
- [InterCom.execute](#)
- ▲ ['InterCom' in 'PowerView User's Guide'](#)

Format: **InterCom.PipeOPEN #<file_number> <file> [/<option>]**

<option>: **Read | Write | Create**

Opens or creates a named pipe. Named pipes allow to exchange data between different applications. The usage depends on the host OS.

Example:

```
;Opens a pipe for listening on Windows  
;NOTE: The directory name is fixed for Windows hosts!  
InterCom.PipeOPEN #1 \\.\pipe\mypipe /Read /Create
```

See also

- [InterCom](#)
- [InterCom.execute](#)
- ▲ ['InterCom' in 'PowerView User's Guide'](#)

Format: **InterCom.PipeREAD #<file_number> <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' in 'PowerView User's Guide'](#)

Format: **InterCom.PipeWRITE** #<file_number> <arglist>

Writes data to a named pipe. Similar to the PRACTICE **WRITE** command.

See also

- [InterCom](#)
- [InterCom.execute](#)
- ▲ ['InterCom' in 'PowerView User's Guide'](#)

InterCom.PORT

Assign user-defined InterCom UDP port number

Format: **InterCom.PORT** <port_number> | **0**. [/<option>]

<option>: **INSTance** <instance> | **UseCore** <core>

Assigns a user-defined InterCom UDP port number to the current TRACE32 PowerView instance. If the InterCom UDP port number was already set in the config file, this command overrides the initial port number from the config file.

To view or return the current InterCom UDP port number, open the **TargetSystem.state** window or use the **InterCom.PORT()** function.

0.	Removes the InterCom UDP port for the currently selected TRACE32 instance.
<option>	For a description of the options, see InterCom.ENABLE .
<port_number>	Parameter Type: Decimal value .

Example: This script is for demo purposes only. It shows how to assign a new InterCom UDP port number to a TRACE32 instance. See also screenshot below.

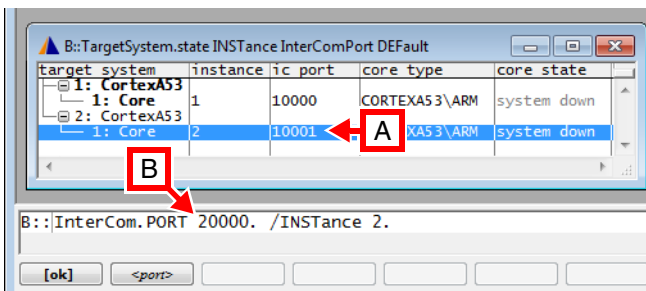
```

;assigns port number 10000. to the current TRACE32 PowerView instance
InterCom.PORT 10000.

;returns: 10000.
PRINT InterCom.PORT()

;displays the port number in the 'ic port' column
TargetSystem.state DEFault InterComPort

```



- A** You can assign a new port number by double-clicking a port number in the **ic port** column.
- B** Double-clicking inserts the command **InterCom.PORT** into the TRACE32 command line. Simply enter a new port number before the **/Instance** option, e.g. 20000.

See also

- [InterCom](#)
- [InterCom.execute](#)
- [InterCom.PORT\(\)](#)

Format: **InterCom.WAIT** <instances> [<condition> | <time>]

<instances>: <intercom_name> | **SELF** | **ALL** | **OTHERS** | <name_pattern> |
[<host>:]<port>

The command **InterCom.WAIT** has two main applications:

- Wait until the remote system is responsive and available.
- Wait until the remote system finished executing a running script i.e. until the PRACTICE interpreter becomes “idle”.

NOTE: **InterCom.WAIT** does *not* work from the TRACE32 command line.

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.

<instance>	For parameter descriptions, see InterCom.execute .
<condition>	PRACTICE functions that return the boolean values TRUE or FALSE as well as PRACTICE functions returning 0 and !=0. For more information about the permissible return values, see: <ul style="list-style-type: none"> • TRUE() and FALSE() • Return Value Type: Decimal value.
<time>	Parameter Type: Time value .

Example 1: Start a second TRACE32 system and wait until it can be controlled via [InterCom](#):

```
DO start_trace32_b.cmm ; start debugger that listens on port 10001
InterCom.WAIT localhost:10001
```

Example 2:

```
InterCom.WAIT ALL !RUN() ;wait till all instances have stopped
InterCom.executeNoWait ALL Data.LOAD.Elf big.elf /NoCODE
InterCom.WAIT ALL
```

See also

- [InterCom](#)
- [InterCom.execute](#)
- ▲ ['Release Information' in 'Legacy Release History'](#)

Using the **LICENSE** command group, you can list the serial numbers and maintenance contracts of your debugging product and update your maintenance license.

See also

- [LICENSE.List](#)
 - [VERSION](#)
 - [LICENSE.REQest](#)
 - [LICENSE.DATE\(\)](#)
 - [LICENSE.state](#)
 - [LICENSE.GRANTED\(\)](#)
 - [LICENSE.UPDATE](#)
 - [LICENSE.MULTICORE\(\)](#)
- ▲ ['LICENSE Functions' in 'PowerView Function Reference'](#)
▲ ['Release Information' in 'Legacy Release History'](#)
▲ ['Do you have a valid Software License Key?' in 'Software Updates'](#)

LICENSE.List

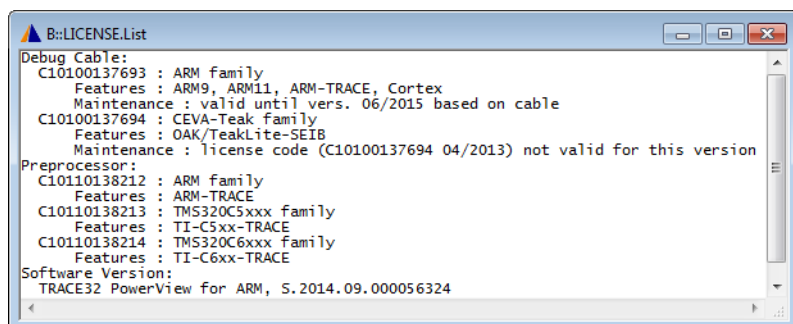
Display all license information

Format: **LICENSE.List**

Opens a window which shows all **serial numbers** and corresponding **maintenance contracts** of your debugging product.

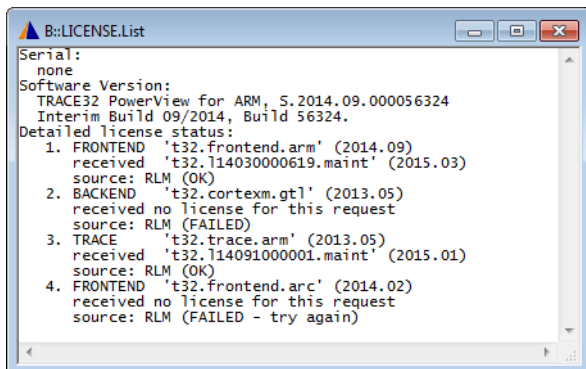
If you are using a In-Circuit Debugger, the window shows also the **feature keys** stored in your Debug Cable, Nexus Adapter or Preprocessor.

Example 1: LICENSE.List window for a setup with a debug cable and preprocessor:



```
B::LICENSE.List
Debug Cable:
C10100137693 : ARM family
Features : ARM9, ARM11, ARM-TRACE, Cortex
Maintenance : valid until vers. 06/2015 based on cable
C10100137694 : CEVA-Teak family
Features : OAK/TeakLite-SEIB
Maintenance : license code (C10100137694 04/2013) not valid for this version
Preprocessor:
C10110138212 : ARM family
Features : ARM-TRACE
C10110138213 : TMS320C5xxx family
Features : TI-C5xx-TRACE
C10110138214 : TMS320C6xxx family
Features : TI-C6xx-TRACE
Software Version:
TRACE32 PowerView for ARM, S.2014.09.000056324
```

Example 2: LICENSE.List window for a software-only setup using [LICENSE.REQuest](#):



```
B::LICENSE.List
Serial:
none
Software Version:
TRACE32 PowerView for ARM, S.2014.09.000056324
Interim Build 09/2014, Build 56324.
Detailed license status:
1. FRONTEND 't32.frontend.arm' (2014.09)
   received 't32.t14030000619.maint' (2015.03)
   source: RLM (OK)
2. BACKEND 't32.cortexm.gt1' (2013.05)
   received no license for this request
   source: RLM (FAILED)
3. TRACE 't32.trace.arm' (2013.05)
   received 't32.t14091000001.maint' (2015.01)
   source: RLM (OK)
4. FRONTEND 't32.frontend.arc' (2014.02)
   received no license for this request
   source: RLM (FAILED - try again)
```

See also

■ [LICENSE](#)

▲ 'Release Information' in 'Legacy Release History'

LICENSE.REQuest

Request a license

Format 1: **LICENSE.REQuest.plain** *<product>* [*<version>*]

Format 2: **LICENSE.REQuest.<sub_cmd>**

<sub_cmd>: **FRONTEND | INTEGRATION | MULTICORE | SIMULATOR |**
 BACKEND | TRACE | plain

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\(\)](#).

BACKEND	Requests a license for the currently selected backend.
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.
TRACE	Requests a trace license for the current architecture and version.

plain	License request for a particular <i><product></i> and <i><version></i> .
<i><product></i>	License product name as a string, e.g. as given in a lauterbach-*.lic file. For example: "t32.trace.x86"
<i><version></i>	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.

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.

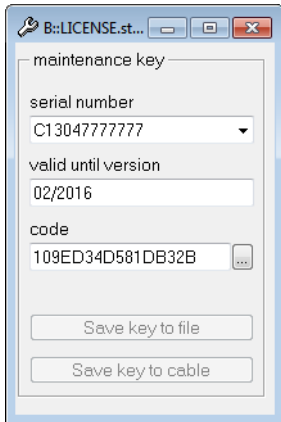
See also

- [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.



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](#)

Format: **LICENSE.UPDATE** [*<license_file>* | *<maintenance>*] [**FILE**]

Updates the **maintenance contract(s)** inside your plugged Debug Cable or Nexus Adapter.

If the license is not intended for a Debug Cable or Nexus Adapter or if the option **FILE** is used, the license in your license file (usually license.t32) is updated.

Examples:

```
; example for <license_file>
; updates all maintenance contracts in currently used Debug Cable/
; Nexus Adapter from data in given file
LICENSE.UPDATE license095970.t32
```

```
; 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"
```

```
; stores currently used maintenance contract to Debug Cable/Nexus Adapter
LICENSE.UPDATE
```

See also

■ [LICENSE](#)

▲ ['Release Information' in 'Legacy Release History'](#)

LOG Log TRACE32 commands and PRACTICE script calls

Using the **LOG** command group, you can trace all executed TRACE32 commands and the call hierarchy of PRACTICE scripts (*.cmm). Operations activated by the mouse will be changed to line-oriented commands. Commands and PRACTICE script calls are stored in log files which have either a default or a user-defined log file name. Commands can additionally be logged by printing them to an **AREA** window and recording them in a command log file at the same time.

Regardless of which output you choose, the trace information is recorded line by line in the command line format.

Command Log File

Every new **LOG.OPEN** command generates a new command log file, overwriting the old one. The size of the command 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.

Log File for PRACTICE Script Calls

The call hierarchy of PRACTICE scripts can be logged automatically or manually. In either case, the log mechanism is based on the **LOG.DO** command.

- The automatic log mechanism is useful for logging the call hierarchy of scripts that are executed automatically on start-up of TRACE32. For more information, refer to [“Logging the Call Hierarchy of PRACTICE Scripts”](#) in PRACTICE Script Language User’s Guide, page 17 (practice_user.pdf).
- After the start-up of TRACE32, you can manually log the calls of PRACTICE script files using the **LOG.DO** command.

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.DO | ■ LOG.OFF | ■ LOG.ON |
| ■ LOG.OPEN | ■ LOG.toAREA | ■ LOG.type | ■ HISTory |
| ■ SYStem.LOG | | | |

▲ ‘Logging Commands’ in ‘PowerView User’s Guide’

▲ ‘Logging the Call Hierarchy of PRACTICE Scripts’ in ‘PRACTICE Script Language User’s Guide’

Format: **LOG.CLOSE**

The activated command log file is closed.

Example:

```
LOG.OPEN                               ; opens file 't32.log'  
; ...  
LOG.CLOSE                             ; close file and terminate logging function
```

See also

■ [LOG](#)

■ [LOG.OPEN](#)

■ [LOG.type](#)

▲ ['Logging Commands' in 'PowerView User's Guide'](#)

▲ ['Create a PRACTICE Script' in 'Training Script Language PRACTICE'](#)

LOG.DO

Log calls of PRACTICE scripts

Format: **LOG.DO** [*<file>*]

Logs the calls of PRACTICE scripts (*.cmm) to a file. Whenever a PRACTICE script is called by **DO**, **RUN**, **PSTEP** or **AutoSTOre** or via an event in a **DIALOG** or **PER** file, a line is appended to the log file.

Logging will stop when:

- The last PRACTICE script ends (**PMACRO** window does not show any active scripts).
- The command **ENDDO** is executed while no script is active.
- The command **LOG.DO** is executed without a file name.
- The command **END** is executed.

Example:

```
LOG.DO "~~~/myScriptNesting.log"     ; start log in temporary directory  
DO myScript.cmm                     ; start PRACTICE script  
; log ends when PRACTICE stack becomes empty
```

Possible output in myScriptNesting.log:

```
// LOG.DO, Started via command line, TRACE32 for ARM, GUI ID: myt32
DO C:\T32\tmp\myScript.cmm
    DO C:\T32\tmp\two.cmm 123    // from line 5.
        ChDir.DO C:\T32\tmp\three.cmm a b c    // from line 2.
        DO C:\T32\tmp\three.cmm d e f    // from line 3.
    ON CoMmanD MYBLUBB DO C:\T32\tmp\three.cmm blubber    // from line 6.
    ON TIME 2000.ms DO C:\T32\tmp\four.cmm    // from line 20.
RUN C:\T32\tmp\three.cmm 456
ENDDO
```

See also

- [LOG](#)
- [LOG.type](#)
- [LOG.DO.FILE\(\)](#)
- ▲ ['Logging Commands' in 'PowerView User's Guide'](#)
- ▲ ['Logging the Call Hierarchy of PRACTICE Scripts' in 'PRACTICE Script Language User's Guide'](#)
- ▲ ['Release Information' in 'Legacy Release History'](#)

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.

Example:

```
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](#)
- ▲ ['Logging Commands' in 'PowerView User's Guide'](#)
- ▲ ['Create a PRACTICE Script' in 'Training Script Language PRACTICE'](#)

Format: **LOG.ON**

All commands are logged. This command can be used after the log has been turned off with the command **LOG.OFF**.

Example:

```
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](#)

▲ ['Logging Commands' in 'PowerView User's Guide'](#)

▲ ['Create a PRACTICE Script' in 'Training Script Language PRACTICE'](#)

Format: **LOG.OPEN** [*<file>*] [*!<option>*]

<option>: **TimeStamp**

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.

TimeStamp Adds global timestamps to log.

Example:

```
LOG.OPEN           ; opens file 't32.log'  
; ...  
; ...  
LOG.CLOSE         ; close file 't32.log' and terminate log function
```

See also

■ [LOG](#)

■ [LOG.CLOSE](#)

■ [LOG.type](#)

▲ ['Logging Commands' in 'PowerView User's Guide'](#)

▲ ['Create a PRACTICE Script' in 'Training Script Language PRACTICE'](#)

Format:	LOG.toAREA ON OFF ["<prefix>"] [<i>/</i> <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	<p>With ALL:</p> <ul style="list-style-type: none"> All commands executed by a PRACTICE script (*.cmm) are displayed in the AREA window. Commands <i>you</i> enter on the TRACE32 command line are also shown. <p>Without ALL (default):</p> <ul style="list-style-type: none"> Only commands from the PRACTICE script being executed 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>	<p>Specifies the AREA window to which the log entries are written. By default, the log entries are written to the AREA window A000.</p> <p>Alternatively, specify a user-defined AREA name you have created with the AREA.Create command.</p>
COLOR <color>	Prints the command log entries in color to the AREA window.

IndentCalls	<p>The lines of sub-scripts called with DO and sub-routines called with GOSUB are indented.</p> <p>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.</p> <p>The commands are indented with a single plus symbol for every hierarchic level opened by GOSUB or DO.</p> <p>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.</p>
--------------------	---

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.

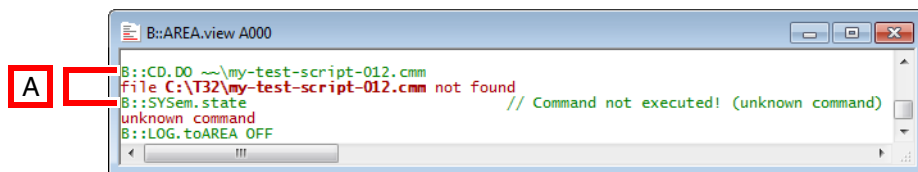
LOG.toAREA ON /ALL /COLOR.GREEN              ;log commands by writing them
                                              ;to the AREA window.

ChDir.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.

```

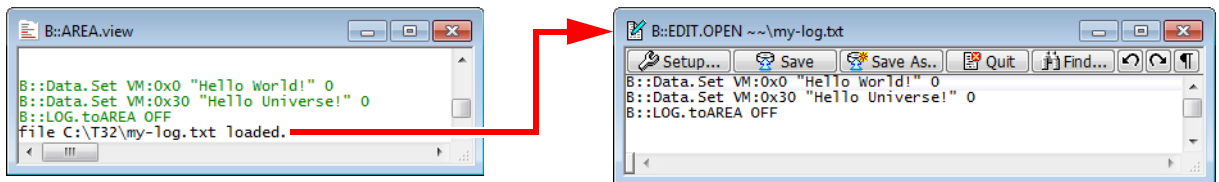


A The commands that have caused the errors. The error messages are printed directly below.

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 ;display the default AREA window.
AREA.OPEN A000 ~~\my-log.txt ;save output that will be shown
;in the AREA window to a file.
LOG.toAREA ON /ALL /COLOR.GREEN ;log commands by printing them
;to the AREA window.
;two commands for demo purposes:
Data.Set VM:0x0 "Hello World!" 0 ;set two zero-terminated strings
Data.Set VM:0x30 "Hello Universe!" 0 ;to the TRACE32 virtual memory.
LOG.toAREA OFF ;terminate the log.
AREA.CLOSE A000 ;close the output file.
EDIT.OPEN ~~\my-log.txt ;open the file in an EDIT window.
```



Example 3

A user-defined **AREA** window is created for command logging, and all lines are preceded by a user-defined prefix.

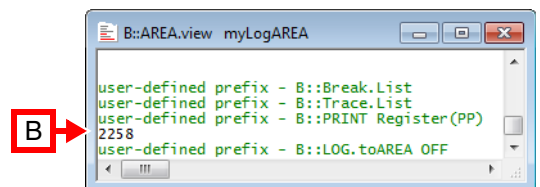
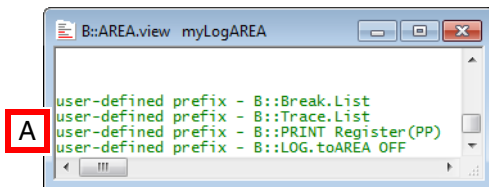
```
;create a user-defined AREA window named myLogAREA for command logging
AREA.Create myLogAREA
AREA.view myLogAREA

;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 'PowerView User's Guide'
- ▲ 'Logging Commands' in 'PowerView User's Guide'

Format: **LOG.type**

Displays the current command log file.

```

B::LOG.type
// T32_1000001 Wed Jul 30 14:43:57 2014
B::Data.LOAD C:\T32\demo\arm\compiler\ti\arm.abs
B::Register.RESet
B::Register.view
B::Register.Set PC main
// B::LOG.ON
B::List.Mix
B::Var.View flags

B::LOG.type
// T32_1000001 Wed Jul 30 14:43:57 2014
B::Data.LOAD C:\T32\demo\arm\compiler\ti\arm.abs
B::Register.RESet
B::Register.view
B::Register.Set PC main
// B::LOG.ON
B::List.Mix
B::Var.View flags

B::LOG.type
// T32_1000001 Wed Jul 30 14:43:57 2014
B::Data.LOAD C:\T32\demo\arm\compiler\ti\arm.abs
B::Register.RESet
B::Register.view
B::Register.Set PC main
// B::LOG.ON
B::List.Mix
B::Var.View flags
  
```

- 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.CLOSE
- LOG.DO
- LOG.OFF
- LOG.ON
- LOG.OPEN
- LOG.toAREA

▲ 'Logging Commands' in 'PowerView User's Guide'

LS

LS

Display directory

Format: **LS [<path>] [/PATH] [/Recursive]**

For a description of the **LS** command, see **DIR**.

See also

- DIR
- ▲ 'File and Folder Operations' in 'PowerView User's Guide'
- ▲ 'Release Information' in 'Legacy Release History'

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.PENDING](#)
 - [MENU.Program](#)
 - [MENU.ReProgram](#)
 - [MENU.RESet](#)
- ▲ 'Icons' in 'PowerView 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"
```

The menu can be removed using [MENU.Delete.NAME USER.ADDMENU](#).

See also

- [MENU.AddTool](#)
 - [MENU](#)
 - [MENU.RESet](#)
- ▲ 'PowerView - Screen Display' in 'PowerView User's Guide'
▲ 'Release Information' in 'Legacy Release History'
▲ 'TRACE32 PowerView' in 'Training Basic Debugging'
▲ 'TRACE32 PowerView' in 'Training Basic SMP Debugging'

Format: **MENU.AddTool** <tooltip_text> <image> <command>

<image>: ";<predefined_image>"
" <shorttext>[,<color>[:<predefined_image>]]"

<color>: **r | R | g | G | b | B | ...**

Adds 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 [MENU.Program](#) or [MENU.ReProgram](#) command. The parameters are the same as described for the [TOOLITEM](#) statement.

<shorttext>	Max. two letters, case-sensitive, i.e. "sT" is displayed as "sT" on the button.
<color>	To receive an overview of the supported colors, choose Misc menu > Tools > Edit bitmap template .
<predefined_image>	To receive an overview of the built-in icons, choose Misc menu > Tools > Display internal icon library .

Example: Four temporary buttons are added to the main toolbar.

```
;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"
```

The button can be removed using [MENU.Delete.NAME USER.ADDTOOL](#).

See also

■ [MENU.AddMenu](#) ■ [MENU](#) ■ [MENU.Delete](#) ■ [MENU.RESet](#)

- ▲ 'PowerView - Screen Display' in 'PowerView User's Guide'
- ▲ 'Release Information' in 'Legacy Release History'
- ▲ 'TRACE32 PowerView' in 'Training Basic Debugging'
- ▲ 'TRACE32 PowerView' in 'Training Basic SMP Debugging'

Format: **MENU.Delete** <file>

Deletes a previously added menu from the TRACE32 menu definition.

See also

■ [MENU](#)

■ [MENU.AddTool](#)

MENU.Delete.NAME

Delete specified menu

[build 136824 - DVD 09/2021]

Format: **MENU.Delete.NAME** <name>

Deletes a menu specified by name.

See also

■ MENU.PENDING.List ■ MENU.PENDING.RESet ■ MENU

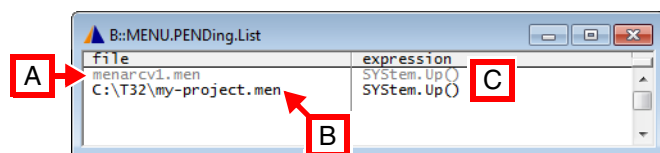
MENU.PENDING.List

List menu files waiting for compilation

Format: **MENU.PENDING.List**

Shows a list of menu files whose compilation is pending due to a **WAIT** command in the menu file (*.men).

As soon as the **WAIT** <condition> for a menu file is met, the file is compiled, added to the TRACE32 menu bar, and the menu file is removed from this list.



- A** Gray: CPU-specific menu file. It is called automatically when you select a CPU with **SYSTEM.CPU**. If the menu file (*.men) contains a **WAIT** <condition>, the menu file is automatically added to the **MENU.PENDING.List** window. The list entry is automatically removed if the condition is met or when you select another CPU.
- B** Black: Any other menu file that is called with **MENU.ReProgram** and contains a **WAIT** <condition>.
- C** Example of a **WAIT** <condition>: The menu is compiled and displayed as soon as the target is up and regular memory can be accessed.

See also

■ MENU.PENDING

MENU.PENDING.RESet

Clear list of pending menu files

Format: **MENU.PENDING.RESet**

Clears all menu files (*.men) from the list of pending menu files. See **MENU.PENDING.List**.

See also

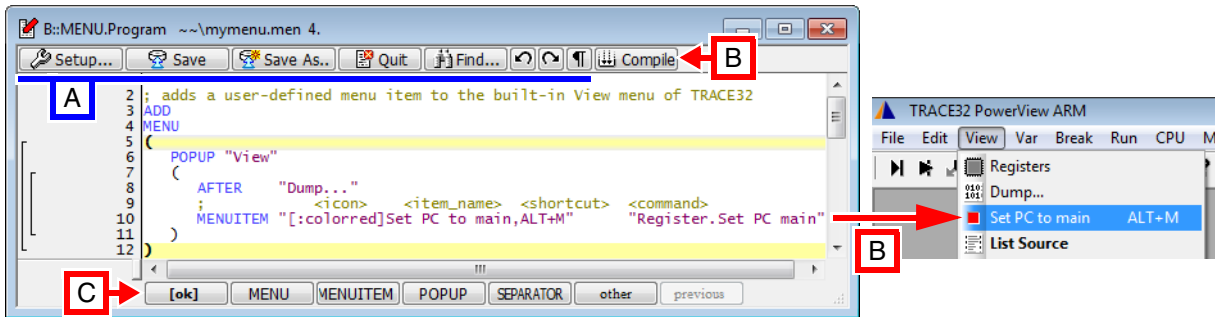
■ MENU.PENDING

Format: **MENU.Program** [*<file>*] [*<line>*] [*[/<option>*]

<option>: **AutoSave | NoSave**

Opens the **MENU.Program** editor window, where you can create menu or toolbar definition files. Using the editor, you can modify the built-in TRACE32 menus, create your own menus, and add new buttons to the TRACE32 toolbar.

The editor provides syntax highlighting, configurable auto-indentation, and an online syntax check. The input is guided by softkeys. The syntax for the definition file is described in section “[Programming Commands](#)”.



Buttons common to all TRACE32 editors:

A For button descriptions, see [EDIT.file](#).

Buttons specific to this editor:

- B** **Compile** performs a syntax check and, if an error is found, displays an error message. If the menu definition file (*.men) is error free, then the user interface is modified as defined in the *.men file. In this example, the **View** menu is modified: A user-defined menu item called **Set PC to main** is added below the **Dump** menu item.
- C** Commands for menu programming. For descriptions and examples, refer to the [MENU](#) command group as well as to the training manual listed in the **See also** block below.

<i><file></i>	The default extension for <i><file></i> is *.men .
<i><line></i> , <i><option></i>	For description of the arguments, see EDIT.file .

See also

- [MENU](#)
 - [MENU.ReProgram](#)
 - [SETUP.DropCoMmanD](#)
 - [SETUP.EDITOR](#)
- ▲ 'PowerView - Screen Display' in 'PowerView User's Guide'
 - ▲ 'Text Editors' in 'PowerView User's Guide'
 - ▲ 'Release Information' in 'Legacy Release History'
 - ▲ 'Customizable GUI Elements' in 'Training Menu Programming'

```
Format 1:      MENU.ReProgram [<file>] [/NAME "<string>"]

Format 2:      MENU.ReProgram
                (
                <menu_definition> | <main_toolbar_definition>
                )
```

Format 1: If you enter the command at the cmdline *without parameter*, then the default menu file t32.men in the system directory is executed.

With parameter, the corresponding file is compiled and executed. You receive an error message if the file contains any errors.

/NAME <string> See [“NAME Define an internal menu name”](#), page 209

Format 2: If the command is used in a PRACTICE script (*.cmm) *without* the <file> 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](#).

Example 1 - Format 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**:

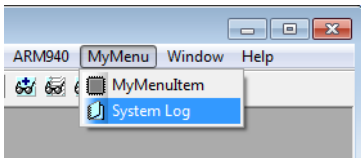
```
;your code

MENU.ReProgram ~/mymenu.men        ;add or modify menu or main toolbar
                                     ;using a *.men file

;your code
```

Example 2 - Format 2: The menu definition is embedded in a PRACTICE script (*.cmm).

```
;your code
MENU.ReProgram    ;embedded 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
```



Example 3 - Format 2: The definition of a user-defined main toolbar button is embedded in a PRACTICE script (*.cmm).

```
;your code

;embedded toolbar button definition (...) here
MENU.ReProgram
(
  ADD
  TOOLBAR
  ( ;          <tooltip>          <icon>          <command>
    TOOLITEM "MyToolbarButton" "[:colors]" "Register.view /SpotLight"
  )
)

;your code
```



Example 4 - Format 2: The opening block delimiter (&+ allows you to pass a PRACTICE macro to the user-defined main toolbar button, which is embedded in a PRACTICE script (*.cmm).

```
LOCAL  &myPath
&myPath=OS.ENV(SystemDrive)+"/SVN/demo"

MENU.ReProgram
(&+
  ADD
  TOOLBAR
  ( ;      <tooltip>      <icon>      <command>
    TOOLITEM      " "      "[:folder]"      "OS.OPEN  ""&myPath"" "
  )
)
```

For more information about passing PRACTICE macros to embedded script blocks, see [“Switching PRACTICE Macro Expansion ON or OFF”](#) in PRACTICE Script Language User’s Guide, page 11 (practice_user.pdf).

See also

- [MENU](#)
- [MENU.Program](#)
- ▲ [‘PowerView - Screen Display’](#) in [‘PowerView User’s Guide’](#)
- ▲ [‘Release Information’](#) in [‘Legacy Release History’](#)

Format: **MENU.RESet**

Restores the default configuration of the menus and the main toolbar.

See also

■ [MENU](#)

■ [MENU.AddMenu](#)

■ [MENU.AddTool](#)

▲ ['PowerView - Screen Display'](#) in ['PowerView User's Guide'](#)

▲ ['TRACE32 PowerView'](#) in ['Training Basic Debugging'](#)

▲ ['TRACE32 PowerView'](#) in ['Training Basic Debugging'](#)

▲ ['TRACE32 PowerView'](#) in ['Training Basic SMP Debugging'](#)

▲ ['TRACE32 PowerView'](#) in ['Training Basic SMP Debugging'](#)

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:

Scenario	Result
Usage of a <i>different</i> menu file or <i>different</i> PRACTICE script	The existing menu is retained and the new menu items are added.
Definition is embedded in the <i>same</i> PRACTICE script file (*.cmm)	Executing the embedded block again: <pre>MENU.Program (...)</pre> replaces the last embedded block with the new one.
Definition is in the <i>same</i> menu file (*.men)	Executing MENU.Program <code><my_men_file></code> with the same path and name replaces the previous menu definition with the new menu definition.

ADDDHERE

Define hook

Format: **ADDDHERE**

When items are added to an existing menu, they are usually added to the end of the menu. The **ADDDHERE** command allows to choose a different insertion point for additional items.

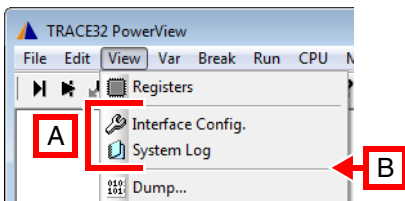
Format: **AFTER** "<menu_item_name>"

```
MENU.ReProgram ;embed menu definition in a PRACTICE script file (*.cmm)
(
  ADD
  MENU
  ( ;in the View menu...
    POPUP "&View"
    ( ;... place two new menu items after the menu item "Registers"
      AFTER "Registers"
      MENUITEM "[:syslog]System Log" "WinResist.WinExt.SYSem.LOG.state"

      AFTER "Registers"
      MENUITEM "[:config]Interface Config." "WinResist.IFCONFIG.state"

      ;... place a separator after the menu item "Registers"
      AFTER "Registers"
      SEPARATOR

      ;... place a separator before the menu item "Dump..."
      BEFORE "Dump..."
      SEPARATOR
    )
  )
)
```



A Two new menu items and a separator have been inserted by the menu command **AFTER**.

B A new separator has been inserted by the menu command **BEFORE**.

Format: **BEFORE** "<menu_item_name>"

For an example, see menu command **AFTER**.

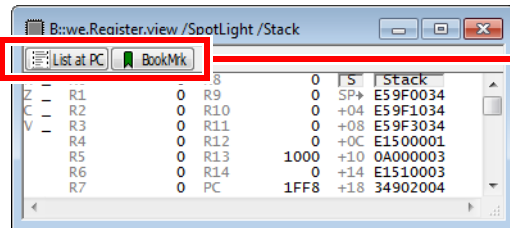
Format: **BUTTONS** <window>

Adds one or more user-defined local buttons to a window. The **BUTTONS** command can be used on the top-level of the script only. User-defined local buttons cannot be added to all windows.

The <window> parameter must be the short form of a command that opens a window, e.g. **F.** for **Frame.view** or **R.** for **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 **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.

- *With an icon*, the max. width of a button text is 6 characters, if you do not use **WIDTH**.
- *Without an icon*, the max. width of a button text is 9 characters, if you do not use **WIDTH**.



Two user-defined local buttons in the [Register.view](#) window

This script adds two user-defined local buttons to the [Register.view](#) window, as shown above, for opening the [List.auto](#) and [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.

```
MENU.ReProgram
(
ADD
BUTTONS "R."
(
;1st button
MENUITEM "[:list]List at PC"
(;determine whether the named window is already open
IF WINDOW.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
```

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.

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.

For deleting a TOOLBAR button the <tooltip_text> of the TOOLITEM definition is used instead.

```
ADD
MENU
(
  POPUP "&OSE Delta"
  (
    DELETE   "Enable OSEDelta awareness"
    DEFAULT
    MENUITEM "Display &Processes"      "TASK.DProc"
    ...
  )
)
ADD
TOOLBAR
( ;previous definition of MyToolbarButton
;TOOLITEM "MyToolbarButton" "[:colors]" "Register.view /SpotLight"
DELETE   "MyToolbarButton"
)
```

Format: **ELSE**

Used together with the **IF** statement to define a block that is only compiled when the **IF** condition is false.

Format: **ENABLE** <condition>

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.

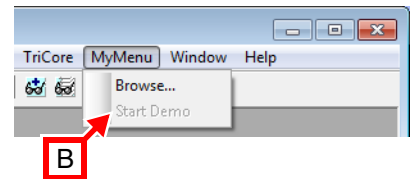
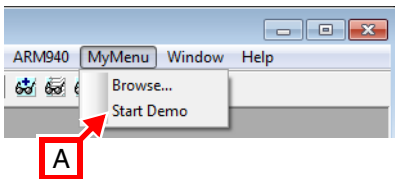
```

LOCAL &path &exe                                ;declare two PRACTICE macros
&path="~/demo/arm/compiler/arm"                ;path to PRACTICE demo scripts
&exe=OS.PresentExecutableFile()                ;get path and file name of TRACE32
                                                ;executable
&exe=OS.FILE.NAME(&exe)                        ;return just the file name

MENU.ReProgram ;embeds a menu definition in a PRACTICE script
(&           ;'&' activates the PRACTICE macro expansion
  ADD
  MENU
    ( ;this menu block creates a user-defined menu with two menu items
      POPUP "MyMenu"
        ( ;this menu item is always enabled
          MENUITEM "Browse..." "ChDir.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" "ChDir.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.

Format: **HELP** <*name*>

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.

Format: **MENU** [*<menu_type>*]

<menu_type>: *<cmd>* | *<special_name>*

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.

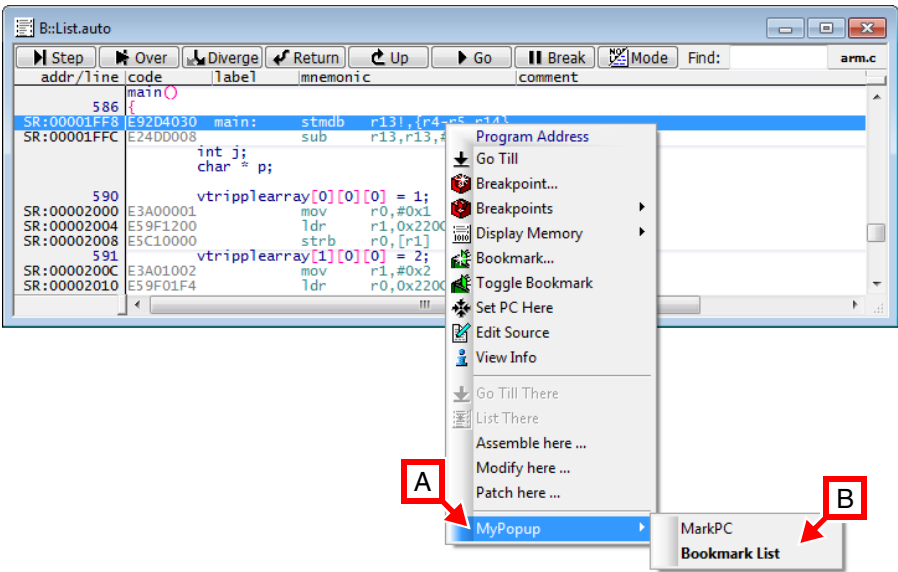
<i><cmd></i>	Short form of a command. For information about command short forms, see “Long Form and Short Form of Commands and Functions” (ide_user.pdf).
<i><special_name></i>	"DATA" "VAR" DATA: adds a local popup to all Data windows (e.g. Data.dump , List.auto). VAR: adds a local popup to all Var windows (e.g. Var.View , Var.Watch)

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"
    )
  )
)
```

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.

Format: **MENUITEM** *<name>* [*<command>*]

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 **MENUITEM** definition.

NAME

Define an internal menu name

Format: **NAME** *<name>*

Defines an internal name for a menu. The internal name is not displayed on the GUI. Internal names can be used to check if a specific menu is loaded using **MENU.EXIST**(*<name>*) or to delete menus using **MENU.Delete.NAME** *<name>*.

The **NAME** must be set on top-level before the **ADD** command.

<name>

The MENU name must only contain the following characters:
[a-z | A-Z | 0-9 | _]

The MENU name is case-insensitive.

Format: **PERMENU** <file> <name> [<level>]

Creates a menu or submenu structure which represents the **TREE** elements of the stated peripheral <file> (*.per). Clicking one of these elements will open a new **PER.view** window showing the selected item.

<file>	Peripheral file from which the menu or submenu will be created. Can be empty ("") to reference the CPU specific default peripheral file (*.per).
<name>	The name of the root entry of the menu or submenu. If empty (""), the name will be extracted from the <file> name.
<level>	Maximum cascading level. Range 1 .. 255.

To update the PERMENU entries after changing the contents of the peripheral file:

- Case 1: <file> is an empty string.
Execute **PER.ReProgram** to update the **PERMENU** entries.
- Case 2: <file> is not empty.
Execute **PER.ReProgram**, and then **MENU.RESet** to update the **PERMENU** entries.

Example 1: The TRACE32 built-in **CPU** menu is extended by a submenu called **Peripherals**. Using the **PERMENU** command, the new submenu is populated with the tree elements defined in the CPU specific default peripheral file (*.per).

```
ADD
MENU
(
  POPUP "&CPU"
  (
    PERMENU " " "Peripherals"
    ; ...
  )
)
```

Menu blocks, such as the one shown above, can be stored in the following menu files (*.men):

- In the CPU specific menu file (men*.men): As a result, tree elements defined in the CPU specific default peripheral file (*.per) are automatically added to a menu or submenu when you select a CPU with **SYSem.CPU**.
- In an extra menu file (*.men): The resulting menu or submenu is only available if you execute the menu file with the command **MENU.ReProgram** <file>.men

Example 2: Custom peripheral file

```
ADD
MENU
(
  POPUP "&CPU"
  (
    PERMENU "perMyPerfile.per" "My Peripherals"
    ;...
  )
)
```

POPUP

Popup definition

Format: **POPUP** <*name*>

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.

Format: **REPLACE**

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
    MENUITEM "Step Over Call,F8" "Step.Over"
  )
  ...
)
```

Format: **SEPARATOR**

Inserts a separator in a menu or toolbar.

Format: **SUBROUTINE**

Defines a PRACTICE subroutine. The subroutine can be called by all MENUITEM and TOOLITEM items that are defined in the same menu file or menu block. Subroutines can be used to consolidate functions that are used by multiple menu items or used by menu items and toolbar items together. The subroutine is not accessible from outside the menu program in which it was defined.

```
ADD
MENU
(
  POPUP "MemoryActions"
  (
    MENUITEM "Init"  "GOSUB InitMemory"
    MENUITEM "Erase" "GOSUB EraseMemory"
    MENUITEM "Show Memory" "Data.dump 0--0xFFFF"
  )
)
TOOLBAR
(
  TOOLITEM "Init"  "IN" "GOSUB InitMemory"
  TOOLITEM "Erase" "ER" "GOSUB EraseMemory"
)

SUBROUTINE AccessCheck
(
  IF !SYSTEM.Up() || STATE.RUN()
  (
    DIALOG.OK "Error: Can not access memory"
    END ;end script execution
  )
)

SUBROUTINE EraseMemory
(
  GOSUB AccessCheck
  Data.Set 0--0xFFFF 0xFF
)

SUBROUTINE InitMemory
(
  GOSUB AccessCheck
  Data.Set 0--0xFFFF 0x00
)
```

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

Toolbar definition

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

Item definition

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
x, X	dark grey / black
w, W	light grey / white
@	foreground color
(blank)	light grey (background color)
s	foreground shadow
S	foreground highlight

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.

```
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
      XXXX
      XXXXXXXX
      XXXXXXXX
      XX XXXX
      XX XXX
      XX XX
      XX
      XX
      XX XX
      XXX XXX
      XXXXXXXXXX
      XXXXXX
    ]
  )
)
```



```
Format          WAIT [<condition>]

<condition>:    <address> | <expression> | <boolean_expression>
```

The **WAIT** command is available for all architectures and menu files, but it should only be used when required (i.e. **IF** with target-dependent values). Most architectures will probably *not* require **WAIT**. But if there is a need to use **WAIT**, then the recommendation is to use **WAIT** at the beginning of a menu file (*.men file).

You can view the list of menu files waiting for compilation with the command **MENU.PENDING.List**.

<address> Target address which has to be accessible; see [example 2](#).

<expression> TRACE32 expression which can be evaluated; see [example 3](#).

<boolean_expression> Boolean expression which has to be true; see [example 4](#).

There are four ways to use the **WAIT** command, see examples 1 to 4.

Example 1: Wait with compilation until the target is up and regular memory can be accessed (this usually means that the target is stopped).

```
WAIT
```

Example 2: Wait with compilation until the target is up and the given memory address can be accessed.

```
WAIT ETM:0
```

Example 3: Wait with compilation until the target is up and the expression can be evaluated (the result does not matter).

```
WAIT Data.Long(D:0)
```

Example 4: Wait with compilation until the target is up and the boolean expression evaluates to true.

```
WAIT Data.Long(D:0) !=0
```

Format: **WIDTH** <arg>

<arg>: **NORMAL | WIDER | WIDEST | SMALLER | SMALLEST**

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"

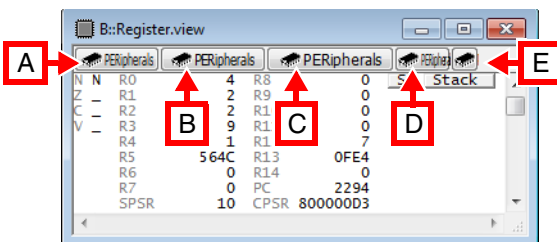
    WIDTH WIDER
    MENUITEM "[:chip]PERipherals" "PER.view"

    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

Format: **MKDIR** <path>

This built-in TRACE32 command **MKDIR** creates a new subdirectory.

Example 1:

```
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.

```
LOCAL &folder
&folder="c:\temp2"

;if the folder does not exist,
IF OS.DIR(&folder)==FALSE()
(
    ;then create it
    MKDIR &folder
)

;open the folder in Windows Explorer
OS.OPEN "&folder"
```

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

■ [MKTEMP](#)

■ [ChDir](#)

■ [RMDIR](#)

▲ 'File and Folder Operations' in 'PowerView User's Guide'

Format: **MKTEMP** [&<macro>],[<template>"] [/<option>]

<option>: **Directory** | **DryRun** | **ID7**

Creates a new empty file or directory, based on a <template> name. The name of the created file is printed to the **AREA** window. Use the pre-command **SILENT** to suppress the output to the **AREA** window and the TRACE32 message line.

<macro>	The name of the created file or directory is stored in the given PRACTICE <macro> name.
<template>	<p>The template on which the name of the new file or directory is based.</p> <p>The template will be expanded by 12 decimal digits to create a unique file name. These extra characters are added in one of the following positions in the name:</p> <ul style="list-style-type: none"> • The first asterisk (*) • Before the last dot (if there is no asterisk) • At the end of the file name (if there is neither an asterisk nor a dot) <p>The first 11 decimal digits are the UNIX timestamp.</p> <p>If the command is called without a <template>, the ID of the PowerView GUI plus an underscore will be used as <template>. The file extension is “.tmp”. Default template: OS.ID()+“_*.tmp”</p> <p>By default, the file is created in the temporary directory of TRACE32. But if the <template> contains a directory part, that directory is used.</p> <p>If the directory part is not an absolute path, then the directory relative to the working directory is used.</p> <p>An error is shown if the directory specified by the directory part does not exist.</p>
Directory	Create a directory instead of a file.
DryRun	<p>Just get a file name but don't create the actual file.</p> <p>The use of this option is potentially unsafe if competing executables happen to suggest the same file name at the same time and if one executable later on creates a file based on this name.</p>
ID7	The UNIX timestamp as a 7-digit alphanumeric string.

The following table shows the difference between the command **MKTEMP** and the PRACTICE function **OS.TMPFILE()**.

MKTEMP command vs. OS.TEMP() function:

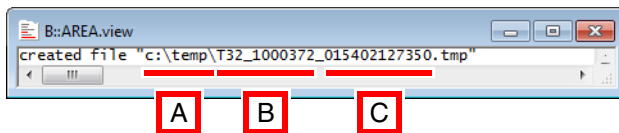
MKTEMP &file	&tempfilename= OS.TMPFILE()
<ul style="list-style-type: none">Creates a file or a folder.You can specify file name, extension, and the folder name, or let TRACE32 make the decision for you.	<ul style="list-style-type: none">Suggests a file name, but does not create the file.The file name never has an extension.The file name is concatenated with the name of the temporary directory of TRACE32.

Examples

Example 1: This script line creates a file with the extension *.tmp in the temporary directory of TRACE32. The file name consists of the ID of the PowerView GUI where the script line is executed plus 12 decimal digits to create a unique file name.

```
MKTEMP ;path and name of the new file are printed to  
;the AREA window and TRACE32 message line
```

Result:



- A** Temporary directory of TRACE32. See also PRACTICE function [OS.PresentTemporaryDirectory\(\)](#).
- B** ID of the PowerView GUI. See also [OS.ID\(\)](#).
- C** The first 11 of the 12 decimal digits are the UNIX timestamp.

Example 2: This script creates a temporary file without a specific extension in the working directory of TRACE32. Path and file name are assigned the PRACTICE macro &tmpfile. Based on the PRACTICE macro &tmpfile, the temporary file is deleted later on with the [RM](#) command.

```
MKTEMP &tmpfile ".\*" ;the path prefix .\ expands to the working  
;directory of TRACE32  
;your code  
  
RM &tmpfile
```

Example 3: In this script, **MKTEMP** creates the new subfolder `logs_` with a 7-digit alphanumeric suffix in the working directory of TRACE32 (. \). Using the **OPEN** command, a new file with a user-defined name is created in the subfolder, and then the current local date is inserted into the new file.

```
;create new subfolder and assign folder path to PRACTICE macro &folder
MKTEMP &folder ".\logs_*" /Directory /ID7

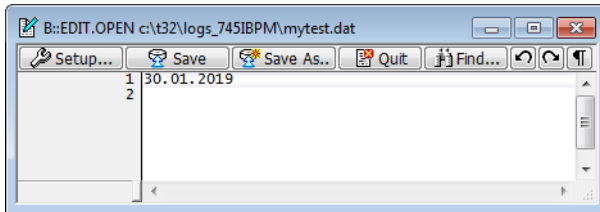
;create a file with a user-defined file name and then write the current
;local date to the file
OPEN #1 &folder\mytest.dat /Create
WRITE #1 FORMAT.UnixTime("d.m.Y",DATE.UnixTime(),DATE.utcoffset())

;your code

CLOSE #1

;let's display the result in the TRACE32 editor
EDIT.OPEN &folder\mytest.dat
```

Result:



See also

■ MKDIR
■ WRITE

■ APPEND
□ OS.TMPFILE()

■ RM

■ RMDIR

Format: **MV** *<oldname>* *<newname>*

Renames a file.

<oldname>,
<newname>

Wildcard characters are **not** supported.

See also

- [REN](#)
- [OS.FILE.readable\(\)](#)
- ▲ 'File and Folder Operations' in 'PowerView User's Guide'

See also

- [OS.Area](#)
- [OS.Hidden](#)
- [OS.screen](#)
- [OS.Window](#)
- [OS.FILE.NAME\(\)](#)
- ▲ 'OS Functions' in 'PowerView Function Reference'
- [OS.Command](#)
- [OS.OPEN](#)
- [OS.SetENV](#)
- [OS.ENV\(\)](#)
- [OS.PresentSystemDirectory\(\)](#)

Overview OS

The **OS** commands allow the execution of host commands within TRACE32 on the system shell of the underlying host operating system.

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.

Comparison of the OS Commands

OS Command	Output	Blocking / Non-Blocking
OS.screen	No output in TRACE32.	Non-Blocking
OS.Area	Output in AREA window.	Blocking
OS.Window	Output in a TRACE32 window of the same name.	Blocking
OS.Hidden	No output at all.	Blocking
OS.Command	Output in system shell	Dependent on shell
OS.OPEN	No output in TRACE32.	Non-Blocking

Comparison of the OS Commands to their Windows and Linux Counterparts

This OS command ... corresponds to this...	Windows Command	Linux Command
OS.screen <code><cmd></code>	<code><cmd></code>	<code>sh -c <cmd> &</code>

This OS command ... corresponds to this...	Windows Command	Linux Command
OS.Area <cmd>	cmd /c <cmd> > "tmpfile" && type "tmpfile" && del "tmpfile"	sh -c <cmd> > "tmpfile" && cat "tmpfile" && rm "tmpfile"
OS.Window <cmd>	cmd /c <cmd> > "tmpfile" && cmd /c notepad "tmpfile" && del "tmpfile"	sh -c <cmd> > "tmpfile" && emacs "tmpfile" && rm "tmpfile"
OS.Hidden <cmd>	cmd /c <cmd> > NUL	sh -c <cmd> > /dev/null
OS.Command <cmd>	cmd /c <cmd>	sh -c <cmd>
OS.OPEN	cmd /c start "" "<string>"	xdg-open "<string>"

Blocking and Non-Blocking OS Commands

The purpose of blocking **OS** commands is to prevent forks in PRACTICE scripts (*.cmm). Whereas non-blocking **OS** commands allow forks in PRACTICE scripts.

Blocking	Non-Blocking
<p>OS.Area, OS.Window, and OS.Hidden</p> <p>These OS commands block the execution of the TRACE32 application and wait for the host command to finish. Once the host command has finished, PRACTICE script execution continues.</p> <p>Use a blocking OS command if you want the PRACTICE script to process the output of the host command.</p>	<p>OS.screen and OS.OPEN do not block PRACTICE script execution. Consequently, the PRACTICE script and the host command will run in parallel.</p> <p>OS.Command: The behavior depends on the system shell.</p> <ul style="list-style-type: none"> • On Windows: always non-blocking. • On Linux/Unix, append an ampersand '&' to the host command to get a non-blocking behavior.

What is the difference between the commands ...?

OS.Window	OS.Area
<p>Executes the host command and re-routes all outputs of this host command to a TRACE32 window called OS.Window.</p> <p>The window opens <i>automatically</i>, any further user interaction is <i>not</i> necessary.</p>	<p>Redirects the host command output to the active TRACE32 message area.</p> <p><i>It is up to you</i> to decide when you want to view the output by executing the AREA.view command at the TRACE32 command line.</p>

Both commands are useful for displaying the output of a host command in TRACE32, e.g. a directory listing of the host command `dir`.

What is the difference between the commands ...?

OS.screen	OS.Command
<p>Opens just the command prompt window of the host shell, where you can execute a host command.</p> <p>OS.screen is not running in a system shell on Windows.</p>	<p>Opens a system shell and starts the host command within this shell.</p> <p>OS.Command allows you to redirect the output of the host command with the redirection symbol (>).</p>

NOTE: The Windows `dir` and the TRACE32 **DIR** command are **not** identical.

Format: **OS.Area** <cmdline>

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
```

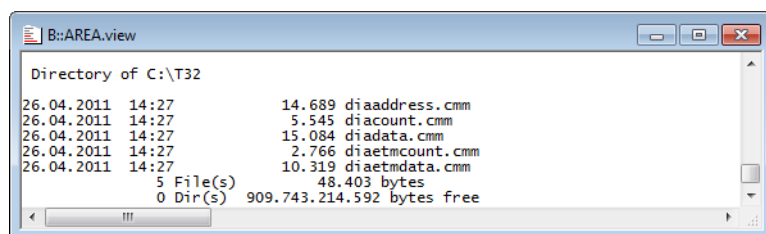
```
AREA.view
```

```
;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\(\)](#)

□ [OS.NEXTFILE\(\)](#)

▲ ['Host Commands' in 'PowerView User's Guide'](#)

Format: **OS.Command** [*<cmdline>*]

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 *<cmdline>* 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.

Example 1 - Copy files (Windows)

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.

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.

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

The environment variables are written to a txt file, which is then opened in an editor.

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: 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.

```
;write environment variables to txt file
OS.Command env > $TMPDIR/env.txt

;open txt file in an editor
OS.Command emacs $TMPDIR/env.txt &
```

See also

- [OS](#)
- [OS.screen](#)
- [OS.ENV\(\)](#)
- ▲ ['Host Commands' in 'PowerView User's Guide'](#)

Format: **OS.Hidden** <cmdline>

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.

Example 1:

```
;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
```

Example 2:

```
;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 'PowerView User's Guide'](#)

Format: **OS.OPEN** <file> | <path> | "<url>"

Opens a file, folder, or URL in the default application of the operating system. That is, for a file, **OS.OPEN** performs the same operation as a double-click on a file in the file explorer of the operating system.

NOTE: If you are using **OS.OPEN** in a PRACTICE script to open a URL you must enclose the URL in double-quotes. Otherwise the two slashes after the schema of a URL will be handled as the beginning of a comment by the PRACTICE interpreter.

Example 1: This script line opens the TRACE32 demo folder in the file explorer of the operating system.

```
OS.OPEN c:\t32\demo
```

Example 2: This script line opens the Lauterbach website in the default web browser of your operating system.

```
OS.OPEN "https://www.lauterbach.com"
```

Example 3: This script exports two *.csv files from TRACE32 and opens the two files in one and the same Excel instance.

```
;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
OS.OPEN func.csv

;the second file will also open in that instance, i.e. another Excel
;instance will not be started
OS.OPEN variables.csv
```

See also

■ [OS](#)

■ [OS.screen](#)

- ▲ ['Host Commands' in 'PowerView User's Guide'](#)
- ▲ ['Release Information' in 'Legacy Release History'](#)

Format: **OS.screen** [*<cmdline>*]

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.

TRACE32 Command Line

Command Shell of the Host

```
;Call up the command shell  
OS.screen
```

```
rem Change from a network drive to the  
rem system directory of TRACE32  
J:\>cd /d C:\T32
```

```
rem List all PRACTICE script files  
rem residing in C:\T32  
C:\T32>dir *.cmm
```

```
rem Close the command shell  
C:\T32>exit
```

```
;Continue your TRACE32 session  
;...
```

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](#)
 - [OS.OPEN](#)
 - [OS.SetENV](#)
 - [OS.Window](#)
 - [OS.ENV\(\)](#)
- ▲ 'Host Commands' in 'PowerView User's Guide'
▲ 'Release Information' in 'Legacy Release History'

OS.SetENV

Set operating system environment variables

[build 135727 - DVD 09/2021]

Format: **OS.SetENV** <name> <value>

While starting an external executable, then it might need to set environment variables, which are evaluated by the external executable.

See also

- [OS](#)
- [OS.screen](#)

Format: **OS.Window** *<cmdline>*

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)

```
;display a listing of the TRACE32 system directory in a TRACE32 window
OS.Window dir c:\t32

;display a tree structure of the demo folder in a TRACE32 window
OS.Window tree c:\t32\demo /f /a

;display the environment variables of the host in a TRACE32 window
OS.Window set
```

Example 2 - Directory listing and environment variables (Linux)

```
;display a listing of the TRACE32 system directory in a TRACE32 window
OS.Window ls -l /home/user/t32

;display the environment variables of the host in a TRACE32 window
OS.Window env
```

See also

- [OS](#)
 - [OS.Hidden](#)
 - [OS.screen](#)
 - [OS.ENV\(\)](#)
- ▲ 'Host Commands' in 'PowerView User's Guide'

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.

Examples:

```
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 'PowerView User's Guide'](#)

Format: **PATCH** [*<file>* [*<offset>*]] *<data>* ...

Patches bytes in a binary file.

See also

■ [DUMP](#)

■ [EDIT](#)

■ [TYPE](#)

▲ ['File and Folder Operations'](#) in ['PowerView User's Guide'](#)

PATH Define search paths for files used by TRACE32 commands

The command group **PATH** defines or modifies the search path for files which are used by the TRACE32 commands listed below.

Please be aware of the following:

- The search paths are only used for file names **without** a path specification. The files will be searched first in the working directory.
- No recursive search will be done.
- **The directory names are case-sensitive.**
- This command **cannot be used** to search for the source files for HLL debugging.
- If **ChDir** is used with the command, the search paths are **deactivated**.
E.g. the command `CO.DO my_script.cmm` will only find the script if it is located in the current working directory.

Searching is enabled for the following commands:

Commands
DO, RUN, PEDIT, PSTEP
PER.Program, PER.ReProgram, PER.view
Analyzer.Program, Analyzer.ReProgram
Break.Program, Break.ReProgram
Data.Program, Data.ReProgram
DIALOG.Program, DIALOG.ReProgram, DIALOG.view
Integrator.Program, Integrator.ReProgram
MENU.Program, MENU.ReProgram
PATTERN.Program, PATTERN.ReProgram
PERF.Program, PERF.ReProgram
Probe.Program, Probe.ReProgram
Trace.FindProgram, Trace.FindReProgram

See also

- [PATH.Delete](#)
 - [PATH.DOWN](#)
 - [PATH.List](#)
 - [PATH.RESet](#)
 - [PATH.Set](#)
 - [PATH.UP](#)
 - [PATH.NUMBER\(\)](#)
 - [PATH.PATH\(\)](#)
- ▲ 'PATH Functions' in 'PowerView Function Reference'

Format: **PATH** [+ | -] [<path> ...] (deprecated)

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.

A **PATH** command without any parameter removes all previous defined search directory entries.

This command **cannot be used** to specify source file search paths for HLL debugging. Please use command group **sYmbol.SourcePATH** instead.

```
PATH \t32\exam\cmm \use\me\mycmm      ; define two search directories
PATH + ..\cmm                          ; add one more directory
PATH - \use\me\mycmm                  ; delete a certain directory
PATH                                   ; delete all directory entries
```

PATH.Delete

Delete search path

Format: **PATH.Delete** <directory> ...

Delete one or more search path entries.

The directory names are treated case-sensitive - even under Windows.

```
PATH.Delete W:\t32\exam\cmm W:\use\mycmm ; delete 2 search
                                           ; directories

PATH.Delete ..\cmm                        ; delete one directory

PATH.Set      W:\mycmms                    ; define one directory
PATH.Delete  w:\Mycmms                     ; given directory isn't
                                           ; found

PATH.Delete W:\mycmms                     ; delete one directory
```

See also

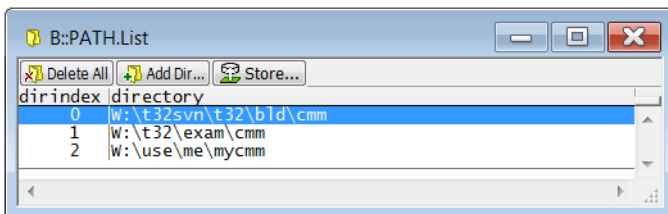
■ [PATH](#)

▲ ['Release Information' in 'Legacy Release History'](#)

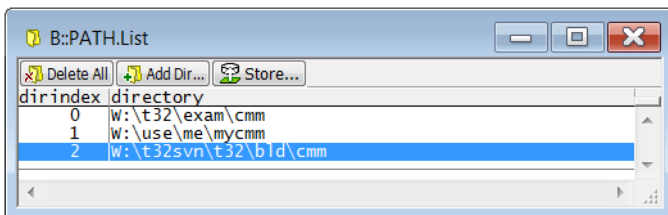
Format: **PATH.DOWN** <directory>

Defines an additional directory entry at the end of the search path order. The files will be searched first in the current or specified directory. An existing entry with the same directory name will be deleted automatically to avoid duplicate entries.

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



```
PATH.DOWN ..\cmm ; move a directory at the end
; new directory order:
```



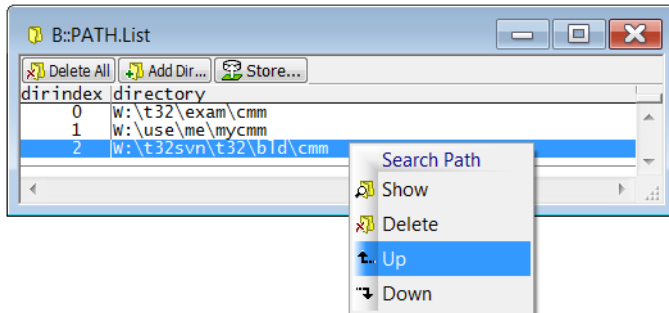
See also

- [PATH](#)

- ▲ ['Release Information' in 'Legacy Release History'](#)

Format: **PATH.List**

Displays the defined search path directory entries. The directory index represents the search order **after** the current or specified directory.



The actual search path settings are saved with command **STORE** in combination with the keyword **PATH**.

See also

- [PATH](#)
- [PATH.NUMBER\(\)](#)
- [PATH.PATH\(\)](#)
- ▲ ['Release Information' in 'Legacy Release History'](#)

PATH.RESet

Reset search path

Format: **PATH.RESet**

Deletes all search path entries.

See also

- [PATH](#)
- ▲ ['Release Information' in 'Legacy Release History'](#)

Format: **PATH.Set** <directory> ...

Defines the search path for some frequent used file formats (e.g. PRACTICE scripts).

The files will be searched first in the working directory and then in all defined search path directories.

e.g. DO abc.cmm

If a file name contains a certain specified directory a search will be restricted exactly to this directory.

e.g. DO C:\t32\abc.cmm

DO .\abc.cmm

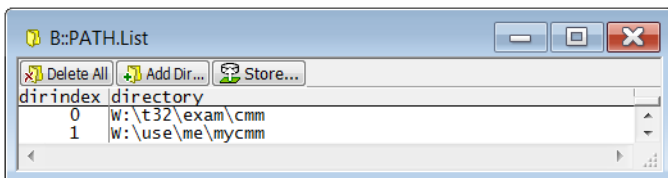
DO your\abc.cmm

The actual search path settings can be saved with command **STOre** in combination with the keyword **PATH**.

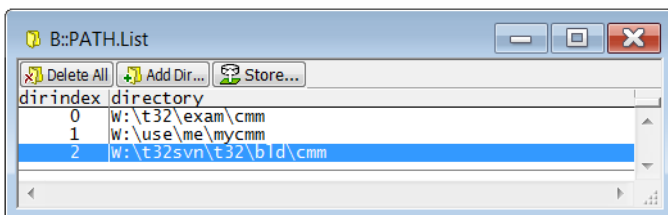
This command **cannot be used** to specify source file search paths for HLL debugging.

Please use command group **sYmbol.SourcePATH** instead.

```
PATH.Set \t32\exam\cmm \use\me\mycmm ; define two search
; directories
```



```
PATH.Set ..\cmm ; add one more directory
```



See also

■ PATH

▲ 'File and Folder Operations' in 'PowerView User's Guide'

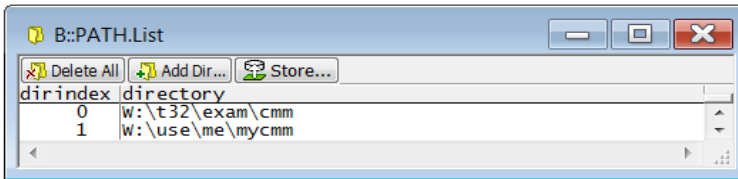
▲ 'Release Information' in 'Legacy Release History'

Format: **PATH.UP** <directory>

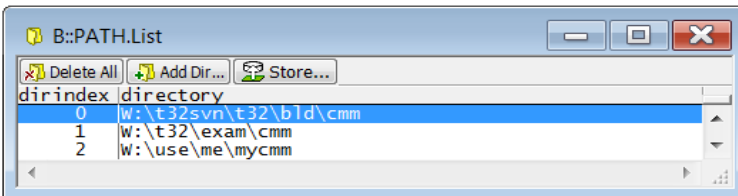
Defines an additional directory entry at the beginning of the search path. The files will be searched first in the current or specified directory.

An existing entry with the same directory name will be deleted automatically to avoid duplicate entries.

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



```
PATH.UP ..\cmm ; add a directory at the top
; directory order now:
```



See also

■ [PATH](#)

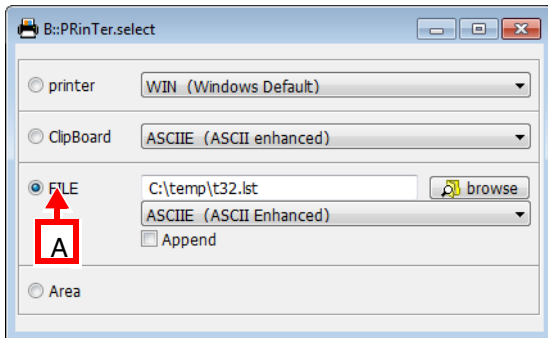
▲ ['Release Information' in 'Legacy Release History'](#)

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, a PRACTICE script (*.cmm), or via 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 the **FILE** option, see **PRinTer.FILE**.

For PRACTICE script examples, see:

- [PRinTer.FILE](#)
- [PRinTer.OPEN](#)
- [PRinTer.HardCopy](#)
- [PRinTer.Area](#)

See also

- | | | | |
|----------------------------------|-------------------------------------|------------------------------------|------------------------------------|
| ■ PRinTer.Area | ■ PRinTer.ClipBoard | ■ PRinTer.CLOSE | ■ PRinTer.CONFIG |
| ■ PRinTer.EXPORT | ■ PRinTer.FILE | ■ PRinTer.FileType | ■ PRinTer.HardCopy |
| ■ PRinTer.OFFSET | ■ PRinTer.OPEN | ■ PRinTer.PRINT | ■ PRinTer.select |
| ■ PRinTer.SIZE | ■ PRINT | ■ WinPrint | ■ WinPRT |

- ▲ 'PRINTER Function' in 'PowerView Function Reference'
- ▲ 'Printer Operations' in 'PowerView User's Guide'

Format: **PRinTer.Area** [*<format>*]

<format>: **ASCII**E

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**

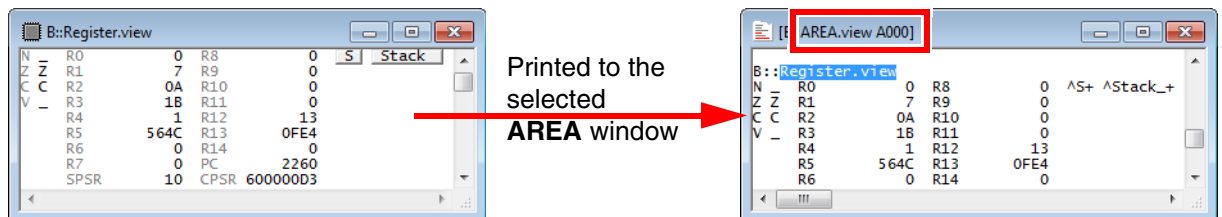
To select an **AREA** window to which you want to re-route the printer output, use the **AREA.Select** command.

<i><format></i>	If the parameter <i><format></i> is omitted, the format used to print to clipboard stays unchanged.
ASCII E	Enhanced ASCII format, underlines are displayed, graphic characters are converted and displayed as ASCII characters where feasible.

Example:

```
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
```



See also

■ [PRinTer](#)

■ [PRinTer.select](#)

Format:	PRinTer.ClipBoard [<format>] PRinTer.ClipType (deprecated)
<format>:	ASCIIE 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.
ASCIIE	Enhanced ASCII file, underlines are displayed, graphic characters are displayed as ASCII characters.
CSV	Comma-separated value.
XHTML XML (deprecated)	XML-formatted file with HTML tags.

See also

- [PRinTer](#)
- [PRinTer.select](#)
- ▲ 'Window System' in 'PowerView User's Guide'

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 'PowerView User's Guide'
- ▲ 'Printer Operations' in 'PowerView User's Guide'

See also

- [PRinTer.CONFIG.HEADER](#)
- [PRinTer.CONFIG.OFFSET](#)
- [PRinTer.CONFIG.SIZE](#)
- [PRinTer](#)
- [PRinTer.select](#)

PRinTer.CONFIG.HEADER

Print window title

[build 147215 - DVD 09/2022]

Format: **PRinTer.CONFIG.HEADER [ON | OFF]**

The window title is printed as first line of the output.

Default: ON

See also

- [PRinTer.CONFIG](#)

PRinTer.CONFIG.OFFSET

Specify print-out borders

Format: **PRinTer.CONFIG.OFFSET [<columns>] [<lines>]**

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.

Example:

```
PRinTer.select LJL           ; choose printer
PRinTer.CONFIG.OFFSET 12.   ; leave space for perforation
PRinTer.CONFIG.SIZE 80.     ; adjust printout size, make it smaller
WinPrint.HELP Data.dump    ; print chapter of manual
```

See also

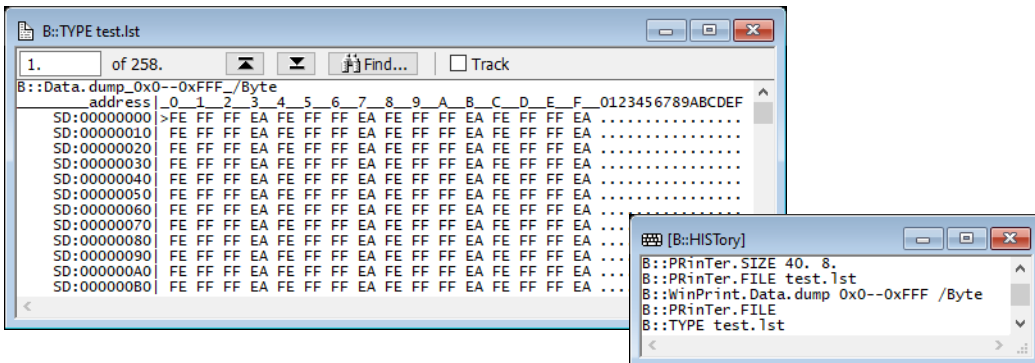
- [PRinTer.CONFIG](#)

Format: **PRinTer.CONFIG.SIZE** [*<columns>*] [*<lines>*]

Adjusts 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.

Example:

```
PRinTer.CONFIG.SIZE 70. 50.      ; make printer-output smaller
PRinTer.CONFIG.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.CONFIG](#)
- ▲ ['Printer Operations' in 'PowerView User's Guide'](#)

Format 1: **PRinTer.EXPORT.<file_format> [<file>] [/Append]**

<file_format>: **ASCIIE | CSV | XHTML**

Format 2: **default [<file>] [/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	<p>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.</p>
CSV	<p>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.</p>
XHTML	<p>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.</p>

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.

Format 2:

default (deprecated)	Sets the output format to CSV (Comma-Separated Values), but does not 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 . NOTE: If <i><file></i> is omitted, the default file name t32.lst is used.
-----------------------------	---

Options for Format 1 and Format 2:

<i><file></i>	In order to simplify multiple file generation, a decimal number contained in the file name (e.g. exam01.csv) is incremented automatically after each print to that file. If <i><file></i> is omitted, the printer output gets redirected to the previously chosen output file name (incremented if the file name contained a decimal number). And PRinTer.EXPORT.<file_format> will only append the extension.
Append	Use the option Append , to append new data to the existing file. Without Append , contents are overwritten if the file already exists.

Examples

Example 1: The file name extension omitted by the user is added automatically by TRACE32. Using the **Append** option, three windows are printed to the same file.

```
;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"
```

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <!DOCTYPE html>
3 <?xml-stylesheet type="text/xsl" href="file:///c:/myfiles/mywinprint.xsl"?>
4 <html xmlns="http://www.w3.org/1999/xhtml">
5 <head>
6 <title> TRACE32 PowerView ARM</title>
7 <meta name="date" content="2016-08-10T10:09:32Z"/>
8 <meta name="generator" content="TRACE32 PowerView for ARM (S.2016.08.000075909)"/>
9 <style type="text/css">body,h2{font-family:monospace;font-size:14px}body,div,table,h2{marg
10 <link rel="stylesheet" href="t32winprint.css" type="text/css"/>
11 </head>
12 <body>
13 <div class="t32 t32.register.view">
14 <h2>B::Register.view</h2>
15 <table>
16 <tr><td>N    R0          0  R8          0  ^S+ ^Stack +</td></tr>
17 <tr><td>7    D1          0  D9          0  /</td></tr>
```

- 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.
- B Basic formatting provided by TRACE32.
- C 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.

```
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
                                         ;DIR window.
                                         ;right-click to open a file
```

See also

■ [PRinTer](#)

■ [PRinTer.FILE](#)

■ [PRinTer.select](#)

■ [WinPrint](#)

■ [Var.EXPORT](#)

▲ ['Printer Operations' in 'PowerView User's Guide'](#)

▲ ['Release Information' in 'Legacy Release History'](#)

Format: **PRinTer.FILE** [*<file>*] [*<format>*] [**/Append**]

<format>: **ASCII | ASCIIP | ASCIIE | CSV | PSxxx | PCLxxx |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.

<i><file></i>	In order to simplify multiple file generation, a decimal number contained in the file name (e.g. exam00.1st) is incremented automatically. If <i><file></i> is omitted, the printer output is redirected to the previously chosen output file name (incremented if the file name contained a decimal number).
<i><format></i>	If <i><format></i> is omitted, the format used to print to file stays unchanged.
Append	Use the option Append to append new data to the existing file. Without Append , file contents are overwritten if the file already exists.

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.
ASCIIE	Enhanced ASCII file, underlines are displayed, graphic characters are displayed as ASCII characters.
CSV	Comma-separated value.
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 .
PCLPxx	Printer Command Language output format. Different resolutions and orientations are available.
XHTML XML (deprecated)	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 .

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

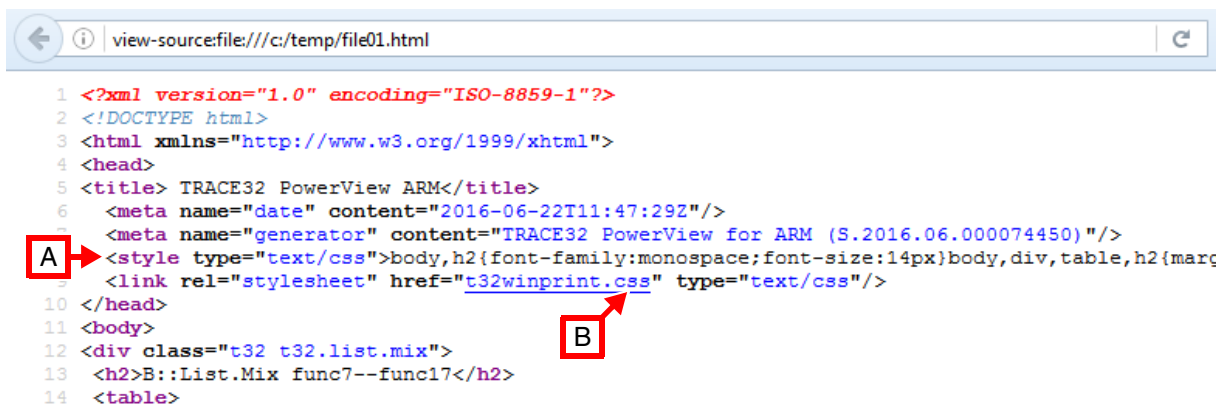
List.Mix windows are printed to separate files in HTML format.

```
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"
```



```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <!DOCTYPE html>
3 <html xmlns="http://www.w3.org/1999/xhtml">
4 <head>
5 <title> TRACE32 PowerView ARM</title>
6 <meta name="date" content="2016-06-22T11:47:29Z"/>
7 <meta name="generator" content="TRACE32 PowerView for ARM (S.2016.06.000074450)"/>
8 <style type="text/css">body,h2{font-family:monospace;font-size:14px}body,div,table,h2{marc
9 <link rel="stylesheet" href="t32winprint.css" type="text/css"/>
10 </head>
11 <body>
12 <div class="t32 t32.list.mix">
13 <h2>B::List.Mix func7--func17</h2>
14 <table>
```

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.

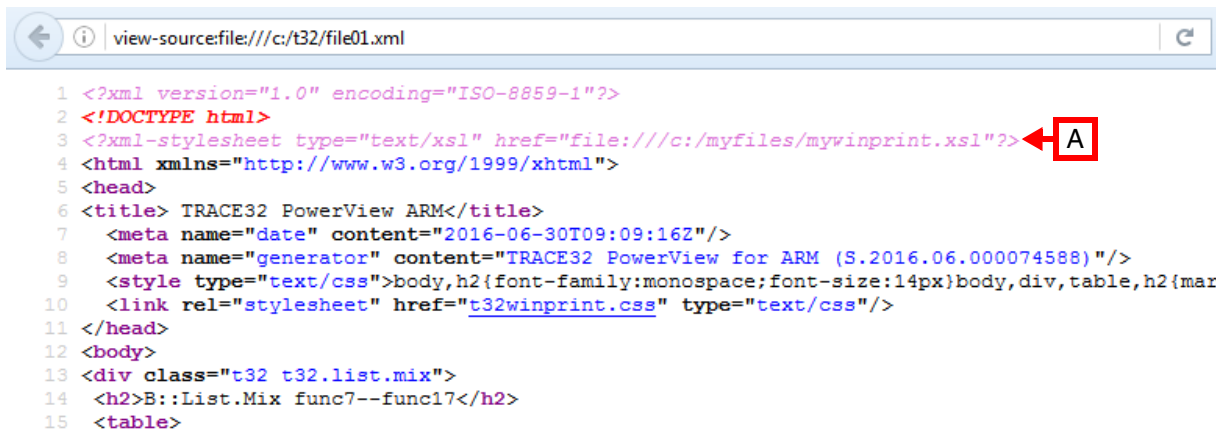
```
;in the *.xml file, insert the tag <?xml-stylesheet ...href="..."?>
;with the specified *.xsl file as href="..." attribute
SETUP.XSLTSTYLESHEET "file:///c:/myfiles/mywinprint.xsl"

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"
```



```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <!DOCTYPE html>
3 <?xml-stylesheet type="text/xsl" href="file:///c:/myfiles/mywinprint.xsl"?>
4 <html xmlns="http://www.w3.org/1999/xhtml">
5 <head>
6 <title> TRACE32 PowerView ARM</title>
7 <meta name="date" content="2016-06-30T09:09:16Z"/>
8 <meta name="generator" content="TRACE32 PowerView for ARM (S.2016.06.000074588)"/>
9 <style type="text/css">body,h2{font-family:monospace;font-size:14px}body,div,table,h2{mar
10 <link rel="stylesheet" href="t32winprint.css" type="text/css"/>
11 </head>
12 <body>
13 <div class="t32 t32.list.mix">
14 <h2>B::List.Mix func7--func17</h2>
15 <table>
```

- 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.

POSTSCRIPT

The style of POSTSCRIPT outputs can be widely varied by modifying the prolog file 't32pro.ps'. This file is prepended to all POSTSCRIPT outputs sent 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 be used as encapsulated postscript files to include them in documentations produced by desktop publishing software.

For information on POSTSCRIPT:

- Adobe Systems Inc.
Postscript Language Reference Manual, Second Edition
Addison Wesley 1991,
ISBN 0-201-18127-4
- Adobe Systems Inc.
Postscript Language Tutorial and Cookbook,
Addison Wesley 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.EXPORT](#)
- [PRinTer.OPEN](#)
- [PRinTer.select](#)
- [SETUP.XSLTSTYLESHEET](#)
- [WinPrint](#)
- ▲ ['Printer Operations' in 'PowerView User's Guide'](#)
- ▲ ['Window System' in 'PowerView User's Guide'](#)
- ▲ ['Release Information' in 'Legacy 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.EXPORT](#) instead.

See also

- [PRinTer](#)
- [PRinTer.select](#)
- ▲ ['Release Information' in 'Legacy Release History'](#)

Format:	PRinTer.HardCopy
---------	-------------------------

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

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

```
;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 'PowerView User's Guide'](#)
- ▲ ['Window System' in 'PowerView User's Guide'](#)

PRinTer.OFFSET

Specify print-out borders

Deprecated. See [PRinTer.CONFIG.OFFSET](#)

See also

- [PRinTer](#)
- [PRinTer.select](#)
- [WinPrint](#)
- ▲ ['Printer Operations' in 'PowerView User's Guide'](#)

Format: **PRinTer.OPEN** [*<file>*] [*<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.

<file> If *<file>* is omitted, the default file name t32.lst is used.
If the specified file already exists, it will be overwritten by default.

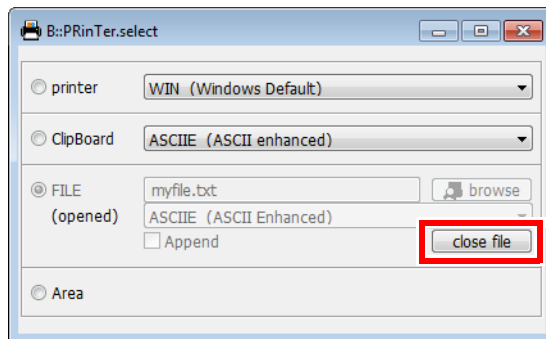
<format> If *<format>* is omitted, the format used to print to file stays unchanged.
For a list of available file formats, see command **PRinTer.FILE**.

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.

```
PRinTer.select
PRinTer.OPEN "~~~/myfile.txt" ASCIIIE ;create and open a file for
                                         ;writing in ASCIIIE 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.

```
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" ASCIIIE ;create and open a file for
                                         ;writing in ASCIIIE 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](#)
- ▲ 'Window System' in 'PowerView User's Guide'
- ▲ 'Printer Operations' in 'PowerView User's Guide'

Format: **PRinTer.PRINT** [*<format>*] *<data>*

<format>: **ASCII | BINary | Decimal | Hex | String**

Writes the specified data to the file selected with **PRinTer.OPEN**. Use the **PRinTer.PRINT** command to store additional information to the printed windows.

Example: A timestamp is printed at the beginning of the file, and then two windows are printed to file.

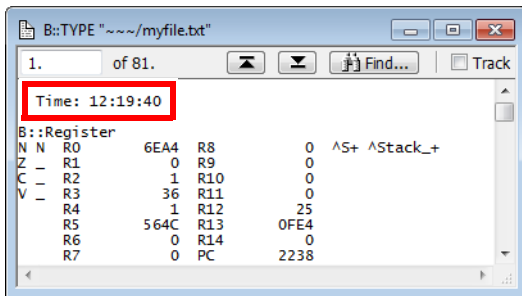
```
PRinTer.OPEN "~~~/myfile.txt" ASCIIIE ;create and open a file for
                                         ;writing in ASCIIIE file format

PRinTer.PRINT " "
PRinTer.PRINT " Time: "+DATE.TIME() ;print timestamp 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](#)

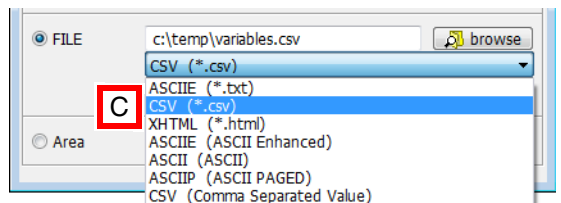
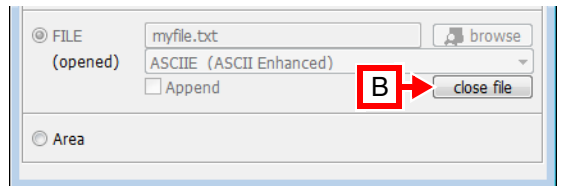
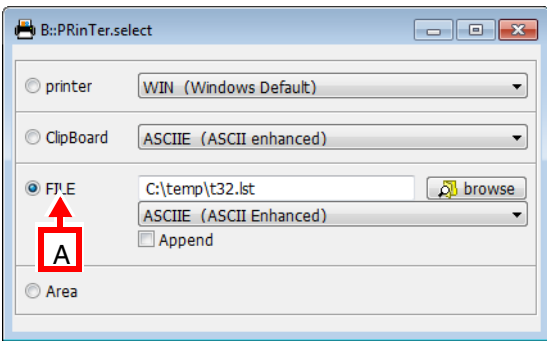
■ [PRinTer.select](#)

```
Format: PRinTer.select [<printer>]
```

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 re-directed 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.Area | ■ PRinTer.ClipBoard | ■ PRinTer.CLOSE |
| ■ PRinTer.CONFIG | ■ PRinTer.EXPORT | ■ PRinTer.FILE | ■ PRinTer.FileType |
| ■ PRinTer.HardCopy | ■ PRinTer.OFFSET | ■ PRinTer.OPEN | ■ PRinTer.PRINT |
| ■ PRinTer.SIZE | | | |

- ▲ 'Window System' in 'PowerView User's Guide'
- ▲ 'Printer Operations' in 'PowerView User's Guide'

Deprecated. See [PRinTer.CONFIG.SIZE](#)

See also

- [PRinTer](#)
- [PRinTer.select](#)
- ▲ 'Printer Operations' in 'PowerView User's Guide'

Format: **PWD** [*<path>*]

If used *without* *<path>*, **PWD** displays the current working directory in the TRACE32 [message line](#).

If used *with* *<path>*, **PWD** changes the working directory as specified in *<path>* and displays the new working directory in the TRACE32 message line.

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](#)
- [DIR](#)
- [OS.DIR\(\)](#)
- [OS.PresentWorkingDirectory\(\)](#)
- ▲ ['Commands' in 'PowerView User's Guide'](#)
- ▲ ['File and Folder Operations' in 'PowerView User's Guide'](#)

PYthon

The PYthon command group allows you to execute Python code directly from TRACE32 PowerView GUI. If you have multiple Python interpreter installed [SETUP.PYthon.EXEcutable](#) can be used to configure which one is used.

Python scripts can be executed using the [PYthon.RUN](#) command. Your python script can then control the debugger using the Python RCL Module.

Please refer to `~/demo/api/python/rcl/doc/html/index.html` on how to install and use the Python RCL module. The source for the Python RCL module can be found in `~/demo/api/python/rcl/dist`. You can either install it manually or use [PYthon.INSTALL](#) command.

You can also execute your Python scripts directly, for instance from your Python IDE or from command line and control the debugger from outside of PowerView.

PYthon.EDIT

Open Python script in editor

Format: **PYthon.EDIT** *<filename>*

Opens given script in Python editor window.

PYthon.INSTALL

Install RCL module and Python interpreter

Format: **PYthon.INSTALL** *<module_name>*

Installs the RCL module and the Python interpreter configured by [SETUP.PYthon.EXEcutable](#). While installing script execution is blocked. The result of the installation process is written into the area window.

Example:

```
PYthon.INSTALL RCL
```


Format: **PYthon.RUN** *<scriptname>* *<par1>* *<par2>*

Executes Python script with given parameter. Output and input of the script are redirected to dedicated window. Python version (min. 3.6) and the presence of the of the Python RCL module will be verified on the first run. Path to the Python interpreter can be defined using [SETUP.PYthon.EXEcutable](#).

PYthon.RUN opens dedicated terminal window for input and output of the Python script or reuses existing one if already open. Please check in case of troubles both the Python terminal and [AREA](#) window for error, warning and info messages.

Format: **QUIT** [*<os_return>*]

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:

```
./t32marm  
echo $?
```

Example for Windows to use the *<os_return>* value in a batch file:

```
start "" /wait t32marm.exe  
echo %ERRORLEVEL%
```

See also

■ [SETUP.QUITDO](#)

▲ ['Program End' in 'PowerView User's Guide'](#)

REN

REN

Rename file

Format: **REN** *<oldname>* *<newname>*

Renames a file.

<oldname>,
<newname>

Wildcard characters are **not** supported.

See also

■ [MV](#)

▲ ['File and Folder Operations'](#) in ['PowerView User's Guide'](#)

Format: **RM** *<file>*

Removes a file.

<file>

Wildcard characters within the file name will open the browser for selecting a file.

Example:

```
RM c:\t32\test.bak
```

See also

■ [DEL](#)

■ [MKTEMP](#)

▲ ['File and Folder Operations'](#) in ['PowerView User's Guide'](#)

Format: **RMDIR** *<path>*

Removes a sub-directory. The directory must be empty.

See also

■ [MKDIR](#)

■ [MKTEMP](#)

▲ ['File and Folder Operations'](#) in ['PowerView User's Guide'](#)

Format: **SScreenShot** [*<file>* [*<imageformat>* [*<window_name>* | **/ACTIVE**]]]

<image format>: **BMP | TIFF | PNG | JPG | GIF**

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.

NOTE: This command is only available if running PowerView on

- Microsoft Windows
- Linux with Qt-Screendriver
- macOS

Depending on the used OS version, some image formats may not be available.

<i><imageformat></i>	<ul style="list-style-type: none"> • BMP: Windows Bitmap format (default, lossless, uncompressed). • PNG: Portable Network Graphics format (lossless, compressed). • JPG: JPEG File Interchange format (lossy, compressed). • TIFF: Tagged Image File Format (lossless, uncompressed). Taking a screenshot in TIFF format is only supported on Microsoft Windows. • GIF: Graphics Interchange Format (lossy, compressed, 256 colors). Taking a screenshot in GIF format is only supported on Microsoft Windows. 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.
<i><window_name></i>	<ul style="list-style-type: none"> • 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	<p>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.</p>

Example 1:

```
; Capture a screenshot of the main window and save in JPEG format:  
ScreenShot "~~~/screenshot.jpeg" JPG
```

Example 2:

```
; Capture a screenshot of window named W001 and save in GIF format:  
WinPOS , , , , , W001  
SYStem.state  
WAIT 200.ms  
ScreenShot "~~~/screenshot.gif" GIF W001
```

Example 3:

```
; Open the SYStem.CONFIG window and capture a screenshot of it:  
WinPOS , , , , , myWin  
SYStem.CONFIG  
ScreenShot "~~~/screenshot.png" PNG myWin
```

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.COLOR](#)
- [SETUP.DropCoMmanD](#)
- [SETUP.EDITOR](#)
- [SETUP.FASTRESPONSE](#)
- [SETUP.HOLDDIR](#)
- [SETUP.InterComACKTIMEOUT](#)
- [SETUP.PDFViewer](#)
- [SETUP.RADIX](#)
- [SETUP.ReDraw](#)
- [SETUP.SOUND](#)
- [SETUP.STOre](#)
- [SETUP.TIMEFORM](#)
- [SETUP.WARNSTOP](#)
- [SETUP.BAKfile](#)
- [SETUP.DEVNAME](#)
- [SETUP.EDITEXT](#)
- [SETUP.EXTension](#)
- [SETUP.FILETYPE](#)
- [SETUP.ICONS](#)
- [SETUP.PDEBUG](#)
- [SETUP.QUITDO](#)
- [SETUP.RANDOM](#)
- [SETUP.RESOLVEDIR](#)
- [SETUP.STOPMESSAGE](#)
- [SETUP.TabSize](#)
- [SETUP.UpdateRATE](#)
- [SETUP.XSLTSTYLESHEET](#)

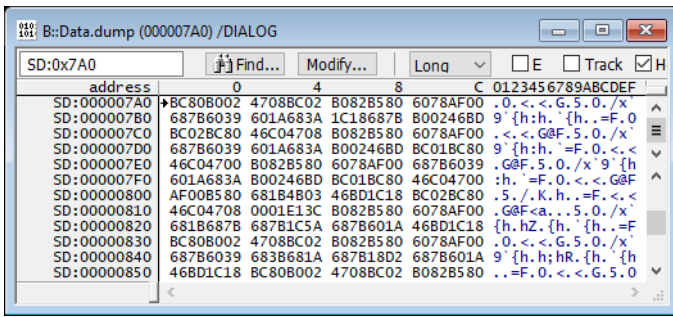
▲ ['SETUP' in 'General Commands Reference Guide S'](#)

Format:	SETUP.ASCIITEXT [<i><mode1></i>] [<i><mode2></i>]
<i><mode1></i> :	FULL8 FULL7 PART8 PART7 UTF-8
<i><mode2></i> :	SPACE BLANK SWAP

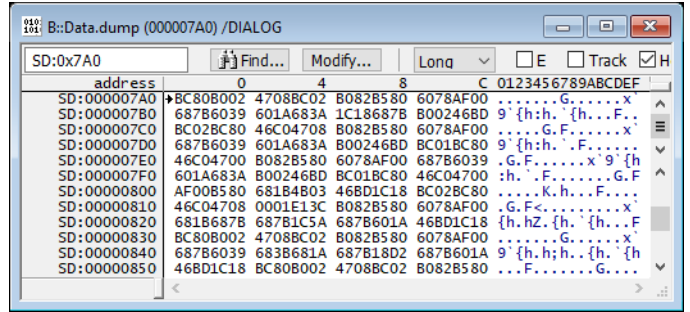
Configures the display mode for all non-standard characters in dump windows.

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.

SPACE	Display space character as '_' in ascii dump window. (Default)
BLANK	Display space character as BLANK (0x20) and not as '_' in ascii dump window.
SWAP	Display with reverse character-order of each two adjacent characters.



B::SETUP.ASCIITEXT PART7



B::SETUP.ASCIITEXT PART8

See also

- [SETUP](#) ■ [Data.dump](#)
- ▲ 'System Setup and Configuration' in 'PowerView User's Guide'
- ▲ 'Release Information' in 'Legacy Release History'

Format: **SETUP.BAKfile** [ON | OFF]

If ON (default) and the debugger is about to overwrite an existing file (due to e.g. Data.SAVE, Trace.SAVE), the debugger will make a backup of the existing file by renaming the file extension.

NOTE:

The default file extension of the backup file is `.BAK`.
The backup file extension can be changed using **SETUP.EXT BAK**.

This command does not affect the backup file creation for files opened in the editor. See **SETUP.EDITOR BAKfile** on how to control backup file creation for files opened in the editor.

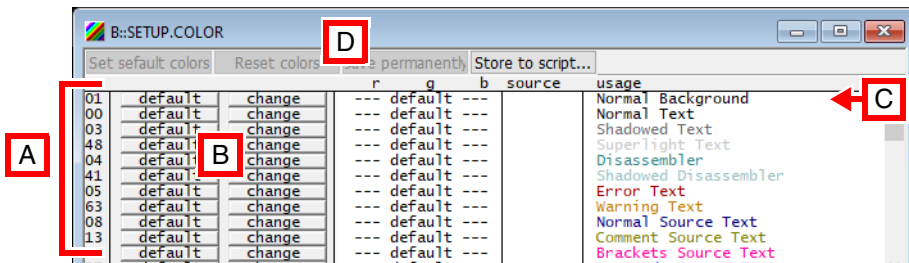
See also

■ [SETUP](#)

▲ ['System Setup and Configuration'](#) in ['PowerView User's Guide'](#)

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 in multicore debug sessions. For example, if green stands for core 1, then information from core 1 will be displayed in windows with a green window background.
- D** Button functions
 - Set default colors: Set all colors to default values
 - Reset colors: Reset colors to the colors currently saved in config.t32 or preferences
 - Save permanently: Save colors to user preferences
 - Store to script: Generates a script to reproduce the currently selected color settings

For all host operating systems, the color depth for each color channel is 8 bit. Values go from 0(darkest) to 255 (lightest color).

```
SETUP.COLOR 40. 0xff 0x14 0x93           ; Change the Info Message
                                           ; Background (40.) to DeepPink
```

Backwards compatibility mode for CDE/Motiv: In addition to the 8 bit depth per color channel, when the CDE/Motif screen driver is used, SETUP.COLOR also supports 16 bits per color channel. Values exceeding 0xFF are automatically scaled down. The compatibility mode exists to for PowerView versions before build 136645. Usage is not recommended for new installations. E.g. DeepPink (FF 14 93) results as below command:

```
;CDE/Motiv only:                                ; Change the Info Message
SETUP.COLOR 40. 0xff00 0x1400 0x9300           ; Background (40.) to DeepPink
;any host OS and screen driver:              ; (RGB values: FF 14 93)
SETUP.COLOR 40. 0xff 0x14 0x93
```

For Unix derivatives the X11 color values, see http://en.wikipedia.org/wiki/web_colors

See example script `~/demo/practice/colors/presentation.cmm`

See also

- [SETUP](#)
- [CmdPOS](#)
- [FramePOS](#)
- [CORE.SHOWACTIVE](#)
- [sYmbol.ColorDef](#)
- [sYmbol.List.ColorDef](#)
- ▲ ['PowerView - Screen Display'](#) in ['PowerView User's Guide'](#)
- ▲ ['System Setup and Configuration'](#) in ['PowerView User's Guide'](#)

SETUP.DEVNAME

Set logical device name

Format: **SETUP.DEVNAME** [*<sysname>*]

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.

Example:

```
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 ['PowerView User's Guide'](#)

Format:

SETUP.DropCoMmanD *<type>* [*<command>*] (deprecated)

Use [SETUP.FILETYPE.DropCoMmanD](#) instead.

See also

■ [SETUP](#)

■ [DO](#)

■ [MENU.Program](#)

■ [Data.LOAD.Eif](#)

■ [Data.LOAD.eXe](#)

■ [Data.PROGRAM](#)

■ [PER](#)

▲ ['Release Information' in 'Legacy Release History'](#)

Format: **SETUP.EDITTEXT ON | OFF** [*<cmdline>*]

Replaces the TRACE32 built-in editor call with an external editor call.

ON	The EDIT.file command starts your external editor - instead of the TRACE32 editor. See example 1 .
OFF	The external editor is only started when you execute the EDIT.EXtern command. See example 2 .
<i><cmdline></i>	<p>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:</p> <ul style="list-style-type: none"> • * will be replaced by the actual file name. • # will be replaced by the actual line number.

Example 1: This script shows how to configure TextPad (or JEDIT or UltraEdit) as an external editor for TRACE32 PowerView with the **ON** setting:

```

;configure TextPad as an external editor
SETUP.EDITTEXT ON "C:\Program Files\TextPad 5\TextPad.exe "*" (#)""

;configure JEDIT as an external editor
;SETUP.EDITTEXT ON "C:\eclipse\jedit5.0.0\jedit.exe "*" +line:#"

;configure UltraEdit as an external editor
;SETUP.EDITTEXT ON "C:\IDM\UltraEdit\uedit32.exe """"

;PRACTICE script file opens in the external editor
EDIT.file ~~/my-script.cmm
    
```

Example 2: This script 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 TRACE32 editor as usual
EDIT.file ~~~/mylog.txt

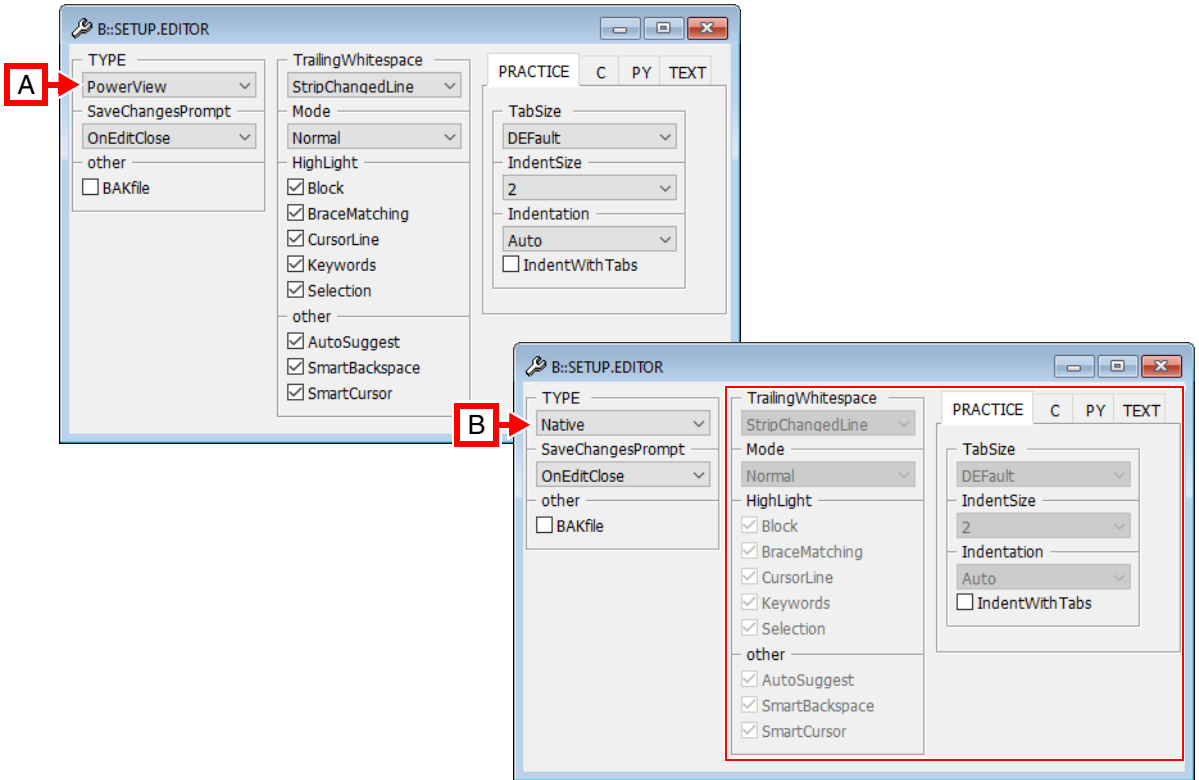
;Text file now opens in the external editor
EDIT.EXtern ~~~/mylog.txt
```

See also

- [SETUPEDITOR](#)
 - [EDIT.EXtern](#)
 - [SETUP](#)
 - [EDIT.file](#)
 - [DIR](#)
 - [EDIT.OPEN](#)
 - [EDIT](#)
- ▲ ['Text Editors'](#) in ['PowerView User's Guide'](#)
 - ▲ ['System Setup and Configuration'](#) in ['PowerView User's Guide'](#)
 - ▲ ['File and Folder Operations'](#) in ['PowerView User's Guide'](#)
 - ▲ ['Release Information'](#) in ['Legacy Release History'](#)

The **SETUP.EDITOR** command group allows you configure the format settings for and the behavior of the TRACE32 editors.

To make your configuration settings, use the TRACE32 command line, a PRACTICE script (*.cmm), or the **SETUP.EDITOR.state** dialog window:



A All settings are only available if the **TYPE** *<editor_feature_set>* is set to **PowerView**.

B When you change the **TYPE** setting from **PowerView** to **Native**, most of the settings are deactivated. For more information, see **SETUP.EDITOR.TYPE** *<editor_feature_set>*.

For a description of the commands on the dialog window, see **SETUP.EDITOR.state**.

See also

- SETUP.EDITOR.AutoSuggest
- SETUP.EDITOR.HighLight
- SETUP.EDITOR.IndentSize
- SETUP.EDITOR.Mode
- SETUP.EDITOR.SmartBackspace
- SETUP.EDITOR.SmartFormat
- SETUP.EDITOR.TabSize
- SETUP.EDITOR.TYPE
- SETUP
- EDIT
- EDIT.FORMAT
- MENU.Program
- SETUP.EDITOR.BAKfile
- SETUP.EDITOR.Indentation
- SETUP.EDITOR.IndentWithTabs
- SETUP.EDITOR.SaveChangesPrompt
- SETUP.EDITOR.SmartCursor
- SETUP.EDITOR.state
- SETUP.EDITOR.TrailingWhitespace
- SETUP.EDITEXT
- DIALOG.Program
- EDIT.file
- EDIT.OPEN
- PEDIT

SETUP.EDITOR.AutoSuggest

Show input suggestions while typing

This command is only available if **SETUP.EDITOR.TYPE** is set to **PowerView**.

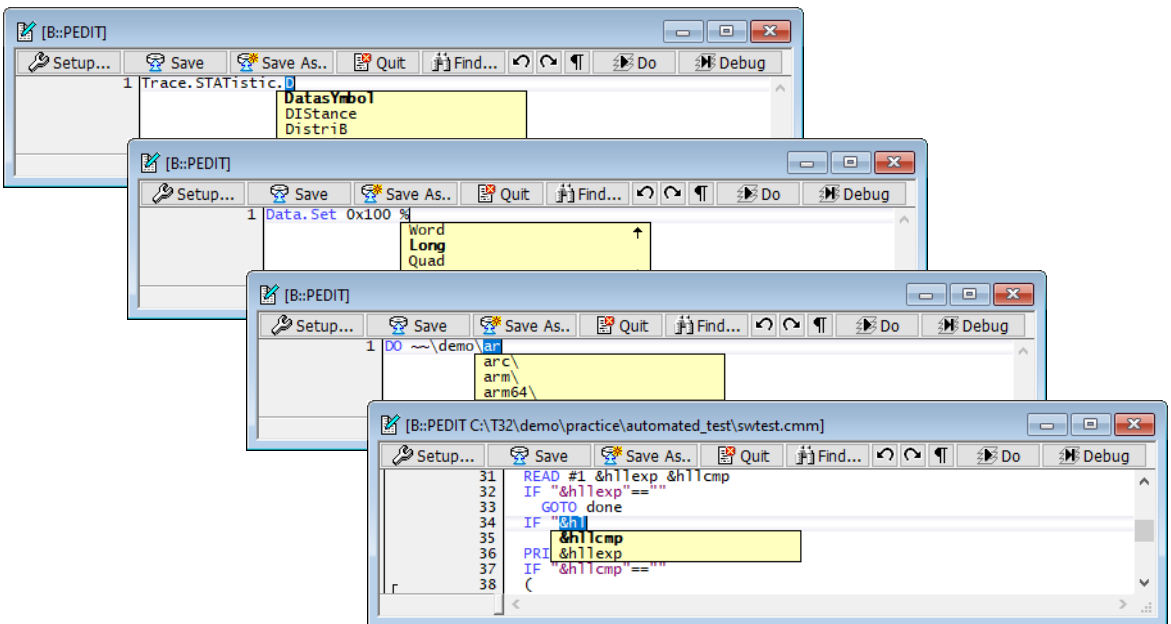
Format:	SETUP.EDITOR.AutoSuggest [ON OFF <mode>]
<mode>	DefaultSelection [ON OFF] SpacebarInsertion [ON OFF]

Defines when the PowerView editor shows input suggestions or auto completion candidates.

OFF Show only in explicit user request (CTRL+SPACEBAR)

ON Show every time a user inputs text.

Input suggestions / auto completion are currently supported for the PRACTICE editor **PEDIT** and supports commands, command parameters, keywords, file paths and PRACTICE macros.



Enhanced modes are available for more efficient (though also more intrusive) use of the auto suggestion feature:

DefaultSelection If OFF (default), the auto completion box appears without a preselection and the best matching item is only selected after (CTRL+SPACEBAR) is pressed. If ON, the auto completion box will always appear with an item selected (implicit CTRL+SPACEBAR after every input)

SpacebarInsertion If OFF, insertion is triggered by RETURN, and on any character which is not a letter or number (exceptions exist depending on type of value, e.g. debug symbol, path etc.). If ON, SPACEBAR will trigger insertion of the selected item in addition to the inputs that trigger insertion when this option is OFF.

See also

■ [SETUP.EDITOR](#)

■ [SETUP.EDITOR.state](#)

SETUP.EDITOR.BAKfile

Make backup copy when file is saved

Format: **SETUP.EDITOR.BAKfile [ON | OFF]**

Defines if the TRACE32 editor should make a backup copy if an existing file before saving the new file.

Note: In order to configure if other commands (e.g. Data.SAVE, Trace.SAVE) should make backups before overwriting existing files, see [SETUP.BAKfile](#).

See also

■ [SETUP.EDITOR](#)

■ [SETUP.EDITOR.state](#)

SETUP.EDITOR.HighLight

Control syntax highlighting

This command is only available if [SETUP.EDITOR.TYPE](#) is set to **PowerView**.

Format: **SETUP.EDITOR.HighLight <item> ON | OFF**

<item>: **Block | BraceMatching | CursorLine | Keywords | Selection**

Default: ON for each <item>.

Switches the syntax highlighting for the selected *<item>* **ON** or **OFF**.

Block	Lines with matching block delimiters are highlighted in yellow as soon as the insertion point is located in either line.
BraceMatching	When you place the insertion point to the left of a brace, the matching brace in the same line is highlighted. Example: <code>IF (&sel&((0x1)))==0x0</code>
CursorLine	Highlights the entire line where you have placed the insertion point.
Keywords	Highlights commands, functions, strings, comments, etc.
Selection	In addition to the current selection, all other occurrences matching the current selection are highlighted in yellow.

See also

■ [SETUP.EDITOR](#)

■ [SETUP.EDITOR.state](#)

SETUP.EDITOR.Indentation

Select indentation method

This command is only available if [SETUP.EDITOR.TYPE](#) is set to **PowerView**.

Format:	SETUP.EDITOR.Indentation [<i><filetype></i>] <i><mode></i>
<i><filetype></i> :	all PRACTICE C PYTHON ASM TRIG TEXT
<i><mode></i> :	OFF Keep Auto

This command defines how the PowerView editor indents new lines when the user presses the RETURN key.

Settings can be applied to either all file types together or for several file types individually. The table below lists all available file types that support individual settings:

all	Settings are valid to all available file types and overwrite individual settings.
PRACTICE	Settings are valid for the PRACTICE editor PEDIT , and for the TRACE32 peripheral view and menu program editors PER.Program and MENU.Program
C	Settings are valid for the text editor in the case a C / C++ source file is opened in the EDIT window.

PYTHON	Settings are valid for the Python editor PY.EDIT .
ASM	Settings are valid for the TRACE32 assembler editor Data.Program
TRIG	Settings are valid for the TRACE32 trigger program editors: Integrator.Program , Probe.Program , Break.Program
TEXT	Settings are valid for all other file types opened in EDIT windows.

Supported indentation modes:

OFF	New lines begin at the first character, no whitespace is inserted.
Keep	New lines are inserted exactly like the previous line.
Auto	Context-sensitive indentation and de-indentation. Supported for PEDIT, PY.EDIT, MENU.Program, PER.Program

See also

- [SETUP.EDITOR](#)
- [SETUP.EDITOR.state](#)

SETUP.EDITOR.IndentSize

Set indentation size

This command is only available if [SETUP.EDITOR.TYPE](#) is set to **PowerView**.

Format:	SETUP.EDITOR.IndentSize <i><filetype></i> <i><size></i>
<i><filetype></i> :	all PRACTICE C PYTHON ASM TRIG TEXT (see SETUP.EDITOR file type table)
<i><size></i> :	DEfault 1..8

This command defines the indentation size used by the PowerView editor.

DEfault	Use value set by SETUP.TabSize . Default of SETUP.TabSize is 8.
1..8	The indentation size is set according to the specified value.

See also

- [SETUP.EDITOR](#)
- [SETUP.EDITOR.state](#)
- [SETUP.TabSize](#)

This command is only available if [SETUP.EDITOR.TYPE](#) is set to **PowerView**.

Format:	SETUP.EDITOR.IndentWithTabs <i><filetype></i> ON OFF
<i><filetype></i> :	all PRACTICE C PYTHON ASM TRIG TEXT (see SETUP.EDITOR file type table)

This command defines if the PowerView editor uses tabs for indentation.

ON	The editor uses tabs or mixed tab/blank for indentation (depending on SETUP.EDITOR.TabSize and SETUP.EDITOR.IndentSize)
OFF	The editor uses only blanks for indentation according to SETUP.EDITOR.IndentSize

See also

- [SETUP.EDITOR](#)
- [SETUP.EDITOR.state](#)

This command is only available if [SETUP.EDITOR.TYPE](#) is set to **PowerView**.

Format:	SETUP.EDITOR.Mode Normal VisibleSpaces ASCII
---------	--

This command defines how the PowerView editor shows the text document.

Normal	The editor shows any whitespace as blank screen
VisibleSpaces	The editor shows visible blanks and tabs.
ASCII	The editor shows all non-printable characters (like ASCII column of e.g. the Data.dump window).

See also

- [SETUP.EDITOR](#)
- [SETUP.EDITOR.state](#)

Format: **SETUP.EDITOR.SaveChangesPrompt [OnT32Quit | OnEditClose]**

Defines when PowerView asks if a file that is opened and modified in the TRACE32 editor should be saved.

OnT32Quit	PowerView asks if the file should be saved when TRACE32 PowerView is quit.
OnEditClose	PowerView asks if the file should be saved when the edit window is closed.

See also

■ [SETUP.EDITOR](#)

■ [SETUP.EDITOR.state](#)

This command is only available if [SETUP.EDITOR.TYPE](#) is set to **PowerView**.

Format:	SETUP.EDITOR.SmartBackspace ON OFF
---------	---

Default: OFF.

Controls the effect of the Backspace key if the cursor is at the first non-whitespace character of the current line.

OFF	Backspace deletes a single character left of the cursor.
ON	If the cursor is at the first non-whitespace character of a line, the backspace key will remove one indentation depth of whitespace. Useful for blank-only indentation schemes like commonly used for Python.

See also

- [SETUP.EDITOR](#)
- [SETUP.EDITOR.state](#)

This command is only available if [SETUP.EDITOR.TYPE](#) is set to **PowerView**.

Format:	SETUP.EDITOR.SmartCursor ON OFF
---------	--

Default: OFF.

Controls how the editor sets the cursor column during vertical cursor movement (cursor up/down).

OFF	The new cursor column is the same column as before vertical cursor movement (standard behavior of most editors).
ON	The new cursor column is the column the cursor was located before text was inserted or removed before vertical cursor movement (TextPad(R)-like behavior).

See also

- [SETUP.EDITOR](#)
- [SETUP.EDITOR.state](#)

This command is only available if [SETUP.EDITOR.TYPE](#) is set to **PowerView**.

Format: **SETUP.EDITOR.SmartFormat ON | OFF**

If this setting is ON (default), automatic formatting will occur on the following occasions:

- Paste event: The pasted block will be formatted so that it matches the preceding indentation.
- Block close input event: When a block closing character is entered, the closed block will be formatted according to preceding indentation.

The automatic formatting can be undone by pressing CTRL-Z once, while pressing CTRL-Z twice will unsi the original action as well.

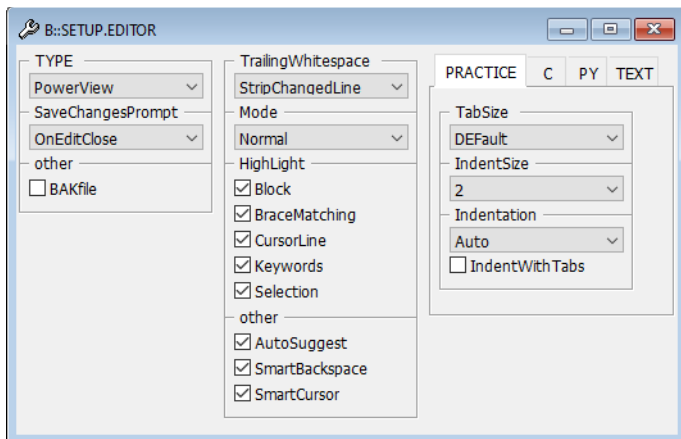
See also

■ [SETUP.EDITOR](#)

■ [SETUP.EDITOR.state](#)

Format: **SETUP.EDITOR.state**

Opens the configuration dialog window for the [TRACE32 editors](#).



TRACE32 supports two editors, the fully integrated PowerView editor and an OS / UI framework native editor as fallback. Most configuration options are only available for the PowerView editor. The desired editor can be set using the command **SETUP.EDITOR.TYPE**.

NOTE: Use the **ClipSTOre** or **STOre** command to obtain the current **SETUP.EDITOR.state** settings in the form of a PRACTICE script.

By copying and pasting the resulting script into your user-settings.cmm or system-settings.cmm, you can re-use your preferred settings in future TRACE32 sessions.

For more information about the files user-settings.cmm and system-settings.cmm, refer to “**Automatic Start-up Scripts**” in PRACTICE Script Language User’s Guide, page 15 (practice_user.pdf).

See also

- SETUP.EDITOR
- SETUP.EDITOR.BAKfile
- SETUP.EDITOR.Indentation
- SETUP.EDITOR.IndentWithTabs
- SETUP.EDITOR.SaveChangesPrompt
- SETUP.EDITOR.SmartCursor
- SETUP.EDITOR.TabSize
- SETUP.EDITOR.TYPE
- SETUP.EDITOR.AutoSuggest
- SETUP.EDITOR.HighLight
- SETUP.EDITOR.IndentSize
- SETUP.EDITOR.Mode
- SETUP.EDITOR.SmartBackspace
- SETUP.EDITOR.SmartFormat
- SETUP.EDITOR.TrailingWhitespace

This command is only available if [SETUP.EDITOR.TYPE](#) is set to **PowerView**.

Format:	SETUP.EDITOR.TabSize <i><filetype></i> <i><size></i>
<i><filetype></i> :	all PRACTICE C PYTHON ASM TRIG TEXT (see SETUP.EDITOR file type table)
<i><size></i> :	DEFault 1..8

This command defines the tabulator size used by the PowerView editor.

DEFault	Use value set by SETUP.TabSize . Default of SETUP.TabSize is 8.
1..8	The tabulator size is set according to the specified value.

See also

- [SETUP.EDITOR](#)
- [SETUP.EDITOR.state](#)

SETUP.EDITOR.TrailingWhitespace

Remove trailing whitespace

This command is only available if [SETUP.EDITOR.TYPE](#) is set to **PowerView**.

Format:	SETUP.EDITOR.TrailingWhitespace Keep Strip StripChangedLine
---------	---

This command defined if and how the PowerView editor will remove trailing whitespace. If enabled, trailing whitespace is removed every time a file is saved.

Keep	The editor does not remove trailing whitespace.
Strip	The editor removes any trailing whitespace.
StripChangedLine	The editor removes trailing whitespace in every line that has been modified since the file was loaded in the editor.

See also

- [SETUP.EDITOR](#)
- [SETUP.EDITOR.state](#)

Format: **SETUP.EDITOR.TYPE** *<type>*

<type>: **PowerView | Native**

This command defines which editor implementation is used. Default: PowerView.

PowerView	Fully integrated editor with <ul style="list-style-type: none">- true syntax highlighting for all TRACE32 programming languages- context sensitive help- automatic suggestions and completion of commands, keywords, file paths, PRACTICE macros and debug symbols- automatic indentation and command formatting
Native	Standard edit container as provided by operating system or UI framework. Not recommended.

See also

■ [SETUP.EDITOR](#)

■ [SETUP.EDITOR.state](#)

Format: **SETUP.EXTension** <file_type> [<extension_def> [<extension_2>]]
(deprecated)
Use **SETUP.FILETYPE.EXTension** instead.

See also■ [SETUP](#)▲ ['System Setup and Configuration'](#) in ['PowerView User's Guide'](#)

SETUP.FASTRESPONSE

Optimize for fast response times

Format: **SETUP.FASTRESPONSE** [ON | OFF]

Configures TRACE32 PowerView for fast response times while streaming trace data. For [RTS](#) and [Analyzer.Mode STREAM](#)

By default (OFF), the debugger gives priority to streaming and processing the trace data to gain maximum transfer bandwidth. This is the recommended setting for most use cases.

If it is required to get a short reaction time on e.g. breakpoint hit notifications, only then set this option to ON.

See also■ [SETUP](#)

See also

- [SETUP.FILETYPE.ENCoding](#) ■ [SETUP.FILETYPE.EXTension](#) ■ [SETUP](#)

SETUP.FILETYPE.DropCoMmanD

Set command for dropped files

[build 148462 - DVD 09/2022]

Format:	SETUP.FILETYPE.DropCoMmanD <i><type></i> [<i><command></i>]
<i><type></i> :	OTHER ASM ELF EXE MENU PER PRACTICE

When you drag a file from an application other than TRACE32 and drop it into the TRACE32 command line, a default command is automatically prepended in the command line in order to open, execute, or load the file. The default command used depends on the file extension.

SETUP.FILETYPE.DropCoMmanD allows to change the *<command>* which is used when a file of a certain type is dropped into the TRACE32 command line.

<i><command></i>	<p>By default, TRACE32 automatically appends an asterisk to the parameter <i><command></i> if you omit the asterisk. This asterisk will be replaced with the name of the file you drop into the command line. Compare example 1 and 2.</p> <p>To restore the default command, omit the <i><command></i> and specify just the <i><type></i>. See example 3.</p>
ASM (*.asm)	Assembler programs (Default command: TYPE)
ELF (*.elf)	Files in the executable and linking format. (Default command: Data.LOAD.Elf)
EXE (*.exe)	Windows executables (Default command: Data.LOAD.eXe)
MENU (*.men)	Menu files (Default command: MENU.Program)
OTHER	A file with an extension unknown to TRACE32. (Default command: Data.LOAD)
PER (*.per)	Peripheral files (Default command: PER)
PRACTICE (*.cmm)	PRACTICE scripts (Default command: CD.DO)

Example 1: This script line changes the default command for PRACTICE scripts from **CD.DO** to **CD.RUN** when they are dropped into the command line:

```
SETUP.DropCoMmanD PRACTICE "ChDir.RUN"
```

Before:

```
B::SETUP.DropCoMmanD PRACTICE "CD.RUN"  
Drop-Command for PRACTICE script: "CD.DO *"
```

After:

```
B:::  
Drop-Command for PRACTICE script: "CD.RUN *"
```

Example 2: This script line changes the command for ELF files when they are dropped into the command line, so that they are loaded with the option **/CYGDRIVE** by default:

```
SETUP.DropCoMmanD ELF "Data.LOAD.Elf * /CYGDRIVE"
```

Example 3:

```
;change the default command "MENU.Program" to "MENU.ReProgram"  
SETUP.DropCoMmanD MENU "MENU.ReProgram"
```

```
;when you now drag&drop a menu file (*.men) into the TRACE32 command  
;line, the file name is prepended with "MENU.ReProgram"
```

```
;let's restore "MENU.Program" as the default command for drag&drop  
SETUP.DropCoMmanD MENU
```

SETUP.FILETYPE.ENCoding

Set encoding mode

[build 148217 - DVD 09/2022]

Format:	SETUP.FILETYPE.ENCoding <i><filetype></i> <i><encoding></i>
<i><filetype></i> :	SOURCE PRACTICE.script PRACTICE.ENCRYPTed
<i><encoding></i> :	AUTODETECT WINCP UTF-8 UTF-8-BOM

This command allows to define the file encoding for several text file types.

Default for Windows is AUTODETECT. For other operating systems, the default is UTF-8.

<encoding>	Description
AUTODETECT	Automatically detects if the file text is encoded in UTF-8 or the Windows code-page for non-unicode programs. This option is only available for Windows.
WINCP	Files are expected in the Windows code-page for non-unicode programs. This option is only available for Windows.
UTF-8	Files are expected to be in UTF-8 encoding. If PowerView saves or modifies the file, it will add the UTF-8 BOM at the beginning of the file, only if the original file also included a BOM.
UTF-8-BOM	Files are expected to be in UTF-8 encoding. If PowerView saves or modifies a file, it will add the UTF-8 BOM at the beginning of the file.

See also

■ [SETUPFILETYPE](#)

```

Format:          SETUP.FILETYPE.EXTension <file_type> [<extension_def> [<extension_2>]]

<file_type>:    AL | ALTERA | AP | ASM | BAK | BNK | BSDL | COV | CSV | DIALOG |
                ELF | LOG | LUA | MENU | ORTI | OS | PATPROG |
                PER | PERF.Data | PERF.program | PRACTICE | PRT |
                STORe | TAPROG | TEXT | XHTML
    
```

This command allows to change the default file type associations in TRACE32 PowerView. The file type association is done using the file extension (suffix).

<i><file_type></i>	The file type for which the extension is to be set. See table below for a list of supported file types, description and default extensions.
<i><extension_def></i>	Default extension for the specified tile type. Used as default extension for all LOAD, OPEN, SAVE or similar operations, drag&drop command association and file filters in file selection dialogs.
<i><extension_2></i>	Secondary extension. Only used for drag&drop command association and file filters in file selection dialogs.

If SETUP.EXTension is called with <file_type> only (no extensions), the currently associated extensions are displayed in the status line.

Each command in TRACE32 PowerView that loads or saves files has a file type associated to it. The file type is used for several actions:

- If a LOAD, OPEN, SAVE or similar command is entered with a file name that has no extension, TRACE32 PowerView will add the default extension automatically.
- If a file is moved to the PowerView command line using drag&drop, the command line will contain the dropped file, preceded by the command that is associated to either the default or second extension.
- If a LOAD, OPEN, SAVE or similar command is entered with wildcard(s), the file selection dialog will contain the associated extensions in the file filter box.

This table lists the file types and their extensions:

<file_type>	<extension_def> <extension_2>	Description
AL	.ad	Extension for the A.LOAD and A.SAVE commands.
ALTERA	.rbf	Extension for FPGA images in Raw Binary Format used by the JTAG.PROGRAM.A Altera command.
ASM	.asm	Extension for the Data.PROGRAM and Data.AssWin command.
BAK	.bak	Extension for all backup files. (See also command SETUP.BAKfile)
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.
ELF	.elf .axf	Extension for executable and linking format files.
LOG	.log	Extension for log-files crated e.g. via LOG.OPEN , LOG.DO , HISTory.SAVE , or SYStem.LOG.OPEN .
LUA	.lua	Extension for LUA scripts used by command LUA.LOAD
MENU	.men	Extension for TRACE32 menu description files used by MENU.Program .
ORTI	.orti .ort	Extension for the OSEK run-time interface used by TASK.ORTI .
OS	()	Extension for TYPE and EDIT commands.
PER	.per	Extension for all PER commands.
PERF.Data	.perf	Extension for Performance Analyzer Results used by the command PERF.SAVE and PERF.LOAD commands.
PERF.program	.ps	Extension for Performance Analyzer Programs used by the commands PERF.Program and PERF.ReProgram .
PRACTICE	.cmm	Extension for the DO , RUN and PEDIT commands.
PRACTICE. ENCRYPTION	.cmmx .cmm	Extension for the DODECRYPT command.
PRT	.lst	Extension for the PRinTer.OPEN and PRinTer.FILE commands.
STOre	.cmm	Extension for the STOre and AutoSTOre command.

<file_type>	<extension_def> <extension_2>	Description
TAPROG	.tap	Extension for the Probe.Program or Probe.ReProgram and Integrator.Program or Integrator.ReProgram commands.
TEXT	.txt	Extension for plain text files.
XHTML	.html .htm	Extension for files formatted in the extensible hypertext markup language.

See also

- [SETUPFILETYPE](#)

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 will be executed. Otherwise the working directory can be changed by the command **ChDir** only.

See also

■ [SETUP](#)

▲ ['System Setup and Configuration'](#) in 'PowerView User's Guide'

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](#)

▲ ['System Setup and Configuration'](#) in 'PowerView User's Guide'

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.

Example:

```
SETUP.InterComACKTIMEOUT 5.s ; Increase timeout to maximum
```

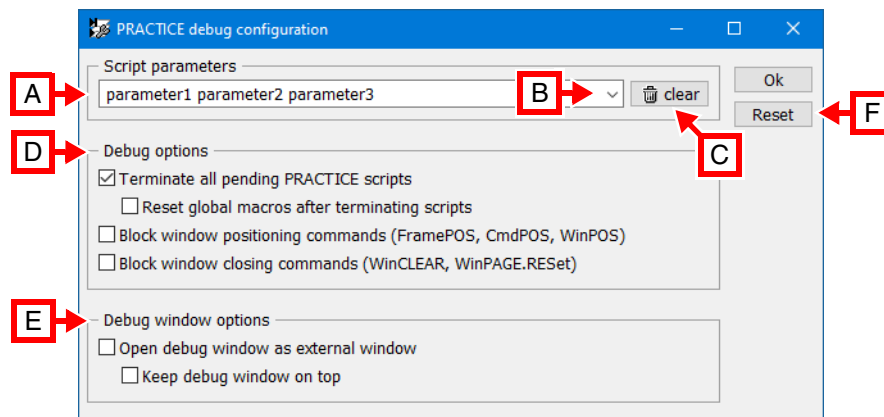
See also

■ [SETUP](#)

■ [InterCom](#)

Format: **SETUP.PDEBUG**
 SETUP.PDEBUG.state (as an alias)

Opens the PRACTICE debug configuration dialog to configure the script parameters and several preconditions a script is started with. The debug configuration settings will only be in effect while a script is started directly out of the PRACTICE editor PEDIT by using the debug toolbar button.



- A** Enter the parameters a script is started with.
 Enter the parameters the same way as the script would be started with a DO command. Press ENTER to finish the input or press the **Ok** button to finish input and instantly close the dialog. Alternatively set the parameters with the command **SETUP.PDEBUG.ScriptParams**.
- B** Show a dropdown list of previously used script parameters.
 All entered parameters are recorded in a history list which is reloaded at program start and stored at program termination (see **HISTORY.Set**)
- C** Use the **clear** button to delete the script parameters input field
- D** Set debug options.
 Terminate all pending PRACTICE scripts. Explained in **SETUP.PDEBUG.TermScripts**
 Reset global macros after terminating scripts. Explained in **SETUP.PDEBUG.MacroRESet**
 Block window positioning commands. Explained in **SETUP.PDEBUG.BlockPosition**
 Block window closing commands. Explained in **SETUP.PDEBUG.BlockClose**
- E** Set debug window options.
 Open debug window as external window as explained in **SETUP.PDEBUG.WindowExternal**
 Keep debug window on top as explained in **SETUP.PDEBUG.WindowOnTop**
- F** Use the **Reset** button to reset all settings to default.
 The script parameters are cleared and all settings are reverted to their default values working for most usecases. Alternatively reset all settings with the command **SETUP.PDEBUG.RESet**

See also

- **SETUP**

Format: **SETUP.PDEBUG.BlockClose [ON | OFF]**

When debugging a script from within the PRACTICE editor, this command allows to toggle how the execution mode of all window closing commands of a debugged PRACTICE script is handled.

The default state is OFF. If set to ON, the **WinPAGE.RESet** and **WinCLEAR** window close commands are ignored in a debugged script.

SETUP.PDEBUG.BlockPosition**Block window positioning commands**

[build 142827 - DVD 09/2022]

Format: **SETUP.PDEBUG.BlockPosition [ON | OFF]**

When debugging a script from within the PRACTICE editor, this command allows to toggle how the execution mode of positioning commands of a debugged PRACTICE script is handled.

The default state is OFF. If set to ON, the **FramePOS**, **CmdPOS** and **WinPOS** window positioning commands are ignored in a debugged script.

SETUP.PDEBUG.MacroRESet**Reset PRACTICE macros after ending script**

[build 142827 - DVD 09/2022]

Format: **SETUP.PDEBUG.MacroRESet [ON | OFF]**

When debugging a script from within the PRACTICE editor, this command allows to toggle how the cleanup of all PRACTICE macros and global handlers is handled.

The default state is OFF. If set to ON, all global macros and global handlers are cleaned up after the stack is flushed and before the script starts.

This has the same effect as executing the **PMACRO.RESet** command on an empty PRACTICE stack.

NOTE: The setting only takes effect if **SETUP.PDEBUG.TermScripts** is enabled (ON). This is because global macros and handlers are only properly cleared when the PRACTICE stack is empty.

Format: **SETUP.PDEBUG.RESet**

This command resets all parameters modified by **SETUP.PDEBUG** to their default values, which will work for most use cases.

SETUP.PDEBUG.ScriptParams

Set PRACTICE debug script parameters

[build 142827 - DVD 09/2022]

Format: **SETUP.PDEBUG.ScriptParams** [*<parameter_list>*]

This command sets the script parameters for a PRACTICE script when it is started from the PRACTICE editor via the **Debug** or the **Do** button.

The default parameter list is empty. Set the parameter list as if the script is started with the **DO** command. Clear the parameter list by running the command with an empty parameter list.

NOTE:

Any input issued to the command is stored in a history list that is displayed in the script parameter drop-down list of the **SETUP.PDEBUG.state** dialog box. The script parameters are stored at program termination and loaded at program start if the command "**AutoSTOre** , **HISTory**" is placed in the `autostart.cmm` script file.

SETUP.PDEBUG.TermScripts

Terminate all pending PRACTICE scripts

[build 142827 - DVD 09/2022]

Format: **SETUP.PDEBUG.TermScripts** [ON | OFF]

When debugging a script from within the PRACTICE editor, this command allows to toggle how the cleanup of the PRACTICE stack is handled.

The default state is ON. If set to ON, the complete PRACTICE stack will be cleaned up before the script is started. Global defined macros will be kept.

This has the same effect as executing the **END** command.

Format: **SETUP.PDEBUG.WindowExternal [ON | OFF]**

When debugging a script within the PRACTICE editor, use this command to toggle how the **PLIST** debug window is opened

The default state is ON. If set to ON, the editor opens the **PLIST** debugging window with the prefix commands **WinResist** and **WinExt**.

If set to OFF, the **PLIST** window will be opened depending on the currently used graphical user interface window mode (“**Graphical User Interface - Window Modes**” in PowerView User’s Guide, page 11 (ide_user.pdf)).

SETUP.PDEBUG.WindowOnTop

Keep debug window on top

Format: **SETUP.PDEBUG.WindowOnTop [ON | OFF]**

When debugging a script within the PRACTICE editor, this command allows to toggle how the order of overlapping windows (z-order) is handled by the MDI user interface.

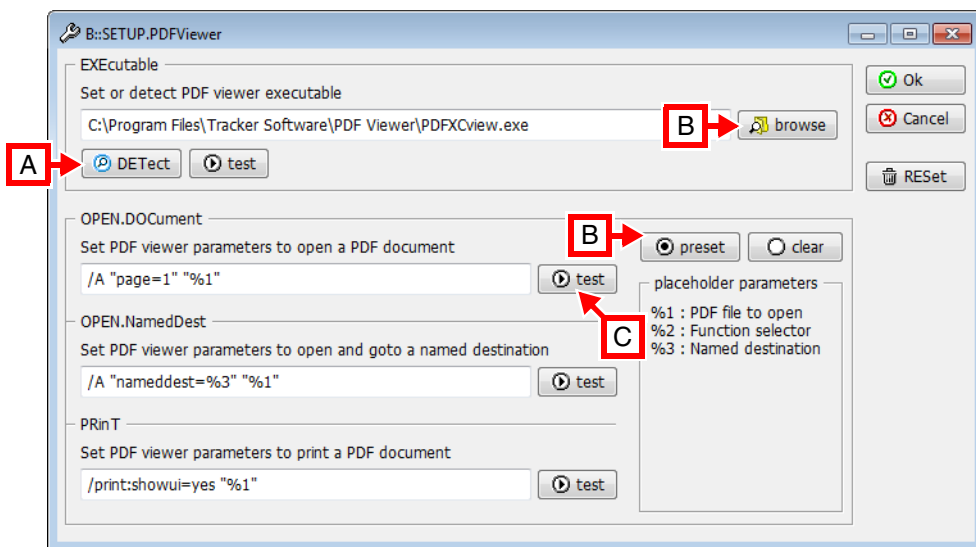
The default state is OFF. When set to ON and the debug window is opened, the **PLIST** debug window is kept on top of all other MDI windows.

NOTE: The command is only effective if TRACE32 runs on the MS Windows operating system.

Format: **SETUP.PDFViewer**
 SETUP.PDFViewer.state (as an alias)

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 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.PDFViewer.EXEcutable](#)
- [SETUP.PDFViewer.OPEN](#)
- [SETUP.PDFViewer.PRinT](#)
- [SETUP.PDFViewer.RESet](#)
- [SETUP.PDFViewer.TEMPorary](#)
- [SETUP](#)
- [HELP](#)

▲ 'Release Information' in 'Legacy Release History'

Format: **SETUP.PDFViewer.EXExecutable** <executable>

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 **EXExecutable** input box in the **SETUP.PDFViewer** dialog window.

See also

■ [SETUP.PDFViewer](#)

SETUP.PDFViewer.OPEN

Open a PDF of the help system

Format: **SETUP.PDFViewer.OPEN.<sub_cmd>**

<sub_cmd>: **DOCument | NamedDest**

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.

See also

■ [SETUP.PDFViewer](#)

Format: **SETUP.PDFViewer.PRinT**

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.

See also

■ [SETUP.PDFViewer](#)

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

Format: **SETUP.PDFViewer.RESet**

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](#)” in PowerView User’s Guide, page 89 (ide_user.pdf).

See also

■ [SETUP.PDFViewer](#)

The **SETUP.PDFViewer.TEMPorary** command group is only used for internal and support purposes.

See also

- [SETUP.PDFViewer.TEMPorary.EXExecutable](#)
- [SETUP.PDFViewer.TEMPorary.PRinT](#)
- [SETUP.PDFViewer](#)
- [SETUP.PDFViewer.TEMPorary.OPEN](#)
- [SETUP.PDFViewer.TEMPorary.RESet](#)

SETUP.PDFViewer.TEMPorary.EXExecutable PDF viewer for demo purposes

Format: **SETUP.PDFViewer.TEMPorary.EXExecutable**

Same meaning as [SETUP.PDFViewer.EXExecutable](#) but nothing is stored in the user preferences.

See also

- [SETUP.PDFViewer.TEMPorary](#)

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

Format: **SETUP.PDFViewer.TEMPorary.OPEN.<sub_cmd>**

<sub_cmd>: **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.

See also

- [SETUP.PDFViewer.TEMPorary](#)

Format: **SETUP.PDFViewer.TEMPorary.PRinT**

Same meaning as [SETUP.PDFViewer.PRinT](#) but nothing is stored in the user preferences.

See also

■ [SETUP.PDFViewer.TEMPorary](#)

SETUP.PDFViewer.TEMPorary.RESet

Reset demo-help configuration

Format: **SETUP.PDFViewer.TEMPorary.RESet**

Same meaning as [SETUP.PDFViewer.RESet](#) but nothing is stored in the user preferences.

See also

■ [SETUP.PDFViewer.TEMPorary](#)

SETUP.PYthon.EXExecutable

Defines path to python interpreter

Format: **SETUP.PYthon.EXExecutable <filepath>**

Defines name of Python interpreter executable, which will be used by [PYthon](#) command group. The existence of executable and version of Python is not verified immediately on setting but on the next [PYthon.RUN](#) call. TRACE32 requires Python version 3.6 or higher. If you want to use the default setting (which is platform dependent - `python.exe` for Windows based OS and `python` for other OS), use an empty string instead of the filepath.

Examples:

```
SETUP.PYthon.EXExecutable "/usr/bin/python"  
SETUP.PYthon.EXExecutable "c:/p/bin/python.exe"  
SETUP.PYthon.EXExecutable "python.exe"  
SETUP.PYthon.EXExecutable ""
```

Format: **SETUP.QUITDO** [*<file>*]

Registers a PRACTICE script *<file>* (*.cmm) that is called when leaving the TRACE32 system. The **SETUP.QUITDO** command is typically included in a start-up script.

<file>

Full path to the PRACTICE script to be executed when TRACE32 is closed. The directory search paths defined with **PATH.Set** aren't considered. The script must end with a **QUIT** command to really quit the TRACE32 system. The *<file>* can be used to automatically save session settings.

Example: When you start TRACE32, the start-up script **start.cmm** calls the **windows.cmm** to restore the window positions of the previous session and registers the **close.cmm**. When you close TRACE32, the **close.cmm** automatically stores the window positions in the **windows.cmm** for re-use in the next session.

NOTE: We recommend to execute the command `DO ~/windows.cmm` only after the start-up procedure in your start-up script has run to completion. For example, load the ELF file before opening windows that refer to symbols.

```
; (a) start.cmm
;<your_start_up_procedure>

DO ~/windows.cmm ;restore the window positions of the previous session

;instruct TRACE32 to automatically execute the script "close.cmm" when
;you close TRACE32, see (b) close.cmm below
SETUP.QUITDO ~/close.cmm

ENDDO
```

```
; (b) close.cmm
DIALOG.YESNO "Save the window positions for the next session?"

LOCAL &answer
ENTRY &answer

IF &answer==TRUE()
    STOrE ~/windows.cmm Win ;save the window positions in a file
                           ;residing in the TRACE32 system directory

QUIT
```

See also

■ [SETUP](#)
■ [STOrE](#)

■ [SETUP.RESOLVEDIR](#)

■ [STOrE](#)

■ [QUIT](#)

▲ 'System Setup and Configuration' in 'PowerView User's Guide'

Format:	SETUP.RADIX.<mode> RADIX.<mode> (deprecated)
<mode>:	Decimal Hex

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.

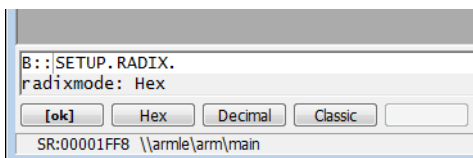
The preferred location for a different radix mode (not the default hex mode) is the user’s own start-up script.

Decimal Number base is decimal.

Hex (default) Number base is hex - default.

Example: By entering `SETUP . RADIX .` at the command line *without executing* the command, you can display the currently used RADIX mode in the TRACE32 [message line](#).

SETUP . RADIX .



See also

- [SETUP](#) □ [RADIX\(\)](#)

Format: **SETUP.RANDOM** [*<seed>*]

Sets a seed value for the internal pseudo random number generator. *<seed>* is an unsigned 64-bit 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 ['PowerView User's Guide'](#)

Format: **SETUP.RESOLVEDIR [ON | OFF]**

Default: OFF.

- OFF** Symbolic links are not resolved when cahnging the current working directory with **ChDir**.
- ON** Symbolic links are resolved when cahnging the current working directory with **ChDir**.

See also

- [SETUP](#) ■ [SETUP.QUITDO](#)

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 'PowerView User's Guide'

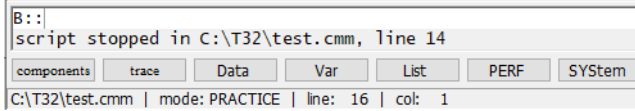
SETUP.STOPMESSAGE

Print message when STOP command is executed

Format: **SETUP.STOPMESSAGE [ON | OFF]**

Default: OFF.

Controls whether a message is printed to the TRACE32 status line when a PRACTICE script executes a **STOP** command.



See also

- [SETUP](#)

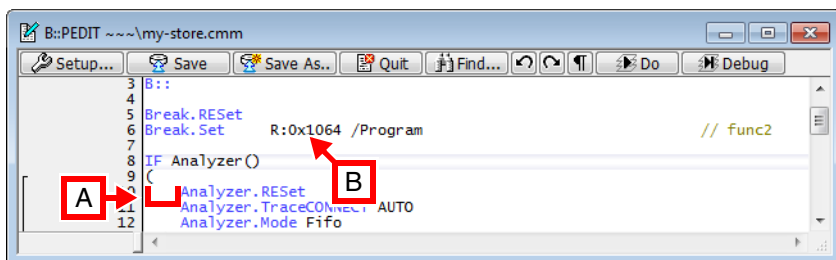
Format:	SETUP.STOre.<sub_cmd>
<sub_cmd>:	INDENTation TAB <spaces> SYMBOLIC [ON OFF] RESet

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. <ul style="list-style-type: none"> 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:

```
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
```



A Indentation: 4 spaces

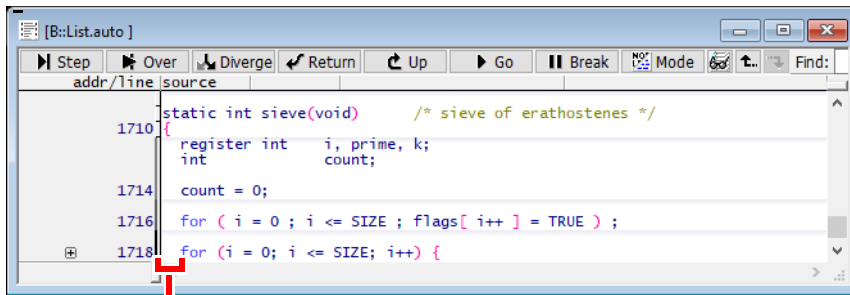
B Only the address, here **R:0x1064** of **func2**, is stored (**SYMBOLIC OFF**).

See also

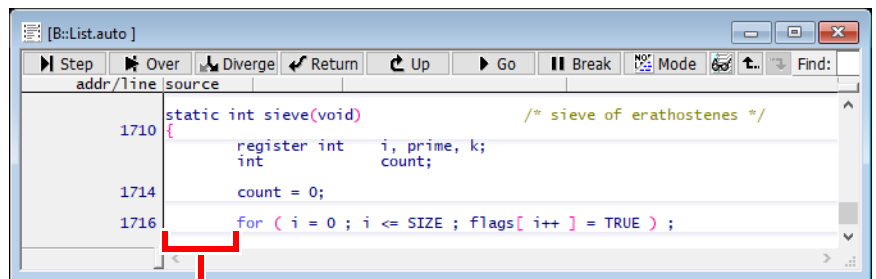
- [SETUP](#)
- [STOre](#)
- [AutoSTOre](#)
- [ClipSTOre](#)
- [AutoSTOre](#)
- [ClipSTOre](#)
- [STOre](#)

Format: **SETUP.TabSize** [*<width>*]

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).



tabsize = 2



tabsize = 8

See also

- [SETUP](#) ■ [SETUP.EDITOR.IndentSize](#)
- ▲ 'System Setup and Configuration' in 'PowerView User's Guide'

Format: **SETUP.TIMEFORM [ON | OFF]**

Time values are displayed by TRACE32 in an easily readable format. If this option is activated, time values are displayed in an scientific floating point format. This format is easier to process by external tools.

default (OFF)	scientific (ON)
12.345us	12.34e-6
12.345ms	12.34e-3
12.345s	12.345
12.345ks	12.345e3

See also

■ [SETUP](#)

▲ ['System Setup and Configuration'](#) in ['PowerView User's Guide'](#)

Format 1: **SETUP.UpdateRATE** <time> | <value>

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.

```
SETUP.UpdateRATE 500.ms      ; update the window information all 500 ms
SETUP.UpdateRATE 3.          ; update the window information 3 times/s
```

See also

- [SETUP](#)
- [SYStem.POLLING](#)
- ▲ ['System Setup and Configuration'](#) in ['PowerView User's Guide'](#)

Format: **SETUP.WARNSTOP** [ON | OFF]

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 ['PowerView User's Guide'](#)

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.

Examples of TRACE32 commands that create XML export files are the commands of the [List.EXPORT](#) and [COverage.EXPORT](#) command group or the [PRinTer](#) command group.

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\`

Example: For an example, please 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.EXPORT](#)

Format: **SHA1SUM** <file> [/<options>]

<option>: **EoIToLf**

Calculates a 160-bit checksum for the given files using the Secure Hash Algorithm (SHA-1). The result is displayed in the **AREA** window. Use the pre-command **SILENT** to suppress the output to the **AREA** window. The result is also available via the PRACTICE function **FILE.SUM()**.

<file> Name of the file for which a checksum is calculated.

EoIToLf For calculating the checksum, this option treats the pair of bytes 0x0D and 0x0A (Carriage Return + Line Feed) as a single 0x0A (Line Feed).

See also

[FILE.SUM\(\)](#)

[▲ 'Release Information' in 'Legacy Release History'](#)

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**, and **FIND**
- **Data.LOAD.***, **PWD**, **ChDir**
- **TargetSystem.NewInstance**
- **SYSTEM.Option** commands that are *manually* toggled at the TRACE32 command line by omitting the keywords **ON** / **OFF**, e.g. **SYSTEM.Option.MMUSPACES**

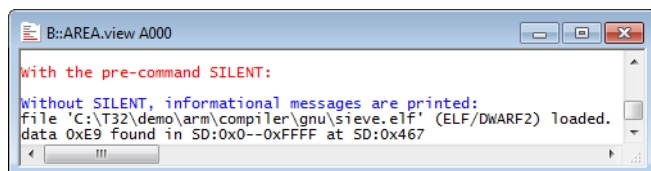
Example: For demo purposes, the same two commands are executed with and without the **SILENT** pre-command. The result is shown in the **AREA** window below.

```
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--0xffff 0xE9
```



See also

- [AREA](#)

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

■ [SETUP:ICONS](#)
■ [TOOLBAR](#)

■ [STATUSBAR](#)

■ [SUBTITLE](#)

■ [TITLE](#)

STATUSBAR

STATUSBAR

Toggle state line

Format: **STATUSBAR [ON | OFF]**

The **STATUSBAR** command without argument toggles the TRACE32 [state line](#).

ON Displays the state line.

OFF Hides the state line.

See also

■ [SETUP:ICONS](#)

■ [SOFTKEYS](#)

■ [SUBTITLE](#)

■ [TITLE](#)

■ [TOOLBAR](#)

Format:	STOre <i><file></i> [[% <i><format></i>] <i><item></i> ...] [<i><option></i>]
<i><format></i> :	sYmbol NosYmbol
<i><item></i> :	ALL HISTory Win WinPAGE ... <i><device_specific_settings></i>
<i><option></i> :	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 analyzers, with more system specific options.

<format>, *<option>* 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

- [SETUP.QUITDO](#)
- [SETUP.STOre](#)
- [AutoSTOre](#)
- [ClipSTOre](#)
- ▲ 'Window System' in 'PowerView User's Guide'
- ▲ 'Release Information' in 'Legacy Release History'
- ▲ 'Breakpoint Handling' in 'Training Basic Debugging'
- ▲ 'Breakpoint Handling' in 'Training Basic SMP Debugging'

Format: **SUBTITLE** [[%<formats>] <your_text>] ...

<format>: **Ascii**
 Binary
 Decimal
 Hex
 String

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 [TRACE32 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 PowerView instance:

```
;maincluster.cmm
;... your code

TITLE "TRACE32 PowerView      ; main cluster"
SUBTITLE %String "      ; main cluster"

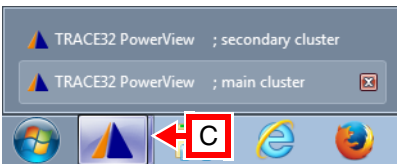
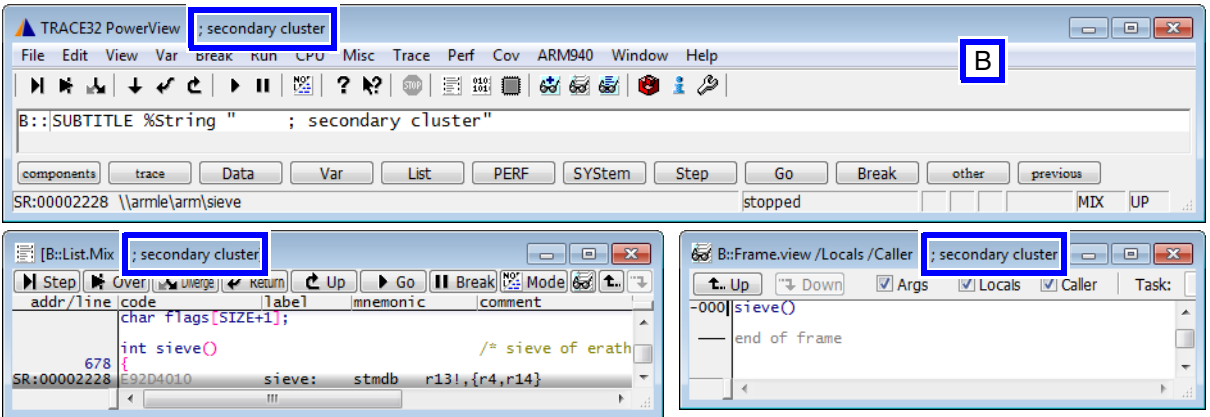
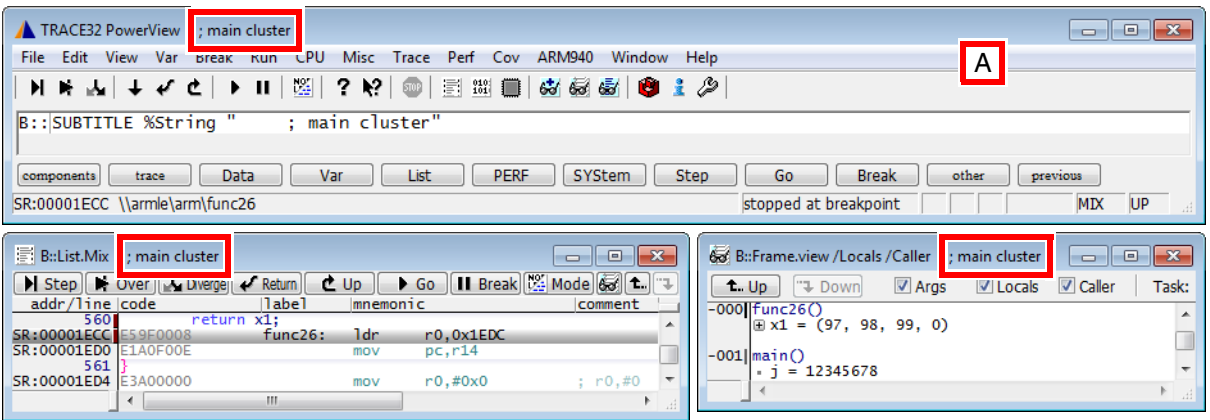
;... your code
```

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 PowerView instance:

```
;secondarycluster.cmm
;... your code

TITLE "TRACE32 PowerView      ; secondary cluster"
SUBTITLE %String "      ; secondary cluster"

;... your code
```



A First TRACE32 PowerView instance

B Second TRACE32 PowerView instance

C In this example, the two TRACE32 PowerView 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 **"Default Advanced Settings"** in T32Start, page 13 (app_t32start.pdf).

See also

- SETUP.ICONS
- SOFTKEYS
- STATUSBAR
- InterCom.ENable
- InterCom.NAME
- TITLE
- TOOLBAR

▲ 'Release Information' in 'Legacy Release History'

Format: **TAR** <archive_name> <file_selector> [/<options>]

<option>: **NoRecursion**
ListOnly

Packs the selected files *without compression* into a tape archive formatted archive. The files are selected from the directory path given by the <file_selector>.

By default, the given directory from the <file_selector> and all its subdirectories are scanned recursively down. All selected files from this directory tree are then stored into the archive.

<archive_name>	File name of the archive to be created.
<file_selector>	The file selector may contain a directory and a file name with wildcard characters to select appropriate files.
NoRecursion	Switch off subdirectory tree scanning. Store only files from the given directory of the <file_selector>.
ListOnly	The files are not packed into an archive but just listed in the default AREA window A000 . 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.

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
```

Example 2:

```
;list all *.c files from the TRACE32 demo directory and all its
;subdirectories in the default AREA.view window
TAR ~/archive.tar    ~/demo/*.c /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

TIMEOUT

Specify timeout for TRACE32 command

Format: **TIMEOUT** *<period>* *<command>*

Terminates a *<command>* after the specified *<period>* has elapsed. The **TIMEOUT** command has same effect as clicking the **STOP** button on the TRACE32 [main toolbar](#) after a defined time.

<period>

Parameter Type: [Time value](#).

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 500.ms"
)
```

See also

■ [SCREEN.WAIT](#)

■ [WAIT](#)

□ [TIMEOUT\(\)](#)

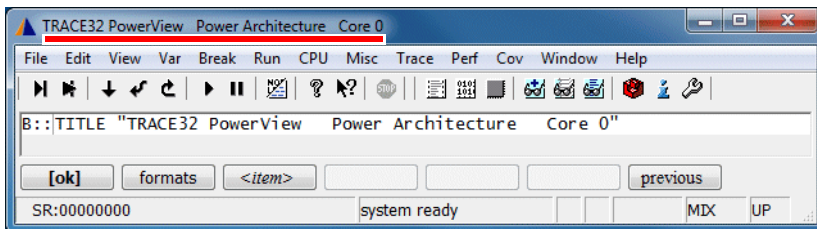
TITLE

TITLE Define a main window title for a TRACE32 PowerView GUI

Format: **TITLE** [[%<formats>] "<your_text>"] ...

<format>:
Ascii
Binary
Decimal
Hex
String

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 PowerView GUIs of a multicore or multi-CPU target.

Example:

```
TITLE %String "TRACE32 Debugger for CPU0"  
TITLE %String "TRACE32 for MPC5676R"
```

See also

- [TOOLBAR](#)
- [InterCom.Enable](#)
- [InterCom.NAME](#)
- [SETUP.ICONS](#)
- [SOFTKEYS](#)
- [STATUSBAR](#)
- [SUBTITLE](#)
- [TITLE\(\)](#)

TOOLBAR

TOOLBAR

Toggle toolbar

Format: **TOOLBAR [ON | OFF]**

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](#)

▲ ['PowerView - Screen Display' in 'PowerView User's Guide'](#)

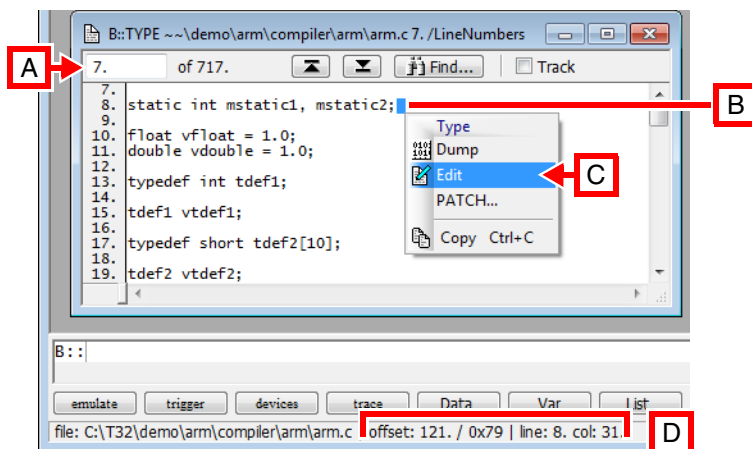
Format: **TYPE** <file> [<line>] [/<options>]

<options>: **Track**
LineNumbers

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.

Example:

```
;display file and scroll to line 7
;display line numbers
TYPE ~/demo/arm/compiler/arm/arm.c 7. /LineNumbers
```



A Scroll to this line number.

B Current selection.

C Right-click for popup menu.

EDIT opens the file in the TRACE32 editor. To configure an external editor, use [SETUP.EDITEXT](#).

D Offset of current selection in decimal and hex as well as in line and column number.

See also

■ ComPare
■ PATCH

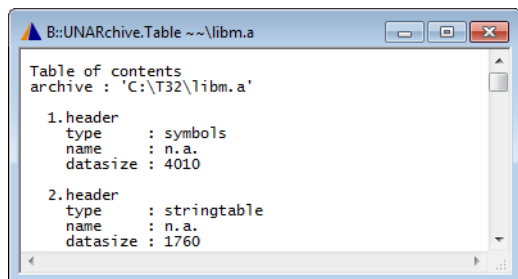
■ DUMP
□ TRACK.COLUMN()

■ EDIT.file
□ TRACK.LINE()

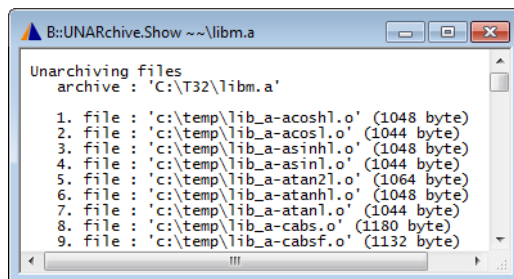
■ FIND

▲ 'File and Folder Operations' in 'PowerView 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.Table displays the files in the library.



UNARchive.Show displays the result of the extract operation.

See also

- [UNARchive.extract](#)
- [UNARchive.Show](#)
- [UNARchive.Table](#)
- [UNPACK](#)
- [UNZIP](#)

- ▲ 'File and Folder Operations' in 'PowerView User's Guide'
- ▲ 'Release Information' in 'Legacy 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](#)

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.TableDisplay 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](#)

Format: **UNPACK** <source> [<destination>]

The compressed file is expanded back to the original file format. The source must be a file in LZW encoding, generated by the **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.

Example:

```
PACK mcc.abs mcc.pak      ; compress object file
; ...
UNPACK mcc.pak mcc.abs    ; restore original file
```

See also

■ [UNARchive](#)

■ [UNZIP](#)

■ [PACK](#)

■ [ZIP](#)

▲ ['File and Folder Operations'](#) in ['PowerView User's Guide'](#)

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.

Example:

```
UNZIP \t32\man.t32                   ; un-pack online manual
```

See also

■ [UNARchive](#)

■ [UNPACK](#)

■ [PACK](#)

■ [ZIP](#)

▲ ['File and Folder Operations'](#) in ['PowerView User's Guide'](#)

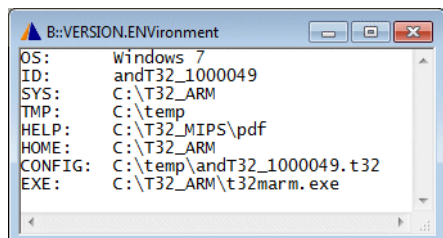
Using the **VERSION** command group, you can display version information about the TRACE32 hardware modules and software as well as the TRACE32 environment settings.

See also

- [VERSION.ENVIRONMENT](#)
- [VERSION.SOFTWARE](#)
- [VERSION.view](#)
- ▲ ['VERSION Functions' in 'General Function Reference'](#)
- [VERSION.HARDWARE](#)
- [VERSION.ThirdPartyLicenses](#)
- [LICENSE](#)

Format: **VERSION.ENVIRONMENT**

Displays the currently used environment settings of the TRACE32 software in the **VERSION.ENVIRONMENT** window. This includes e.g. the currently started executable, TRACE32 system directory, TRACE32 configuration file, etc.



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.PresentHELPDDirectory\(\)](#)
- [OS.PresentSystemDirectory\(\)](#)
- [OS.PresentWorkingDirectory\(\)](#)
- [VERSION.ENVIRONMENT\(\)](#)
- [VERSION.view](#)
- [OS.PresentConfigurationFile\(\)](#)
- [OS.PresentExecutableFile\(\)](#)
- [OS.PresentHomeDirectory\(\)](#)
- [OS.PresentTemporaryDirectory\(\)](#)
- [OS.VERSION\(\)](#)

Format: **VERSION.HARDWARE**

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 (*.cmm) to return individual values from the window. For more information, refer to the [□ functions\(\)](#) listed below.

See also

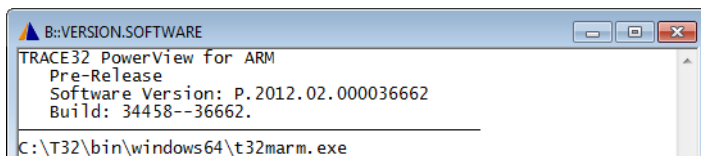
- [VERSION](#) ■ [VERSION.view](#) □ [CABLE.NAME\(\)](#) □ [ID.PREPROcessor\(\)](#)
- [SYStem.USEMASK\(\)](#) □ [VERSION.SERIAL.CABLE\(\)](#) □ [VERSION.SERIAL.DEBUG\(\)](#)
- ▲ ['VERSION Functions'](#) in ['General Function Reference'](#)
- ▲ ['Version Management and Licensing'](#) in ['PowerView User's Guide'](#)

VERSION.SOFTWARE

Display software versions

Format: **VERSION.SOFTWARE**

Displays the versions of the TRACE32 software modules in the **VERSION.SOFTWARE** window.



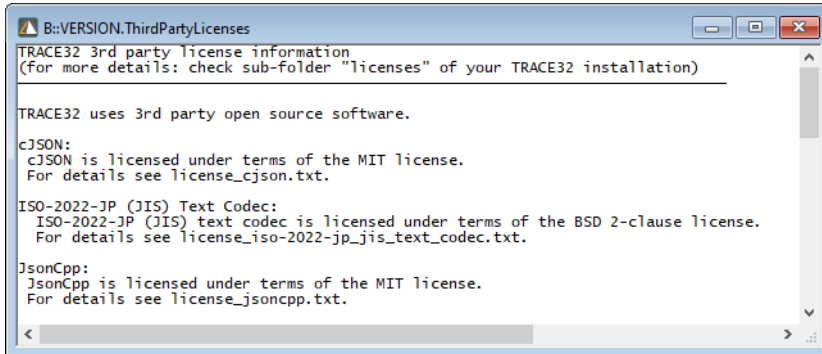
PRACTICE functions can be used in PRACTICE scripts (*.cmm) to return individual values from the window. For more information, refer to the [□ functions\(\)](#) listed below.

See also

- [VERSION](#) ■ [VERSION.view](#) □ [OS.PresentExecutableFile\(\)](#) □ [VERSION.BUILD\(\)](#)
- [VERSION.BUILD.BASE\(\)](#) □ [VERSION.SOFTWARE\(\)](#)
- ▲ ['Version Management and Licensing'](#) in ['PowerView User's Guide'](#)
- ▲ ['Appendix - About the TRACE32 Software Version Numbers'](#) in ['PowerView User's Guide'](#)

Format: **VERSION.ThirdPartyLicenses**

Displays the versions of the TRACE32 3rd party components and their licenses terms in the **VERSION.ThirdPartyLicenses** window.



```
B::VERSION.ThirdPartyLicenses
TRACE32 3rd party license information
(for more details: check sub-folder "licenses" of your TRACE32 installation)

TRACE32 uses 3rd party open source software.

cJSON:
cJSON is licensed under terms of the MIT license.
For details see license_cjson.txt.

ISO-2022-JP (JIS) Text Codec:
ISO-2022-JP (JIS) text codec is licensed under terms of the BSD 2-clause license.
For details see license_iso-2022-jp_jis_text_codec.txt.

JsonCpp:
JsonCpp is licensed under terms of the MIT license.
For details see license_jsoncpp.txt.
```

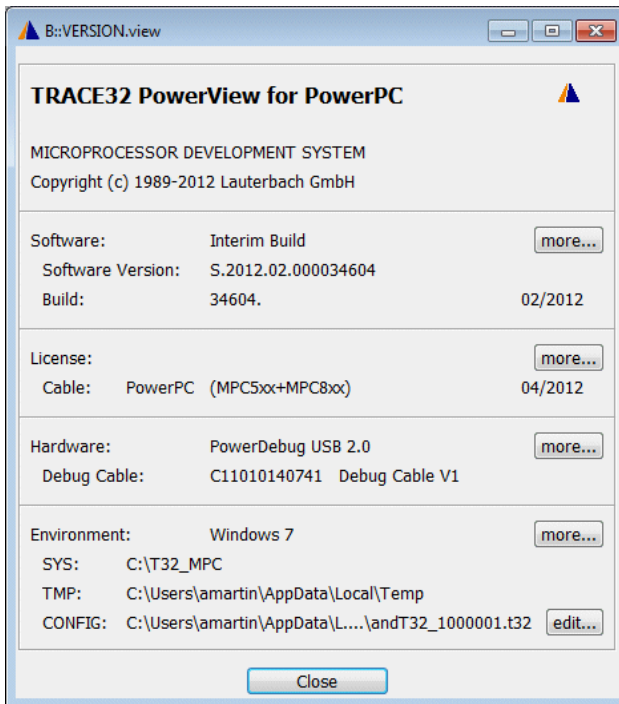
See also

■ [VERSION](#)

■ [VERSION.view](#)

Format: **VERSION.view**

Displays the versions of the TRACE32 modules (hardware and software) and TRACE32 hardware serial numbers in the **VERSION.view** window.



See also

- [VERSION](#)
 - [VERSION.HARDWARE](#)
 - [VERSION.ThirdPartyLicenses](#)
 - [VERSION.ENVIRONMENT](#)
 - [VERSION.SOFTWARE](#)
- ▲ 'Version Management and Licensing' in 'PowerView User's Guide'
- ▲ 'Appendix - About the TRACE32 Software Version Numbers' in 'PowerView User's Guide'

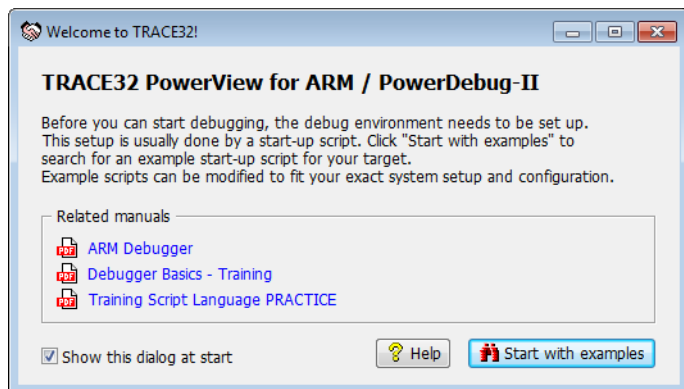
WELCOME

WELCOME

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.FILTER](#)
 - [WELCOME.CONFIG.ReMoveDIR](#)
 - [WELCOME.CONFIG.RESet](#)
 - [WELCOME.CONFIG.state](#)
 - [WELCOME](#)
 - [WELCOME.view](#)

Format: **WELCOME.CONFIG.ADDDIR** *<path>*

See also

■ [WELCOME.CONFIG](#)

Format: **WELCOME.CONFIG.FILTER** " *<filter>* "

Default: *.cmm

See also

■ [WELCOME.CONFIG](#)

Format: **WELCOME.CONFIG.ReMoveDIR** *<path>*

See also

■ [WELCOME.CONFIG](#)

Format: **WELCOME.CONFIG.RESet**

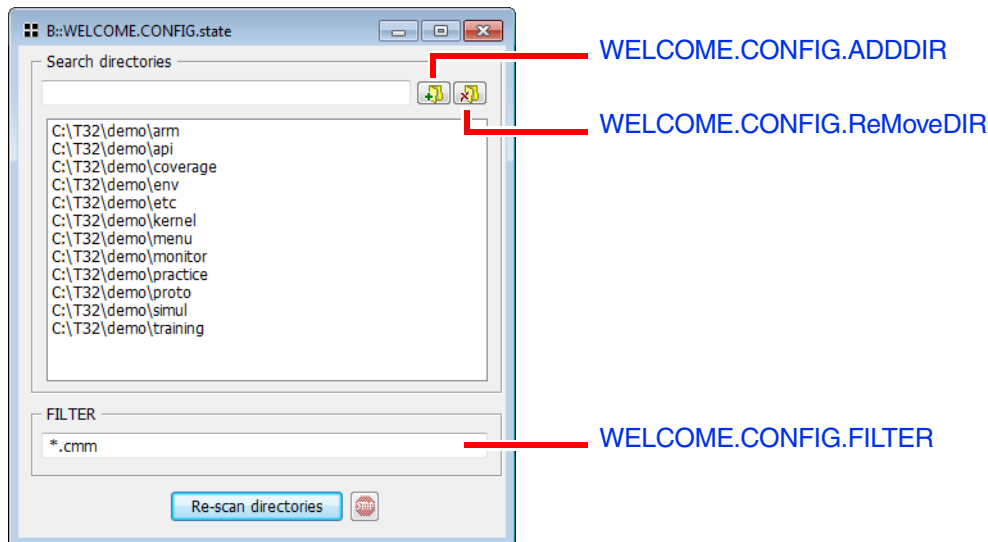
See also

■ [WELCOME.CONFIG](#)

Format: **WELCOME.CONFIG.state**

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](#)

Format: **WELCOME.SCRIPTS**

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](#)” in PRACTICE Script Language User’s Guide, page 24 (practice_user.pdf).

See also

- [WELCOME](#) ■ [WELCOME.view](#)
- ▲ ['Release Information' in 'Legacy Release History'](#)

Format: **WELCOME.STARTUP**

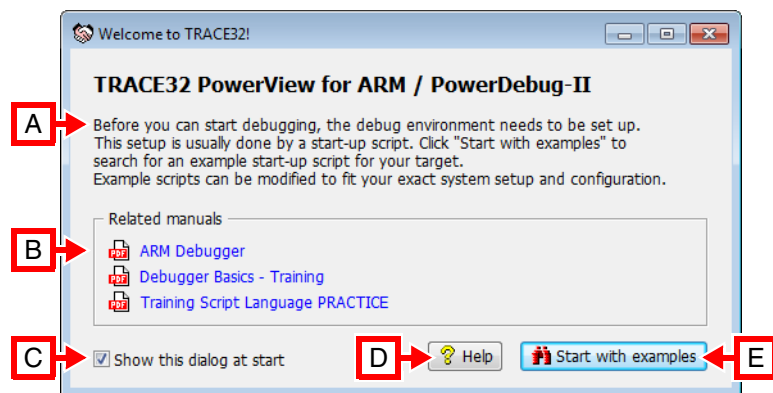
Displays the **Welcome to TRACE32!** window unless it was disabled by the user; see check box in the **Welcome to TRACE32** window ([WELCOME.view](#)).

See also

- [WELCOME](#) ■ [WELCOME.view](#)
- ▲ ['Release Information' in 'Legacy Release History'](#)

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 'Legacy Release History'

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 | ■ WinCLEAR | ■ WinDEFaultSIZE | ■ WinDuplicate |
| ■ WinExt | ■ WinFIND | ■ WinFreeze | ■ WinLarge |
| ■ WinMid | ■ WinOverlay | ■ WinPAGE | ■ WinPAGE.Create |
| ■ WinPAGE.Delete | ■ WinPAGE.List | ■ WinPAGE.REName | ■ WinPAGE.RESet |
| ■ WinPAGE.select | ■ WinPAN | ■ WinPOS | ■ WinPrint |
| ■ WinPRT | ■ WinResist | ■ WinRESIZE | ■ WinSmall |
| ■ WinTABS | ■ WinTOP | ■ WinTrans | □ WINDow.EXIST() |
| □ WINDow.POSition() | □ WINDow.EXIST() | | |

▲ ‘[WINDow Functions](#)’ in ‘[PowerView Function Reference](#)’

Format: **WinBack.***<command>*

Pre-command for creating a background window, i.e., the window is pushed into the background after operations.

See also

- [Win](#)
- [WinFreeze](#)
- [WinResist](#)
- [WINDOW.NAME\(\)](#)
- ▲ ['Window System' in 'PowerView User's Guide'](#)

WinCLEAR

Erase windows

[\[Example\]](#)

Format: **WinCLEAR** [**WinTOP** | {*<window_name>*} | *<page_name>*]

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 **WinResist.***<window>* or **WR.***<window>* are **not** cleared.

WinTOP (or TOP as an alias)	Deletes the uppermost window.
<i><window_name></i>	Window names are case-sensitive. They are created with the WinPOS command.
<i><page_name></i>	Page names are case-sensitive. They are created with the WinPAGE.Create command.

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 'PowerView User's Guide'](#)
- ▲ ['I/O Commands' in 'Training Script Language PRACTICE'](#)

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 executing 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.

<i><hsize></i>	Applies a user-defined default width to windows.
<i><vsize></i>	Applies a user-defined default height to windows.
no parameters	Restores the TRACE32 settings for window default sizes. NOTE: You can display the current user-defined default size in the TRACE32 state line by just typing the command and appending a blank.

TRACE32 ignores any user-defined setting (width or height or both) that exceeds the built-in minimum or maximum size for a particular window. A warning is displayed in the TRACE32 state line if the user-defined setting exceeds the desktop size.

Example: This 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. 10. ;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
AREA.view

WinRESIZE 120. 20. myWin01 ;Resize the window named myWin01

;WinDEFaultSIZE has no effect on dialog-style windows, such as:
SYSTEM.state

```

See also

■ [Win](#)

■ [WinPOS](#)

■ [WinRESIZE](#)

▲ ['Release Information' in 'Legacy Release History'](#)

Format:	WinDuplicate
---------	---------------------

Allows to open another window with exactly the same command than an already existing window, because sometimes it is useful to open two or more windows with the same command line and arguments.

Usually if you execute a command to open a window, PowerView will check if a window with exactly the same command line already exists. If a such a window exists, it bring this window in the foreground instead of opening a new window. This happens only if the command line is identical in its complete notation, considering case sensitivity and all of the commands arguments.

If you execute **WinDuplicate** before opening a new window you will get a new window, no matter if an identical window already exists or not. However, some special windows can exist in PowerView only once and those windows will not be created again, even when using **WinDuplicate**.

You can see the command line, which was used to open any of the existing windows, in the window **WinPAGE.List**. The command line of an existing window is *usually* also its window title (unless this was changed with the **WinPOS** command).

Example: This script will open two SYStem.CONFIG windows with the same window title.

```
;Open a first configuration window
SYStem.CONFIG

;Open an area window
AREA

;Start duplicating a window
WinDuplicate

;Open a second configuration window with the same title.
SYStem.CONFIG
```

See also

■ [Win](#)

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.

Example:

```
;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 ‘PowerView User’s Guide’](#)

WinFIND

Search for text in window

[\[Example\]](#)

Format: **WinFIND** *[[<lines>] "<string>"] [WinTOP | <window_name>] [/<option>]*

<option>: **Back**
Case

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 *<window_name>* does not exist.

WinTOP (or TOP as an alias)	Performs a search operation in the uppermost window.
<code><window_name></code>	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.

Example:

```

; find the string "Shell>" in the terminal window
WinPOS 4. 4. 80. 25. 0. 0. MyTerm
TERM.METHOD COM COM1 115200. 8 NONE 1STOP NONE
TERM.view
WinFIND "Shell>" MyTerm
IF FOUND()
    PRINT "EFI Shell"

```

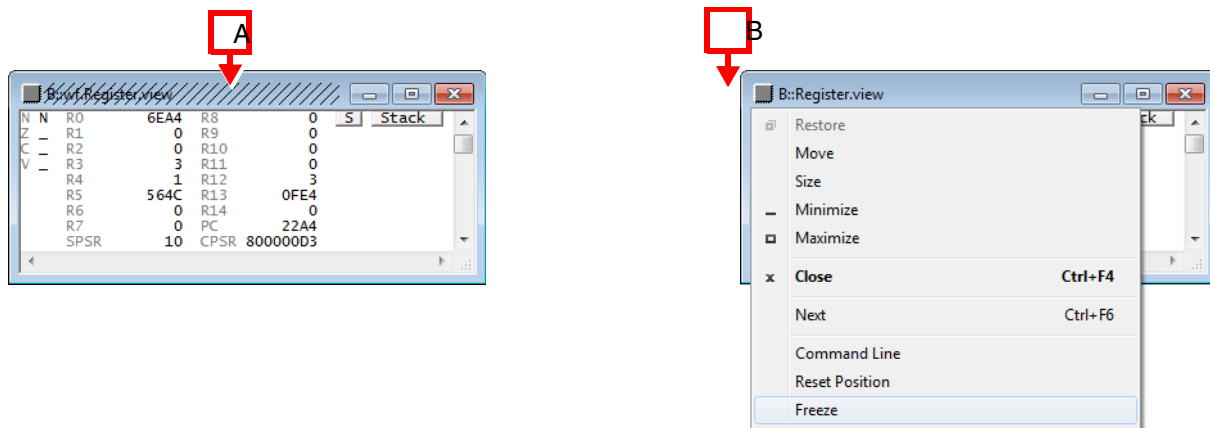
See also

- [Win](#)
- [Data.GREP](#)
- ▲ 'Window System' in 'PowerView User's Guide'
- [FIND](#)
- [FOUND\(\)](#)
- [Data.Find](#)
- [WINDOW.NAME\(\)](#)
- [Data.GOTO](#)

Format: **WinFreeze.***<command>*

Pre-command for generating a frozen window. Note that frozen window are not updated to the current state.

You can also choose **Freeze** from the window manager menu (left mouse) to freeze or unfreeze the window contents.



A Diagonal lines indicate that the window contents are frozen.

B Click the top left icon to open the [window manager menu](#).

Example:

```
WinFreeze.Register.view ;Open the Register window as a frozen window
```

See also

■ [Win](#)

■ [WinBack](#)

■ [WinResist](#)

□ [WINDOW.NAME\(\)](#)

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

Format: **WinLarge.<command>**

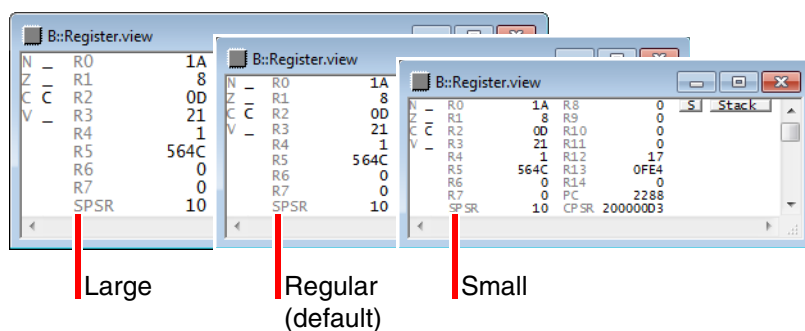
Pre-command for generating a window with large font. Switching to large font is very useful in presentations before large audiences.

Example:

```
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 'PowerView User's Guide'

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 'PowerView User's Guide'](#)

WinOverlay

Pile up windows on top of each other

Format: **WinOverlay.***<command>*

Superimposes the next window on the active window.

This behavior is used in a [List](#) or [Data.GREP](#) window to open a new [List](#) window on top of and with nearly the same size as the active window. Press **Esc** to return to the previous window, or drag the new window to a new position to make the previous window visible again.

Double-clicking a function or variable name in an HLL listing executes the **WinOverlay** command by default.

- NOTE:**
- Window sizes may vary for windows that do not display the same type of content.
 - The double-click behavior within a [List](#) or [Data.GREP](#) window can be changed by the [SETUP.LISTCLICK](#) command.

Example: The [Data.List](#) and [List.auto](#) windows display the same type of content and can thus be exactly superimposed in terms of position and size.

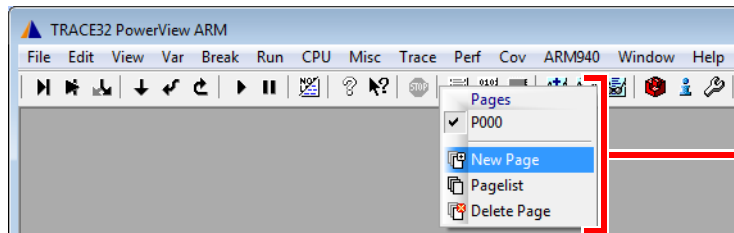
```
Data.List ;active window

WinOverlay.List.auto func2 ;next window is superimposed on Data.List,
                           ;displaying a listing for the function func2
```

See also

- [Win](#)
- [Data.GREP](#)
- [List](#)
- [SETUP.LISTCLICK](#)
- ▲ ['Window System' in 'PowerView 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.Create
- WinPAGE.Delete
- WinPAGE.List
- WinPAGE.REName
- WinPAGE.RESet
- WinPAGE.select
- WINPAGE.EXIST()
- ▲ 'PowerView - Screen Display' in 'PowerView User's Guide'
- ▲ 'Window System' in 'PowerView User's Guide'

WinPAGE.Create

Create and select page

[\[Example\]](#)

Format: **WinPAGE.Create** [*<page_name>* [/NoSElect]] | [, /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 **P000**, **P001**, etc.

<i><page_name></i>	<ul style="list-style-type: none"> 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. <p>Page names are case-sensitive.</p>
,	Auto-increments the name of the next page; additionally you can use NoSElect .
NoSElect	A new page is created in the background, but not selected. The current page continues to remain the active page.

Example:

```
WinPAGE.Create ANALYZER      ; create a page for Analyzer windows
Analyzer.List                ; create an Analyzer window on this page
WinPAGE.select P000          ; select the default page
```

See also

■ [WinPAGE](#)

■ [WinPAGE.List](#)

■ [Win](#)

□ [WINPAGE.EXIST\(\)](#)

▲ ['Window System' in 'PowerView User's Guide'](#)

WinPAGE.Delete

Delete page

Format: **WinPAGE.Delete** *<page_name>*

Removes one page from the page list including all windows within it.

<page_name>

Page names are case-sensitive.

Example:

```
WinPAGE.Delete P000          ; delete the first page
```

See also

■ [WinPAGE](#)

■ [WinPAGE.List](#)

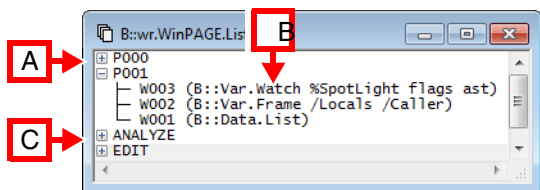
■ [Win](#)

□ [WINPAGE.EXIST\(\)](#)

▲ ['Window System' in 'PowerView User's Guide'](#)

Format: **WinPAGE.List** [/ShowAlways]

Opens the **WinPAGE.List** window, listing all pages and their windows by name.



- 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** *<page_name>*.

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**.

```
WinResist.WinPAGE.List      ; open a resistant window to navigate
                             ; between pages
```

See also

[WinPAGE](#) [WinPAGE.Create](#) [WinPAGE.Delete](#) [WinPAGE.REName](#)
[WinPAGE.RESet](#) [WinPAGE.select](#) [Win](#) [WINPAGE.EXIST\(\)](#)

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

Format: **WinPAGE.REName** <old_pagename> <new_pagename>

Renames an existing page. Page names are case-sensitive.

Example:

```
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 'PowerView User's Guide'](#)

WinPAGE.select

Select page

Format: **WinPAGE.select** [<page_name>]

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 'PowerView User's Guide'](#)

Format: **WinPAN** [<x>] [<y>] [**WinTOP** | <window_name>]

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 <window_name> does not exist.

<x>	Use positive values to pan to the right; negative values to pan to the left.
<y>	Use positive values to scroll down; negative values to scroll up.
WinTOP (or TOP as an alias)	Scrolls or pans the uppermost window.
<window_name>	Window names are case-sensitive. They are created with the WinPOS command.

See also

■ [WinPOS](#)

■ [Win](#)

□ [WINDOW.NAME\(\)](#)

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

Format: **WinPOS** [*<pos>*] [*<size>*] [*<scale>*] [*<window_name>*] [*<state>*] [*<header>*]

<state>: **Normal | Iconic | Maximized**

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% , , , myWinName
)
ELSE
(
    ;before build 72592, only 2 commas were required
    WinPOS 50% 0% 50% 100% , , myWinName
)
```

<i><pos></i>	<ul style="list-style-type: none"> <i><left></i> = x-coordinate as a floating point or integer or percentage value. <i><up></i> = y-coordinate as a floating point or integer or percentage value.
<i><size></i>	<ul style="list-style-type: none"> <i><hsize></i> = width of a window as an integer or percentage value (range: 0% to 100%). <i><vsize></i> = height of a window as an integer or percentage value (range: 0% to 100%).
<i><scale></i>	<ul style="list-style-type: none"> <i><hscale></i> = width of the scale area of a window. <i><vscale></i> = height of the scale area of a window.
<i><window_name></i>	The <i><window_name></i> 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.
<i><header></i>	Specify the user-defined window caption as a quoted string.

Examples

```
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
```

```
;                               <window_name> <state>           <header>
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
```

```
;the individual arguments can optionally be comma-separated
WinPOS 10. , 20. , 30 , 40. , 1. , 2. , myName3
Frame.view
```

Script in Demo Folder

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.

```
;register the user-defined WinPOS2 command
ON CMD WinPOS2 DO "~/demo/practice/winpos.cmm"

WinPOS2 0%,0%,50%,50%,,,myListWindow
List.auto

WinPOS2 50%,0%,,,,,myRegisterWindow
Register.view /SpotLight
```

See also

■ [WinPAN](#)
■ [WinTABS](#)

■ [Win](#)
■ [WinTOP](#)

■ [WinDEfaultSIZE](#)
□ [WINDOW.EXIST\(\)](#)

■ [WinRESIZE](#)
□ [WINDOW.NAME\(\)](#)

- ▲ ['Window System' in 'PowerView User's Guide'](#)
- ▲ ['Release Information' in 'Legacy Release History'](#)
- ▲ ['I/O Commands' in 'Training Script Language PRACTICE'](#)

Format: **WinPrint.<command>**

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.

Example 1:

```
WinPrint.Data.dump 0--0xfff  
  
WinPrint.Analyzer.List (-1000.)--100. Address Data sYmbol
```

Example 2: For an example of how to print the contents of TRACE32 windows to file in XML format, see **PRinTer.FILE**.

See also

- [WinPRT](#)
- [PRinTer.EXPORT](#)
- [WINDOW.NAME\(\)](#)
- ▲ ['Printer Operations' in 'PowerView User's Guide'](#)
- ▲ ['Window System' in 'PowerView User's Guide'](#)
- ▲ ['Window System' in 'PowerView User's Guide'](#)
- ▲ ['Release Information' in 'Legacy Release History'](#)
- [Win](#)
- [PRinTer.FILE](#)
- [PRINT](#)
- [PRinTer.OFFSET](#)
- [PRinTer](#)
- [PRinTer.OPEN](#)

Format: **WinPRT [WinTOP | <window_name>]**

Prints the uppermost window or the window that has the specified name. It is the same command as **Print** in the [window manager menu](#). **WinPRT** is used to make multi-page printouts of windows where the print range can be specified only in the form of lines.

If the print range can be specified as an address, symbol or record range, use the **WinPrint.<window>** command.

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 <window_name> does not exist.

WinTOP, TOP	Prints the uppermost window. TOP is an alias.
<window_name>	Window names are case-sensitive. They are created with the WinPOS command.

Example: In this script, the first 80 lines of a **PER.view** window are printed to file.

```

;define a) the width and b) the height of the PER.view window:
;a) set the width to the size of the longest line, here 200 characters
;b) set the height to 10 lines so that we can print in steps of 10 lines
WinPOS 0. 0. 200. 10. 0. 0. myWIN
PER.view , "*"                ;open the window and expand all subtrees
SCREEN.WAIT

LOCAL &page
&page=0.

WHILE &page<8.
(
    WinPRT myWIN                ;print the lines displayed in the window
                                ;named myWIN
    WinPAN 0. 10. myWIN         ;scroll down 10 lines in the window
    SCREEN.WAIT

    &page=&page+1.
)

```

See also

- [WinPrint](#) ■ [Win](#) ■ [PRinTer](#) ■ [PRinTer.HardCopy](#)
- [WINDOW.NAME\(\)](#)
- ▲ ['Printer Operations' in 'PowerView User's Guide'](#)
- ▲ ['Window System' in 'PowerView User's Guide'](#)

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**.

Example:

```
WinResist.PEDIT test.cmm ;open PRACTICE script in a resistant window
```

See also

- [Win](#)
- [WinBack](#)
- [WinCLEAR](#)
- [WinFreeze](#)
- ▲ ['Window System' in 'PowerView User's Guide'](#)

Format: **WinRESIZE** [*<width>*] [*<height>*] [**WinTOP** | *<window_name>*]

Resizes the uppermost window or the window that has the specified *<window_name>*.

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 *<window_name>* does not exist.

WinTOP (or TOP as an alias)	Resizes the uppermost window.
<i><window_name></i>	Window names are case-sensitive. Use WinPOS to assign a user-defined name and an initial size to a window.

Example: In this script, the command **WinPOS** is used to open a window with a user-defined size and name. If the named window is already open, **WinRESIZE** is used to re-apply the user-defined size. In addition, the named window is displayed on top of all other windows.

```
;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
)
;bring the named window to the top of the display hierarchy
WinTOP myWin01
```

See also

- [Win](#)
- [WinDefaultSize](#)
- [WinPOS](#)
- [WinTOP](#)
- [WINDOW.EXIST\(\)](#)
- ▲ ['Window System' in 'PowerView User's Guide'](#)

Format: **WinSmall.***<command>*

Pre-command for generating a window with small font. For an example, see [WinLarge](#).

See also

- [Win](#)
- [WinLarge](#)
- [WinMid](#)
- ▲ ['Window System' in 'PowerView User's Guide'](#)

WinTABS

Specify widths of re-sizable columns

Format: **WinTABS** *<col1_width>* [*<col2_width>...*]

TRACE32 PowerView 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 to specify the width of re-sizable columns for the next window that will be opened.

Examples:

```
WinTABS 20. 5. 20. 40. ;specify the width of the columns code, label,
                        ;mnemonic, and comment in a List.Mix window
```

```
List.Mix
```

```
WinTABS 50. 20. ;specify the width of the columns tree and
                ;InternalBAR.Log in a Trace.STATistic.TREE
                ;window
```

```
Trace.STATistic.TREE
```

See also

- [Win](#)
- [WinPOS](#)
- ▲ ['Window System' in 'PowerView User's Guide'](#)

Format: **WinTOP** [<window_name>]

Brings the named window to the top of the display hierarchy. You can now see that focus is on the window. 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\(\)](#).

<window_name>

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 script, a custom dialog with the user-defined name `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.

```
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
  (
    HEADER "MyDialog"
    POS 0. 0. 30. 1.
    TEXT "A named dialog window"
    BUTTON "Close" "DIALOG.End"
  )
)
ENDDO
```

See also

■ [Win](#)

■ [WinPOS](#)

■ [WinRESIZE](#)

□ [WINdow.EXIST\(\)](#)

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

Format: **WinTrans.***<command>*

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.

Example:

```
WinExt.WinTrans.Register.view ; open a transparent Register.view window  
                               ; while TRACE32 is in MDI window mode
```

See also

■ [Win](#)

▲ ['Window System' in 'PowerView User's Guide'](#)

Appendix A - Help Filters

The following help filters are available for the **HELP.FILTER** command group:

- [Help Filters for TRACE32 Hardware/Software](#)
- [Help Filters for OS Awareness Manuals](#)
- [Help Filters for Third-Party Integrations](#)
- [Help Filters for UEFI Debuggers](#)
- [Help Filters for Debug Back-Ends](#)

Help Filters for TRACE32 Hardware/Software

Filter	TRACE32 Hardware/Software
bdm*	TRACE32 debugger e.g. bdmarm, bdmsh4
esi	TRACE32 ERPOM simulator
gdb*	TRACE32 GDB Front-end e.g. gdbarm, gdbi386
icr*	TRACE32 real-time trace e.g. icretm, icrsh4
icrstm	TRACE32 CombiProbe
mon*	TRACE32 ROM monitor e.g. mon68k, mon166
nat386	Windows native process debugger
nexus*	TRACE32 NEXUS debugger e.g. nexusppc, nexusmac
pdg*	TRACE32 pdg Front-end e.g. pdgarm
pi	PowerIntegrator
pp	PowerProbe
sim*	TRACE32 Instruction Set Simulator or TRACE32 Front-end e.g. simarm, simpcc
stg	Stimuli generator
time	TRACE32 timing analyzer
tp	Trigger probe

Filter	OS Awareness
rtos*	All OS Awareness Manuals. The asterisk can be replaced with the suffixes listed below.
rtosamx	AMX
rtosartk	ARTK
rtosartx166	ARTX-166
rtosbios	DSP/BIOS
rtoschibios	ChibiOS/RT
rtoschorus	Chorus Classic and Chorus Micro
rtoscmicro	Cmicro
rtoscmx	CMX and CMX-TINY+
rtosecos	eCos
rtosembos	embOS
rtosepoc	Symbian OS EKA1
rtosfamos	FAMOS
rtosfreertos	FreeRTOS
rtoshi7000	HI7000
rtoshios	HIOS
rtoslinux	Linux
rtoslynxos	LynxOS
rtosmqx	MQX
rtosmtos	MTOS-UX
rtosnetbsd	NetBSD
rtosnorti	NORTi
rtosnucleus	Nucleus PLUS
rtosokl4	OKL4
rtosorti	OSEK/ORTI
rtosorti	OSEK/ORTI
rtosos21	OS21
rtosos9	OS-9
rtososeb	OSE Epsilon

Filter	OS Awareness
rtososec	OSE Classic
rtososeck	OSEck
rtososed	OSE Delta
rtososee	OSE Epsilon
rtospikeos	PikeOS
rtosprkernel	PrKERNEL
rtospsos	pSOS+
rtospxros	PXROS
rtosqnx	QNX
rtosquadros	RTXC Quadros
rtosrealos	REALOS
rtosrt7700	RTOS/7700
rtosrtc	RealTimeCraft
rtosrtems	RTEMS
rtosrtx166	RTX166 and RTX166 tiny
rtosrtx51	RTX51and RTX51 tiny
rtosrtxarm	RTX-ARM
rtosrtxc	RTXC
rtosrubus	Rubus OS
rtossciopta	Sciopta
rtossmx	SMX
rtossymbian2	Symbian OS EKA2
rtossysbios	SYS/BIOS
rtosthreadx	ThreadX
rtosuc3cmp	MicroC3/Compact
rtosuc3std	MicroC3/Standard
rtosuclinux	uClinux
rtosucos, rtosucos3	MicroC/OS-II MicroC/OS-III
rtosuiplus	uiPLUS
rtosuitron	uiTRON

Filter	OS Awareness
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

Filter	Third-Party Tool
intcodeblock	CodeBlocks
intcw	CodeWright
inteasy	EasyCase
intclipse	Eclipse
intexdi2	Windows CE Platform Builder
intlabview	LabView
intose	OSE Illuminator
intrhapsody	Rhapsody in MicroC
intrhapsodycpp	Rhapsody in C/C++
intsimulink	Simulink
inttornado	Tornado I
intxtools	X-Tools and X32

Help Filters for UEFI Debuggers

Filter	UEFI Debuggers
uefibldk	UEFI Awareness for BLDK
uefih2o	UEFI Awareness for H2O
uefitiano	UEFI Awareness for TianoCore

Help Filters for Debug Back-Ends

Filter	Debug Back-Ends
back*	Debug Back-Ends
backgtl	GTL Debug Back-End
backxcp	XCP Debug Back-End