LAUTERBACH
DEVELOPMENT TOOLS

PowerView User's Guide

Release 02.2023

MANUAL

Version 10-Feb-2023

Structure and Contents of the Documentation

This chapter describes the structure of the TRACE32 documentation.

The release history in the documentation always lists the latest changes in the TRACE32 software. When you get a new version of the TRACE32 software, please always check the **Release history** first.

Online Documentation

There are several ways to get access to the documentation:

1. If the TRACE32 software is already running, you can use the **Help** command in the main menu bar.

2. On the TRACE32 software DVD and in your TRACE32 system path (e.g. `C:\T32`), you can find a directory `pdf`. This directory contains the complete TRACE32 documentation in PDF format.

   Open `directory.pdf` to get the table of contents for the complete TRACE32 documentation.

Documentation on how to use the online help can be found in chapter **Help System**.

The documentation is *automatically* filtered by your currently used hardware and/or software configuration. The filter automatically reduces the whole documentation to the part that is relevant for you. If you want to change the filter, take a look at the command **HELP.FILTER**.

TRACE32-ICD includes all debuggers based on an on-chip debug interface (e.g. JTAG, BMD, OCDS …) as well as ROM monitor solutions. Lauterbach also provides a trace extension for most debuggers (TRACE32-ICT). TRACE32-ICD comes with a number of manuals that should make you familiar with important features of TRACE32-ICD.

Manuals to help you get started:

- **“ICD Tutorial” (icd_tutorial.pdf)**

  A guided tour through the TRACE32 graphical user interface (GUI) called TRACE32 PowerView. We use a simple program example in C to illustrate the most important debug features and give lots of helpful tips & tricks for everyday use.

- **“Training Basic Debugging” (training_debugger.pdf)** - An introduction to debugging with TRACE32

- **“Training Basic SMP Debugging”** (training_debugger_smp.pdf) - An introduction to SMP debugging

- **“Training Script Language PRACTICE” (training_practice.pdf)** - An introduction to PRACTICE, the scripting language for TRACE32

Sources of information beyond the PDF files of the TRACE32 online help:

- **https://www.lauterbach.com/reference_card_web.pdf** - Reference Card for the most common commands of the PRACTICE scripting language

- **https://www.lauterbach.com/publications/debugging_amp_smp_systems.pdf** - An introduction to asymmetrical and symmetrical multiprocessing (AMP/SMP)

- **https://www.youtube.com/user/lauterbachgmbh** - A variety of tutorials on the Lauterbach YouTube channel

For more information on the features of TRACE32-ICD, refer to the following parts of the TRACE32 online help:

- **“TRACE32 Installation Guide” (installation.pdf)**

  This part is the general installation guide for all TRACE32 development tools.

- **“ICD In-Circuit Debugger”**

  This part provides all CPU specific information for your TRACE32-ICD, chiefly how to set up the debugger for your target. Here you will also find all extra features that are supported for your CPU.

- **“General Reference Guide” (general_ref_*<x>*.pdf)**

  This part provides an alphabetical list of all debugger commands.

- **“TRACE32 Functions Reference” (*<x>*_func.pdf)**

  Refer to this part for information about the TRACE32 PRACTICE functions.

- **“PowerView User’s Guide” (ide_user.pdf)**

  All TRACE32 development tools share the common user interface TRACE32 PowerView. This part describes the basic functions of the user interface (command structure, online help, editing and managing files, printer operations, etc.)

- **“PowerView Command Reference” (ide_ref.pdf)**

  This part provides an alphabetical list of all TRACE32 PowerView commands.

- **“PRACTICE Script Language User’s Guide” (practice_user.pdf)**

  The TRACE32 script language PRACTICE is mainly used to perform automatic setups, to automate test sequences or to store the system settings for later recall. This part describes the basic structure and features of PRACTICE.

- **“PRACTICE Script Language Reference Guide” (practice_ref.pdf)**

  This part provides an alphabetical list of all PRACTICE commands.

- **“OS Awareness Manuals” (rtos_*<os>*.pdf)**

  Refer to this part if you want to use the TRACE32 OS Awarenesses (= RTOS Debuggers in previous TRACE32 releases).

- **“3rd-Party Tool Integrations” (int_*<x>*.pdf)**

  Refer to this part, if you want to run TRACE32-ICD from a 3rd-party user interface.

Program Start

After installing the driver program to the appropriate host system, the executable can be started.

The TRACE32 system has to be powered up. If this is not the case, the error message "NO CARRIER …", "LINK ERROR …" or "TRACE32 not responding" will appear.

If all environment variables are installed correctly, the driver program can be invoked from any sub-directory or drive.

To start a TRACE32 executable, you can use:

- The T32Start utility
- The command line of the operating system

T32Start

The user interface of the T32Start utility assists you in creating as many start environments for TRACE32 as you need for your different debug projects. Based on the start environment you have created with a few mouse-clicks, T32Start auto-generates the configuration file that is essential for starting TRACE32 correctly.

For more information, see **“T32Start”** (app_t32start.pdf).

Command Line

If you want to start TRACE32 via the command line of the operating system, you need to manually create the configuration file (by default config.t32). The configuration file settings are described in **“TRACE32 Installation Guide”** (installation.pdf).

For information about the command line syntax and command line options, see **“TRACE32 Installation Guide”** (installation.pdf).

The following list is a selection of the available command line options:

- **--t32-help**
- **--t32-safestart**
- **--t32-logautostart**

In-Circuit Debugger TRACE32-ICD

t32m*<cpu>*.exe	Windows version for TRACE32-ICD. TRACE32-ICD system software is running on PC.
t32m*<cpu>*	Workstation version for TRACE32-ICD. TRACE32-ICD system software is running on workstation.

Program End

Getting back to the operating system command level is possible by using the command **QUIT** or by choosing **File** menu > **Exit**.

QUIT	Return to operating system

```
::QUIT
```

The **QUIT** command quits the driver program and resets the TRACE32 system. When the driver program is restarted, a complete boot sequence will be executed.

If for any reason the host crashes, the TRACE32 system should be switched off for a few seconds.

NOTE:	If your TRACE32 development tool is connected to the target, it is important to use the proper power on/power off sequence. For detailed information, refer to your **Processor Architecture Manual**.

PowerView - Screen Display

Concept

The graphical user interface (GUI) of TRACE32 is called TRACE32 PowerView. The TRACE32 user interface is based on an extremely fast, character oriented window system. Up to 128 different windows can be composed for display, each can contain up to 250 * 250 characters. Window type, size and status can be defined very flexibly by the user. Each window is assigned to one task, which is sequentially executed to update the window information.

Windows may be frozen to prevent them from being updated.

An array of windows is called a “PAGE”. Several pages can be defined in this manner, with each page representing a part of the user’s work area. Multiple pages cause no performance degradation, as only the visible windows are updated.

Graphical User Interface - Window Modes

The user interface TRACE32 PowerView supports 2 different window modes:

- **MDI** (multiple document interface): All sub-windows are placed inside the TRACE32 main window.
- **MWI** (multiple window interface): The TRACE32 main window and the sub-windows are placed freely on the desktop.

On MS Windows systems, the MWI window mode is split into 2 sub-modes:

- **FDI** (floating document interface): Same as MWI; the taskbar shows only one icon for all windows. Minimizing the main window will also minimize the sub-windows.
- **MTI** (multiple top-level window interface): The taskbar shows an icon for the main window and each sub-window. Minimizing the main window does not minimize the sub-windows.

These modes can be set in the `SCREEN=` section of the configuration file (config.t32). Depending on the version of TRACE32, not all window modes are supported:

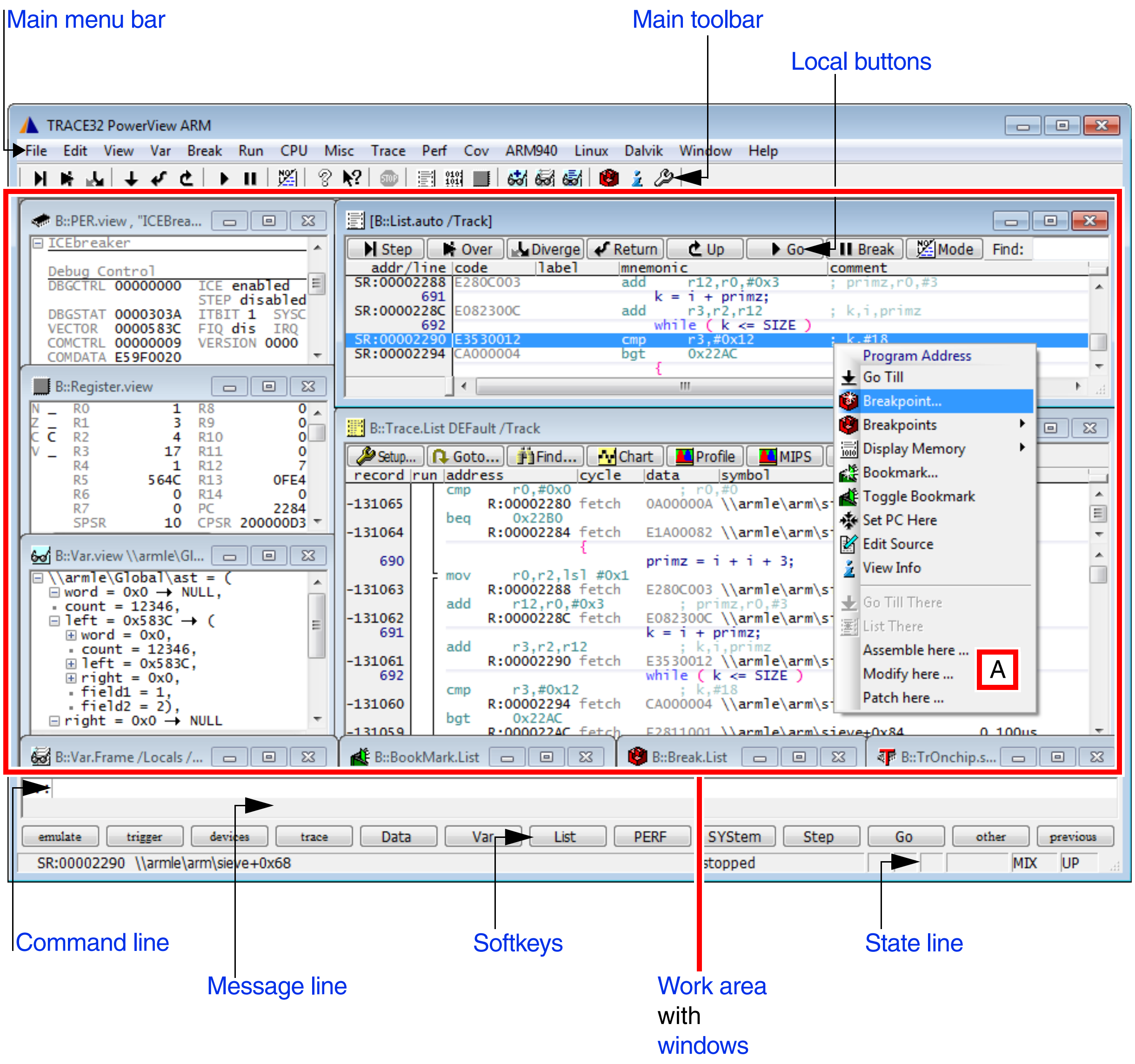
	Windows	**Linux/ Motif**	**Linux/ Qt**	**HP-UX**	**OS X/ Motif**	**OS X/ Qt**
MDI	+	-	+	-	-	+
MWI	+	+	+	+	+	+

MDI User Interface

[Back to Top]

After starting TRACE32, the main window of the TRACE32 PowerView GUI is displayed.

For more information, click the blue GUI terms.



```
ICEbreaker
 Debug Control
 DBGCTRL  00000000  ICE  enabled
                    STEP disabled
 DBGSTAT  0000303A  ITBIT 1   SYSC
 VECTOR   0000583C  FIQ dis   IRQ
 COMCTRL  00000009  VERSION  0000
 COMDATA  E59F0020
```

```
addr/line       code      label  mnemonic              comment
SR:00002288     E280C003         add    r12,r0,#0x3    ; primz,r0,#3
        691                          k = i + primz;
SR:0000228C     E082300C         add    r3,r2,r12      ; k,i,primz
        692                          while ( k <= SIZE )
SR:00002290     E3530012         cmp    r3,#0x12       ; k,#18
SR:00002294     CA000004         bgt    0x22AC
                                     {
```

```
N _  R0       1  R8       0
Z _  R1       3  R9       0
C C  R2       4  R10      0
V _  R3      17  R11      0
     R4       1  R12      7
     R5    564C  R13   0FE4
     R6       0  R14      0
     R7       0  PC    2284
     SPSR    10  CPSR 200000D3
```

```
record  run address          cycle  data       symbol
                cmp   r0,#0x0                  ; r0,#0
-131065         R:00002280 fetch   0A00000A   \\armle\arm\s
                beq   0x22B0
-131064         R:00002284 fetch   E1A00082   \\armle\arm\s
                           {
    690                            primz = i + i + 3;
                mov   r0,r2,lsl #0x1
-131063         R:00002288 fetch   E280C003   \\armle\arm\s
                add   r12,r0,#0x3              ; primz,r0,#3
-131062         R:0000228C fetch   E082300C   \\armle\arm\s
    691                            k = i + primz;
                add   r3,r2,r12                ; k,i,primz
-131061         R:00002290 fetch   E3530012   \\armle\arm\s
    692                            while ( k <= SIZE )
                cmp   r3,#0x12                 ; k,#18
-131060         R:00002294 fetch   CA000004   \\armle\arm\s
                bgt   0x22AC
-131059         R:000022AC fetch   E2811001   \\armle\arm\sieve+0x84     0.100us
```

```
\\armle\Global\ast = (
  word = 0x0 → NULL,
  count = 12346,
  left = 0x583C → (
    word = 0x0,
    count = 12346,
    left = 0x583C,
    right = 0x0,
    field1 = 1,
    field2 = 2),
  right = 0x0 → NULL
```

```
SR:00002290  \\armle\arm\sieve+0x68      stopped
```

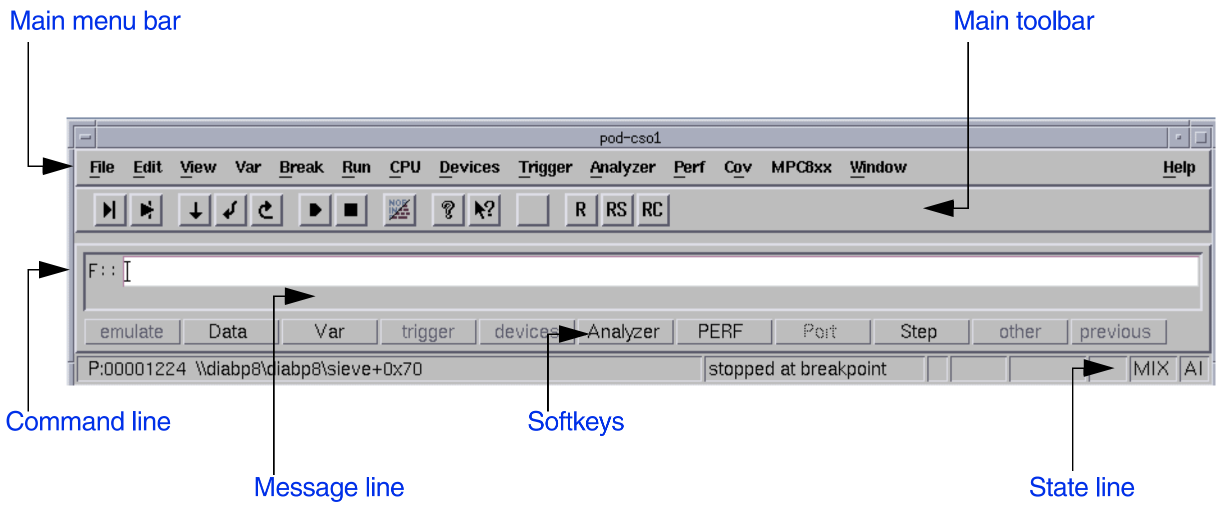
A Local popup menu

MWI User Interface

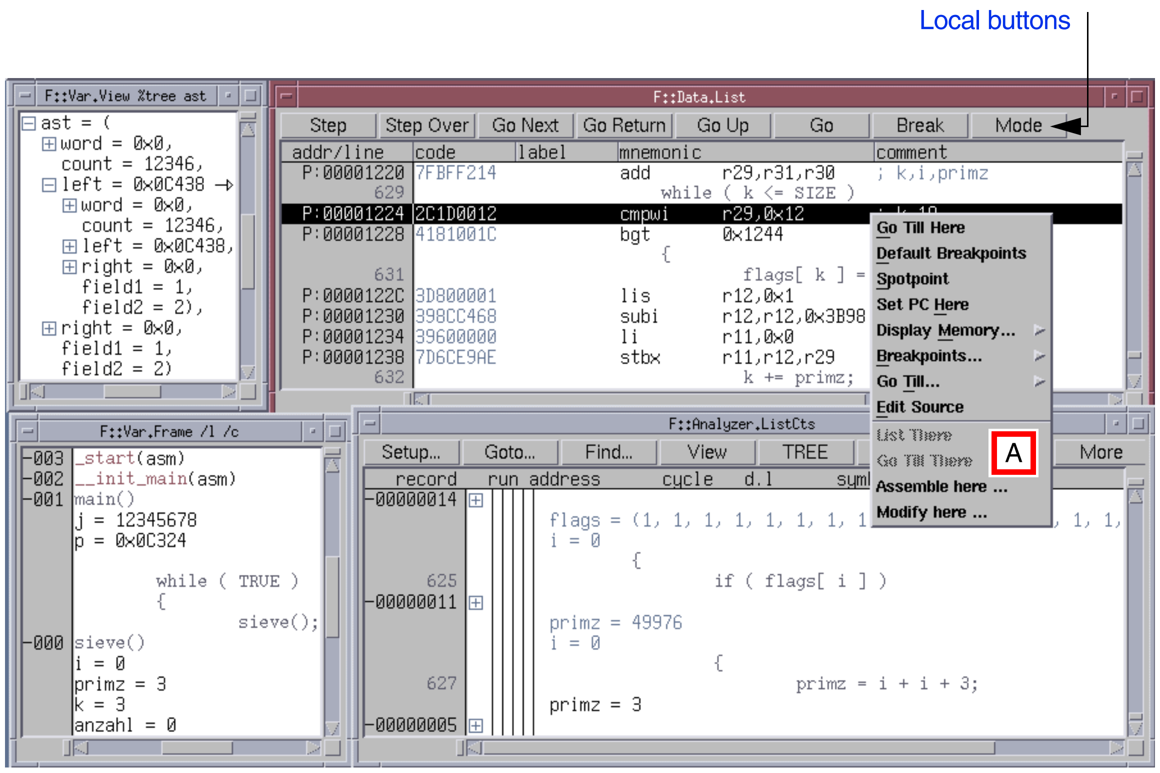
[Back to Top]

After starting TRACE32, the main window of the TRACE32 PowerView GUI is displayed.

For more information, click the blue GUI terms.


Main menu bar
Main toolbar
pod-cso1
File Edit View Var Break Run CPU Devices Trigger Analyzer Perf Cov MPC8xx Window Help
R RS RC
emulate Data Var trigger devices Analyzer PERF Port Step other previous
Command line
Softkeys
Message line
State line

```
F::
P:00001224 \\diabp8\diabp8\sieve+0x70     stopped at breakpoint     MIX AI
```



```
ast = (
  word = 0x0,
  count = 12346,
  left = 0x0C438 ->
    word = 0x0,
    count = 12346,
    left = 0x0C438,
    right = 0x0,
    field1 = 1,
    field2 = 2),
  right = 0x0,
  field1 = 1,
  field2 = 2)
```

```
addr/line   code      label  mnemonic               comment
P:00001220  7FBFF214         add     r29,r31,r30    ; k,i,primz
       629                         while ( k <= SIZE )
P:00001224  2C1D0012         cmpwi   r29,0x12
P:00001228  4181001C         bgt     0x1244
                                   {
       631                                 flags[ k ] =
P:0000122C  3D800001         lis     r12,0x1
P:00001230  398CC468         subi    r12,r12,0x3B98
P:00001234  39600000         li      r11,0x0
P:00001238  7D6CE9AE         stbx    r11,r12,r29
       632                                 k += primz;
```

```
-003 _start(asm)
-002 __init_main(asm)
-001 main()
     j = 12345678
     p = 0x0C324

            while ( TRUE )
            {
                    sieve();
-000 sieve()
     i = 0
     primz = 3
     k = 3
     anzahl = 0
```

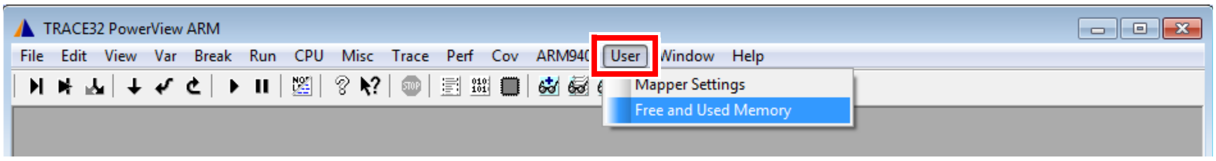
```
record     run address   cycle   d.l   sym
-00000014                 flags = (1, 1, 1, 1, 1, 1, 1, 1 ... 1, 1,
                          i = 0
                              {
                                   if ( flags[ i ] )
      625
-00000011                 primz = 49976
                          i = 0
                                        {
      627                                        primz = i + i + 3;
                          primz = 3
-00000005
```

A Local popup menu

Main Menu Bar

[Back to Top]

The main menu bar provides all important commands for each functional unit of the TRACE32 development tool. You can add user-defined menus to the main menu bar by using the **MENU** commands.


TRACE32 PowerView ARM
File Edit View Var Break Run CPU Misc Trace Perf Cov ARM940 User Window Help
Mapper Settings
Free and Used Memory

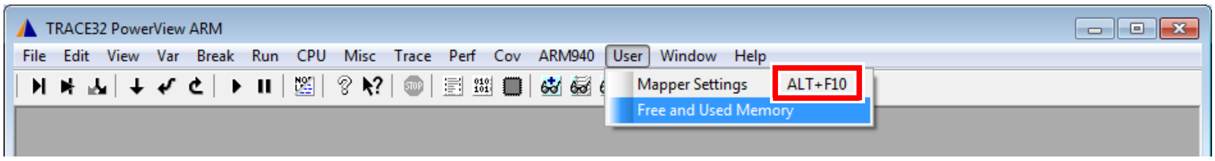
MENU.AddMenu	Allows you to quickly add one menu for temporary usage. Default name of the temporary menu is **User**.
MENU.ReProgram	Allows you to embed a menu definition in a PRACTICE script (*.cmm) or create a *.men file for a menu definition.

Example: This script adds the **User** menu shown in the above screenshot to the main toolbar.

```
; menu User with two menu options
MENU.AddMenu "Mapper Settings"       "MAP.List"
MENU.AddMenu "Free and Used Memory" "MAP.state"
```

Accelerators

Accelerators allow you to execute commands with a single keystroke. Usually the function keys are used for this purpose. Accelerators can be changed by using the **MENU** commands.


TRACE32 PowerView ARM
File Edit View Var Break Run CPU Misc Trace Perf Cov ARM940 User Window Help
Mapper Settings ALT+F10
Free and Used Memory

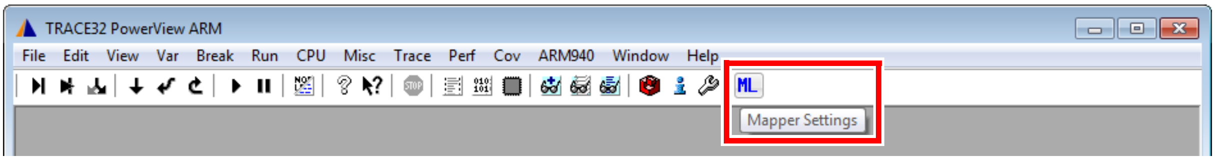
Example:

```
; the example shows how to include an accelerator in a temporary menu
MENU.AddMenu "Mapper Settings, ALT+F10" "MAP.List"
```

Main Toolbar

[Back to Top]

The main toolbar provides buttons for the most important TRACE32 commands. You can add user-defined buttons with tooltips to the main toolbar by using the **MENU** commands.


TRACE32 PowerView ARM
File Edit View Var Break Run CPU Misc Trace Perf Cov ARM940 Window Help
ML
Mapper Settings

MENU.AddTool	Add a temporary button to the main toolbar, i.e. the button is available only for the current TRACE32 session
TOOLBAR	Toggle main toolbar

MENU.Program	Editor to write a program that customizes the TRACE32 menu
MENU.ReProgram	Menu programming
MENU.RESet	Restore default menu and configuration of main toolbar

Example: This script adds the button shown in the above screenshot to the main toolbar.

```
; the example shows how to add a temporary button to the main toolbar
;                   <tooltip>        <button_letters,color>    <command>
MENU.AddTool "Mapper Settings"          "ML,B"                 "MAP.List"
```

Work Area

[Back to Top]

The work area is used as the general input and output area. For more detailed information, see **Windows**.

In addition to working with windows in the work area, you can place windows on *user-defined* pages. This is useful if you need to open lots of windows and want to group them. For more information, see **Pages**.

Message Line

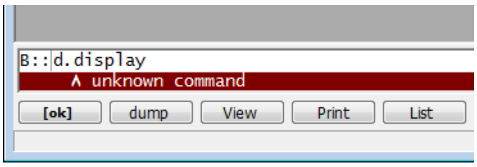
[Back to Top]

The message line displays error and general messages, information on cursor position, etc. The message line is located below the command line.

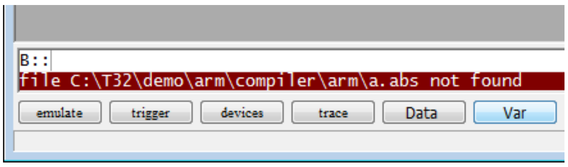
Error Messages

Error messages are displayed by a special attribute (e.g. red or blinking). The error message is erased automatically. If an input error was made, an arrow will point to the mistake on the command line.

⚠	The softkeys will no longer correspond to the entered data! If the error message is still unclear, the appropriate page in the on-line manual will be displayed, when using the «help» key.



```
B::d.display
     ∧ unknown command
```



```
B::
file C:\T32\demo\arm\compiler\arm\a.abs not found
```

General Messages

When entering configuration commands, the current state is displayed during the command input. Some command outputs are also displayed in the message line.

```
B::TRANSlation.TableWalk
Address translation: OFF
```

Additional Information on Cursor Position

If the left mouse button is pressed down while the cursor is positioned within a window, additional information in regard to the current context will be displayed. In the example below the variable flags is selected in the **Data.List** window.

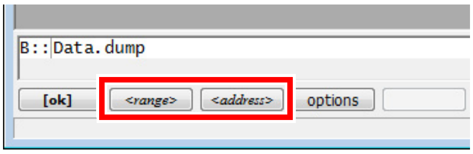
```
B::
flag = {1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1}
```

Softkeys

[Back to Top]

The softkey structure represents a hierarchical selection menu. Each softkey can be activated by clicking the left mouse button.

Softkeys with **pointed brackets** (e.g., «*<file>*, *<range>*, *<address>*») are placeholders for parameters which have to be entered in the command line.



In the case of softkeys with **square brackets** ([or]) the command is executed immediately after being selected without a written entry to the command line.

Softkeys written completely in **lower case characters** represent command hierarchy branching which does not alter the command line (e.g., emulation).

Softkeys written in **upper case** and **mixed case** represent command words which can also be entered via the keyboard. You can enter either the entire word, or just the upper case letters. Upper and lower case characters are not differentiated.

By means of the «**other**» softkey additional menu selections located in the same hierarchical level can be started. By «**previous**» you can return to the former level in the menu hierarchy. The commands for those softkeys which have been shadowed in on the display are inaccessible at this time.

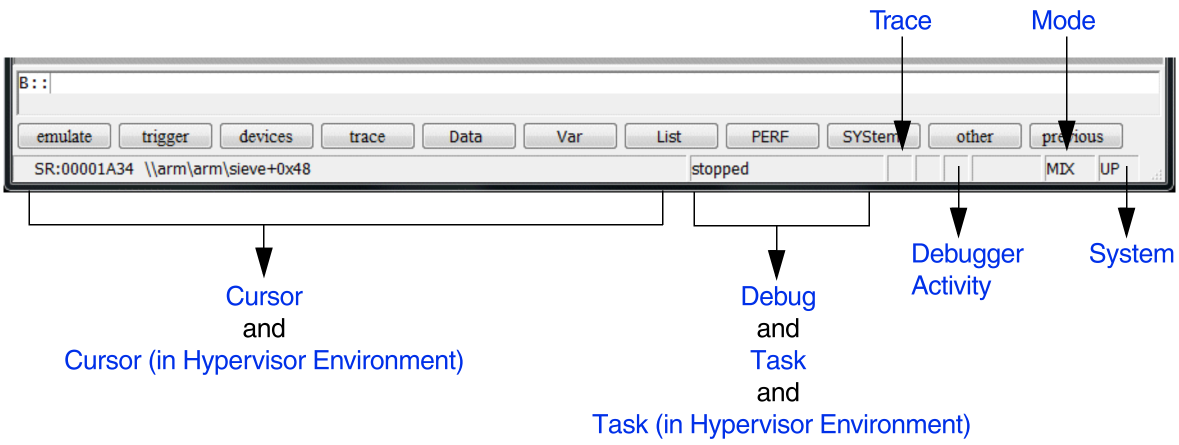
`Data`	`Command`
`emulate`	`Command path`
`[Step]`	`Direct command`
`<address>`	`Parameter`
`previous`	`Previous menu`
`other`	`Next menu`

State Line

[Back to Top]

The state line is located at the bottom of the TRACE32 main window.

For more information about the individual fields in the state line, click the blue GUI terms.



Cursor

[Back to Top]

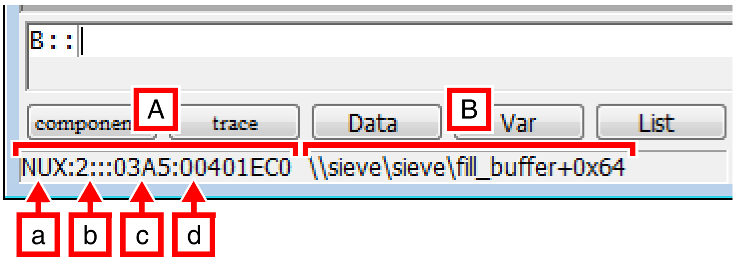
The **Cursor** field provides:

- Boot information (Booting …, Initializing … etc.).
- Information on the item selected by the cursor, such as:
  - Address (e.g. `SR:00001A34`) and symbol (e.g. `\\arm\arm\sieve+0x48`)
  - File name, offset, line number, column number

Cursor - In a Hypervisor Environment

[Back to Top]

In a hypervisor environment, the **Cursor** field provides the following information:



NUX:2:::03A5:00401EC0 \\sieve\sieve\fill_buffer+0x64

A Example of a fully qualified address in a hypervisor environment

- **a** Access class (`NUX:`)
- **b** Machine ID (`2:::`)
- **c** Space ID (`03A5:`)
- **d** Logical address (`00401EC0`)

B Symbol (`\\sieve\sieve\fill_buffer+0x64`)

The machine ID [**b**] is displayed only if you set **SYStem.Option.MACHINESPACES** to **ON**, and the space ID [**c**] is displayed only if you set **SYStem.Option.MMUSPACES** to **ON**.

Debug

[Back to Top]

The **Debug** field provides:

- Information on the debug communication (system down, system ready etc.)
- Information on the state of the debugger (running, stopped, stopped at breakpoint etc.)

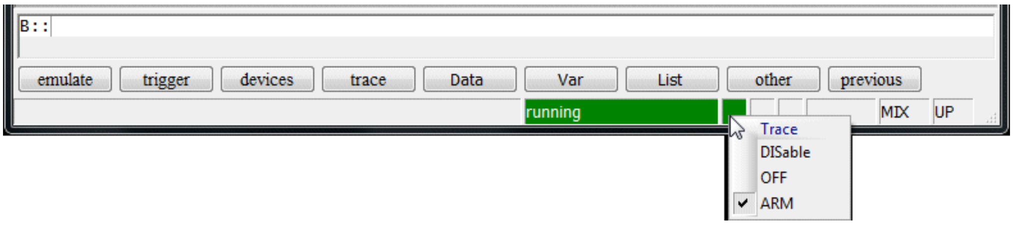
Trace

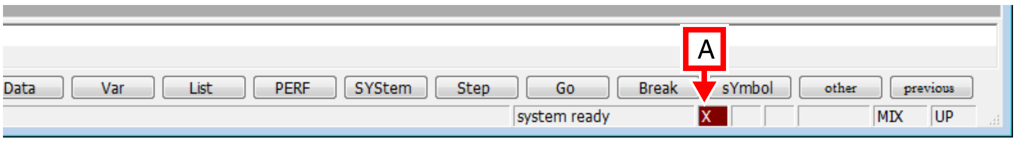
[Back to Top]

The **Trace** field provides:

- Information on the state of the trace (**DISable**, **OFF**, **ARM** …).

The state of the trace can be changed by using the **Trace** pull-down.





A A white X against a red background indicates that the trace method is set to **NONE**. For more information, see **<trace>.METHOD NONE**.

Debugger Activity

[Back to Top]

The **Debugger Activity** field provides information on the target activity of the debugger, for example:

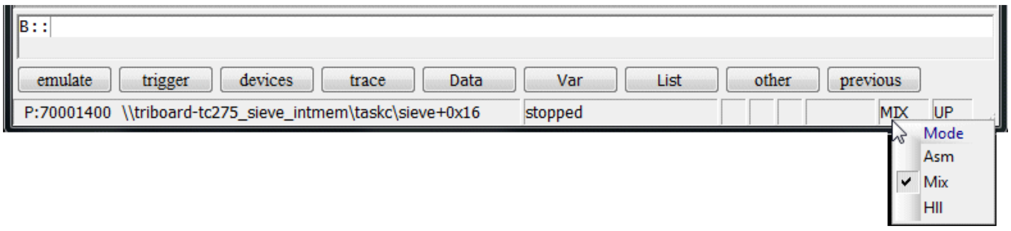
- A red S indicates that the debugger shortly interrupts the program execution to realize a debugger feature, e.g. **intrusive breakpoints**.
- RUN in green indicates that TRACE32 has started an algorithm on the target to realize a debugger feature, e.g. target-controlled FLASH programming.

Mode

[Back to Top]

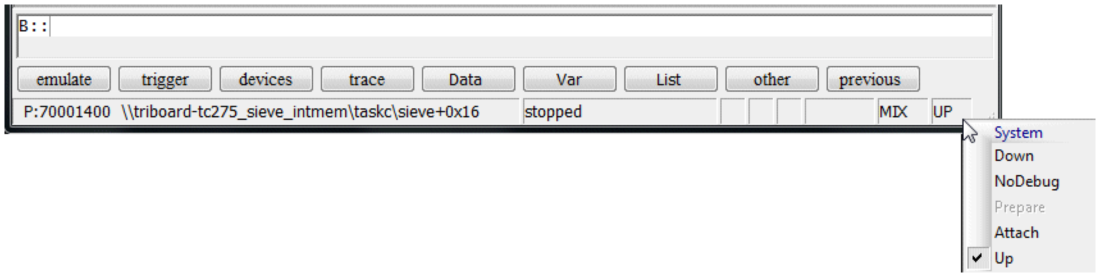
The **Mode** field indicates the debug mode. The debug mode defines how source code information is displayed (assembler code ASM or programming language code HLL or a mixture of both MIX) and how single stepping is performed (assembler line-wise or programming language line-wise).

The debug mode can be changed by using the **Mode** pull-down.



The **System** field indicates **Up** if the communication between the debugger and the processor/core is established and nothing is otherwise.

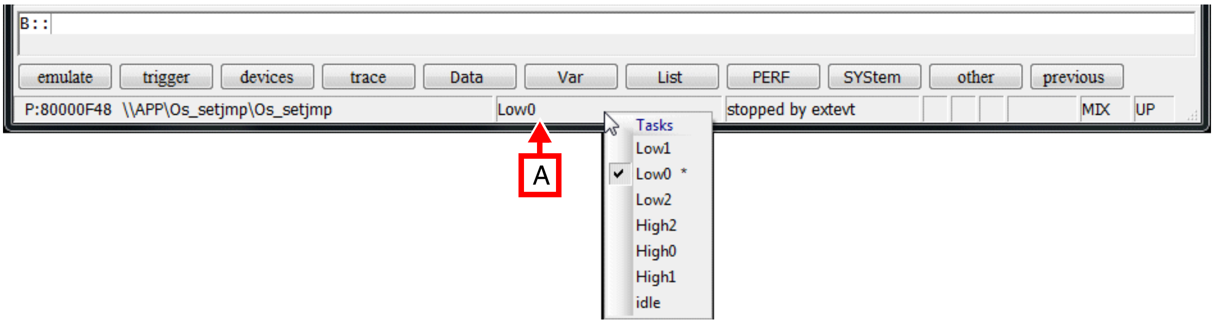
The communication between the debugger and the processor/core can be established and ended by the **System** pull-down.



Task

[Back to Top]

The name of the current task is displayed in the **Task** field after the TRACE32 OS Awareness was activated, see [**A**].



Selecting another task from the **Task** pull-down allows to switch the task context (mainly **Register.view** window and **Frame.view** window).

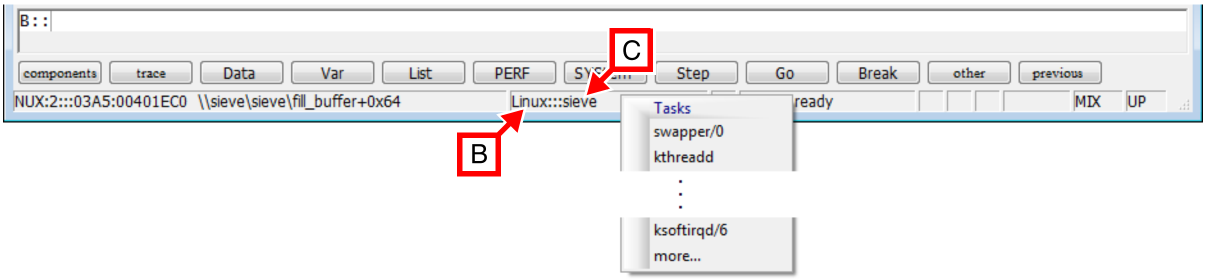
- A check mark is used to mark the task for which the task context is displayed.
- A asterisk is used to mark the currently active task.

This feature is not supported for all operating systems.

Task - In a Hypervisor Environment

[Back to Top]

In a hypervisor environment, the machine name precedes the task name, and the three colons `:::` serve as the separator between machine name [**B**] and task name [**C**].



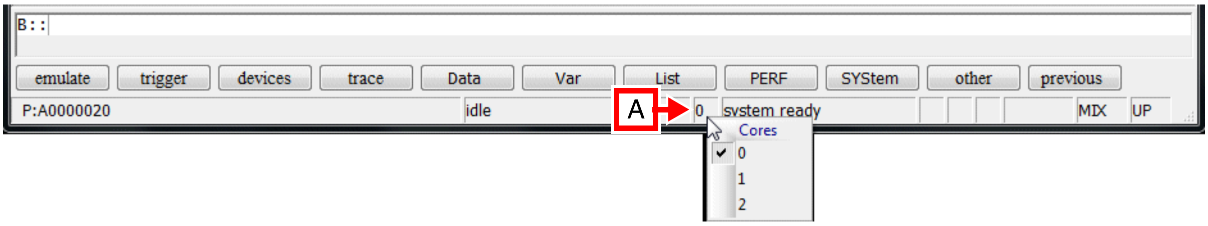
SMP Systems

[Back to Top]

The **Cores** field shows the currently selected core [**A**].

- TRACE32 PowerView visualizes all system information from the perspective of the selected core if not specified otherwise.

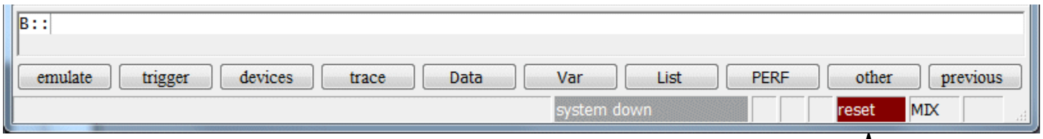
The **Cores** pull-down allows to change the selected core.



Advanced

[Back to Top]

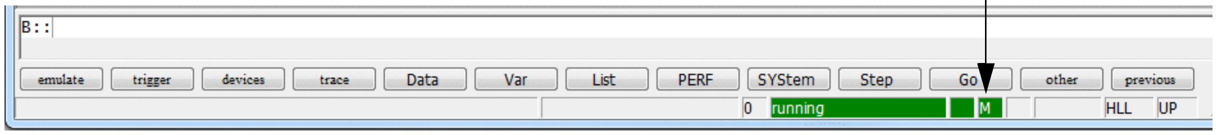
The **Target** field indicates an active target reset or a locked JTAG interface (command: **SYStem.LOCK ON**).



Target

If “Integrated Run & Stop Mode Debugging via JTAG” is used TRACE32 indicates that a debug agent is running in the Monitor field. For details refer to **“Run Mode Debugging Manual Linux”** (rtos_linux_run.pdf).

Monitor



Show/Hide State Line

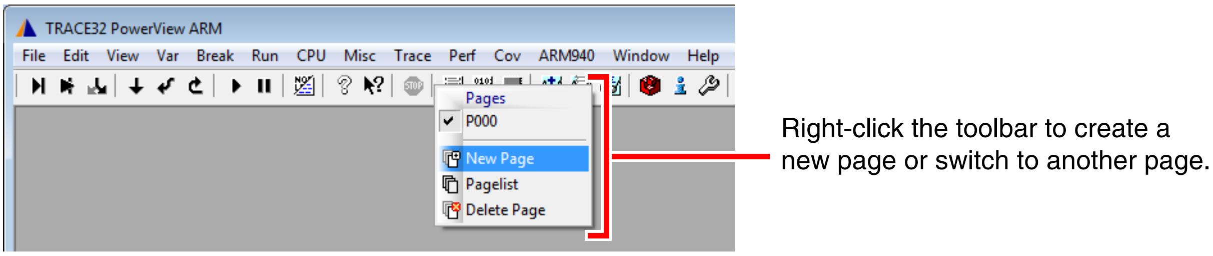
[Back to Top]

STATUSBAR ON	Show state line.
STATUSBAR OFF	Hide state line.

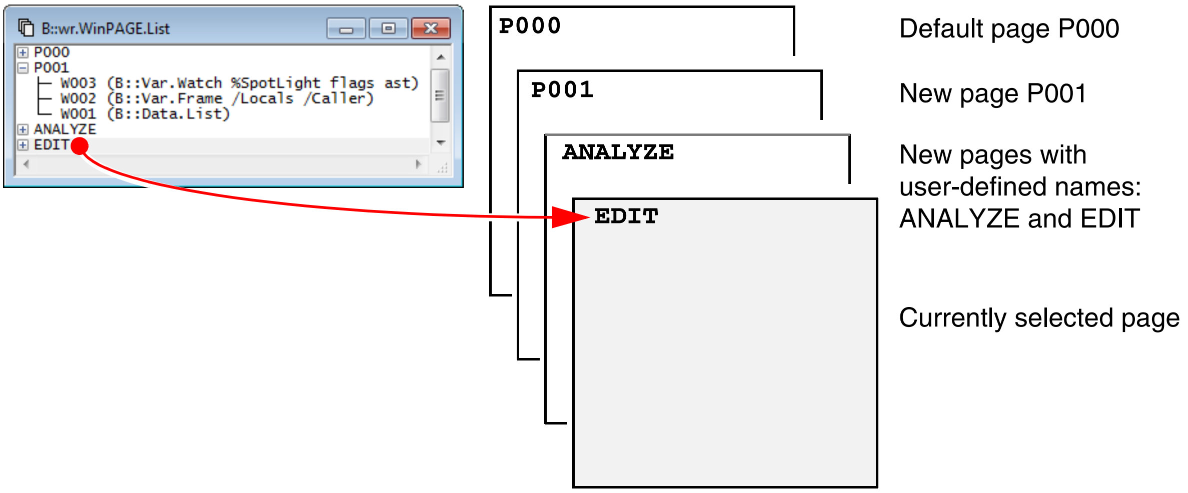
Window Pages

Window pages in TRACE32 are similar to workspaces in other applications. In TRACE32, you can open windows on different pages, but only the windows on the selected page are visible. Windows located on the other pages are temporarily hidden.

You can create a new page and switch between pages by right-clicking anywhere on the TRACE32 main toolbar. By default, TRACE32 auto-increments the names of new pages P001, P002, etc. But you can also create new pages with user-defined names.


TRACE32 PowerView ARM
File Edit View Var Break Run CPU Misc Trace Perf Cov ARM940 Window Help
Pages
P000
New Page
Pagelist
Delete Page
Right-click the toolbar to create a new page or switch to another page.

The **WinPAGE.List** window serves as the table of contents for your pages. You should always open the **WinPAGE.List** window with the **WinResist** pre-command to keep the table of contents visible on all pages.



Example:

```
WinResist.WinPAGE.List ;keep the table of contents visible on all pages

WinPAGE.Create         ;new page with auto-incremented page name

                       ;open these windows on the new page
Data.List
Var.Frame /Locals /Caller
Var.Watch %SpotLight flags ast

WinPAGE.Create ANALYZE ;create a new page named ANALYZE
WinPAGE.Create EDIT    ;create a new page named EDIT
```

WinPAGE.select	Select page
WinPAGE.Create	Create page with a user-defined name
WinPAGE.Delete	Delete page
WinPAGE.List	List pages
WinPAGE.REName	Rename page
WinPAGE.RESet	Reset window system

Colors

SETUP.COLOR	Change colors
sYmbol.List.ColorDef	List keyword colors
sYmbol.ColorDef	Modify keyword colors
CmdPOS	Toolbar and/or background color for multicore debugging (TRACE32 is in MWI window mode)
FramePOS	Toolbar and/or background color for multicore debugging (TRACE32 is in MDI window mode)
CORE.SHOWACTIVE and **CORE.List**	Show active cores in an SMP system. Each core has its own color.
SETUP.COLORCORE	Enable coloring for core-specific info in SMP systems
<trace>.STATistic.COLOR	Assign colors to function for colored graphics
GROUP.COLOR	Define color for group indicator

How the TRACE32 PowerView GUI Assists You in Scripting

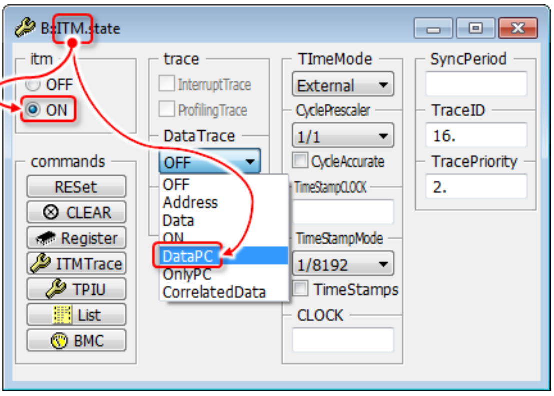
The TRACE32 PowerView GUI is designed to assist you in writing PRACTICE scripts (*.cmm), with which processes can be automated in TRACE32:

1. The GUI controls in TRACE32 windows are labeled such that they reveal the command syntax for use in a PRACTICE script. See (A) below.
2. The commands shown in window captions can be modified and re-used with one mouse-click. See (B) below.

(A) Writing Scripts based on the Text Labels of the TRACE32 PowerView GUI

Let's assume you are writing a PRACTICE script and require the configuration settings from a window, such as the **ITM.state** window. A window can contain all sorts of GUI controls: radio options, check boxes, drop-down lists, input boxes, and so on. To write a script that takes all of these GUI controls into account, follow these two simple rules:

1. Type the GUI labels into your script file.
2. Omit the GUI labels that are all lowercase (here: itm, trace, commands)



TRACE32 does not accept:
`ITM.itm.ON`

TRACE32 accepts these 2 solutions:
`ITM.ON`
`itm.on`

Resulting command syntax for use in a PRACTICE script:

Solution 1	**Solution 2**	**Solution 3**
`ITM.ON`	`itm.on`	`itm.on`
`ITM.DataTrace DataPC`	`itm.datatrace datapc`	`itm.dt   dpc`
`ITM.PCSampler OFF`	`itm.pcsampler off`	`itm.pcs  off`
`ITM.TImeMode  External`	`itm.timemode  external`	`itm.tim  e`
`ITM.CyclePrescaler 1/1`	`itm.cycleprescaler 1/1`	`itm.cp   1/1`
`ITM.TimeStampMode  ALL`	`itm.timestampmode  all`	`itm.tsm  all`
`ITM.TraceID        16.`	`itm.traceid        16.`	`itm.tid  16.`
`ITM.TracePriority   2.`	`itm.tracepriority   2.`	`itm.tp    2.`

Solution 2 is the recommended solution in terms of typing effort and source code maintainability - for you and your colleagues.

Solution 3 is very useful for frequently-used commands when you are working with the TRACE32 command line.

(B) Modifying and Re-using Commands Shown In Window Captions

Commands shown in window captions can easily be modified. This is a TRACE32 feature which is very useful if you want to add, remove, or change the options or parameters of a command. This feature is also useful when you are writing a PRACTICE script (*.cmm) and require a command that is already displayed in a window caption; there is no need to re-type the command.

If you want to reproduce the step-by-step procedure below, use this source code:

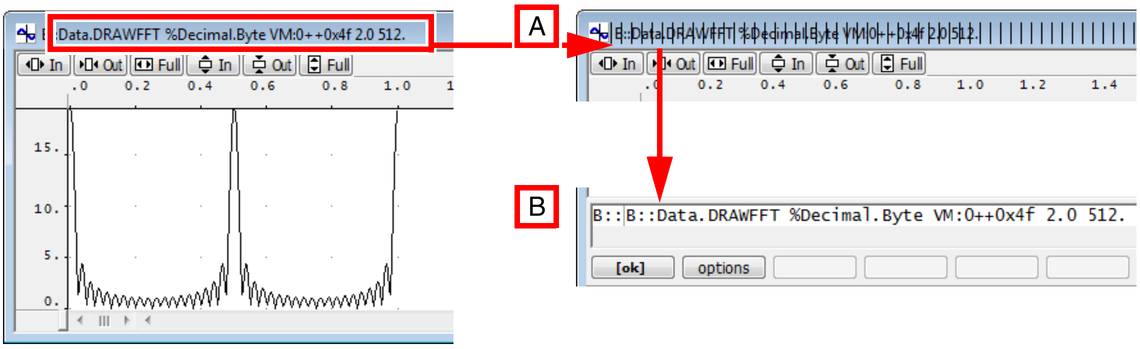
```
;set a test pattern to the virtual memory of TRACE32
Data.Set VM:0--0x4f %Byte 1 0 0 0

Data.dump VM:0x0  ;open the Data.dump window

;visualize the contents of the TRACE32 virtual memory as a graph
Data.DRAWFFT %Decimal.Byte VM:0++0x4f 2.0 512.
```

To modify / re-use commands shown in window captions:

1. As a Windows user, right-click the window caption.
   As a Linux user, click the top left icon, and then select **Command Line**.

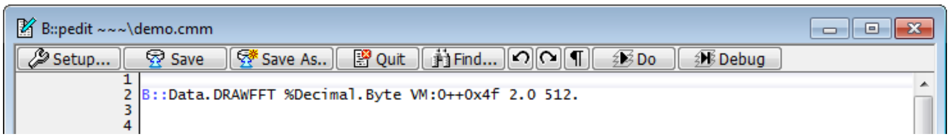
   Vertical lines are shown in the window caption [**A**].

   The command is inserted into the TRACE32 command line [**B**].



```
B::B::Data.DRAWFFT %Decimal.Byte VM:0++0x4f 2.0 512.
```

   You can now modify the command string in the command line. You can also select and copy the command in the TRACE32 command line and paste the command into a PRACTICE script file (*.cmm).



```
B::Data.DRAWFFT %Decimal.Byte VM:0++0x4f 2.0 512.
```

2. To execute the (modified) command again, click **OK**.

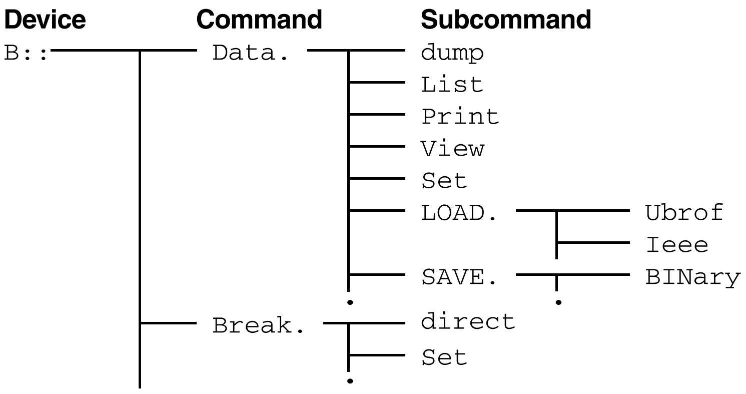
3. To deselect the window caption without executing the command again, press the **Esc** key.

Commands

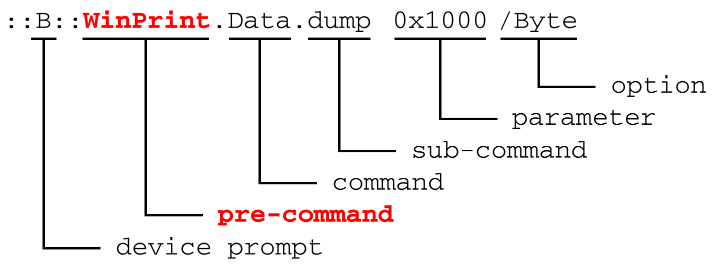
- Command Structure
- Long Form and Short Form of Commands and Functions
- Entering Commands
- Command History
- Command and Function Parameters
- For information about tab completion for commands, see **"Shortcuts"**, page 123.

Command Structure

Most commands consist of a command word, parameters, and options. The command word consists of several tokens, which are separated by a dot. Commands are combined into command groups whereby the first token of the command designates the command group. The other tokens define subcommands.



Commands can be preceded by a pre-command. Examples of pre-commands are **ChDir** (for changing the directory), **WinPrint**, or **WinExt**. Window pre-commands are used to modify the behavior of the window for a command.
WinPrint generates a hardcopy or a file from a command.



WinExt allows you to detach a window from the TRACE32 main window.

```
You can detach the window - even if TRACE32 is in MDI window mode.
WinExt.SYStem.state
```

Long Form and Short Form of Commands and Functions

Commands and functions have a long form and an equivalent short form. The two forms can be used in the TRACE32 command line and in PRACTICE scripts (*.cmm). In addition, the two forms are **not** case sensitive.

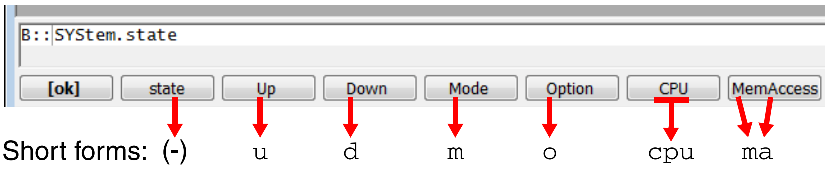
Short forms are a time saver when you are working with the TRACE32 command line. In PRACTICE scripts, the use of short forms is not recommended because short forms tend to make scripts difficult to maintain later on - for you and your colleagues.

Example of the two forms:

Long form	`SYStem.state`
Short forms	`SYS` or just `sys` • You can use short forms in UPPER CASE *or* lower case. • You can omit words in all lower-case letters, e.g. **`state`**

UPPER CASE letters in the TRACE32 application and documentation are just visual cues to indicate the short forms of commands. You can see the UPPER CASE letters of the short forms in the following places:

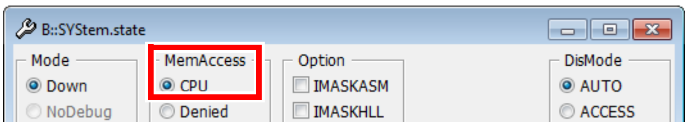
- On the softkeys below the TRACE32 command line:



```
B::SYStem.state
```

Short forms: (-) `u` `d` `m` `o` `cpu` `ma`

- In the TRACE32 windows; for example, in the **SYStem.state** window:

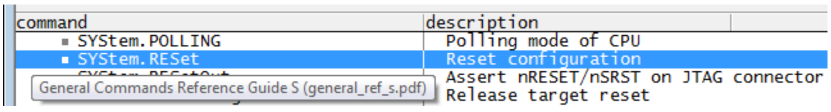


B::SYStem.state

Mode: [✓ Down] NoDebug
MemAccess: [✓ CPU] Denied
Option: ☐ IMASKASM ☐ IMASKHLL
DisMode: [✓ AUTO] ACCESS

Long form: `SYStem.MemAccess CPU`
Short form: `sys.ma cpu`

- In the online help (For example, choose **Help** menu > **Tree** to open the command tree.)

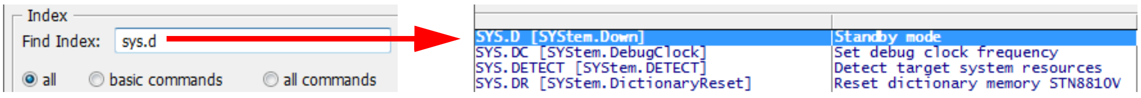


```
command                            description
  ▪ SYStem.POLLING                   Polling mode of CPU
  ▪ SYStem.RESet                     Reset configuration
                                     Assert nRESET/nSRST on JTAG connector
                                     Release target reset
```

General Commands Reference Guide S (general_ref_s.pdf)

To retrieve the long form of an unfamiliar short form (e.g. for `sys.d`):

1. Choose **Help** menu > **Index**.
2. Type the short form in the **Find Index** box, and then press **Enter**.



Index
Find Index: sys.d
[✓ all] basic commands all commands

```
SYS.D  [SYStem.Down]              Standby mode
SYS.DC  [SYStem.DebugClock]       Set debug clock frequency
SYS.DETECT  [SYStem.DETECT]       Detect target system resources
SYS.DR  [SYStem.DictionaryReset]  Reset dictionary memory STN8810V
```

Entering Commands

The long and short forms of TRACE32 commands are **not** case sensitive.

For example: Var.Watch can be abbreviated to **v.w** or to **v.watch** or to **V.WATCH** or to **var.w**

Command Line

[Back to Top]

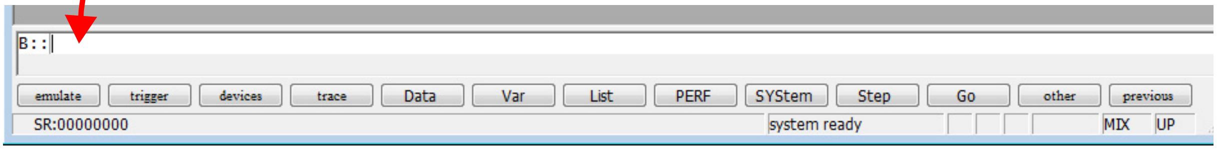
All line-oriented commands are entered to the TRACE32 command line. The command line will automatically come into focus when an alphanumeric character is entered (except Editor windows or fields). All line oriented commands are not executed until confirmed by «return» or «[ok]».

The syntax is checked immediately after every key stroke.

OS level	**::**					
	devices	**HELP**	**os**	**windows**	**practice**	**EDIT**
Device level	**::B::**					
	emulate	**Data**	**Var**	**trigger**	**devices**	**Analyzer**
Command	**B::Data.**					
	[ok]	**dump**	**View**	**Print**	**List**	**Set**
Sub-command	**B::Data.List**					
	[ok]	<range>	<address>	**options**		
Parameter	**B::Data.List 0x1000**					
	[ok]	**options**				
Option	**B::Data.List**		**/MarkPC**			
	[ok]	**Mark**	**Track**	**TOrder**	**SOrder**	**MarkPC**

NOTE:	You can copy and paste up to 300 commands (i.e. 300 lines including comments) into the command line. TRACE32 executes them like a PRACTICE script (*.cmm).

```
;set a test pattern to the virtual memory of TRACE32
Data.Set VM:0--0x4f %Byte 1 0 0 0

Data.dump VM:0x0   ;open the Data.dump window

;visualize the contents of the TRACE32 virtual memory as a graph
Data.DRAWFFT %Decimal.Byte VM:0++0x4f 2.0 512.
```



Device Selection

Each TRACE32 system has an identifier ending with two colons. The currently selected device is displayed by the prompt of the command line. System identifiers can be entered prior to each command. When a new device selector is entered prior to a command, the device selector is only valid for this specific command. The permanent selection of a device is done by entering the identifier without any command word. The TRACE32 operating system level can be accessed by entering two colons. Operating system commands can be executed from any device without using a device identifier.

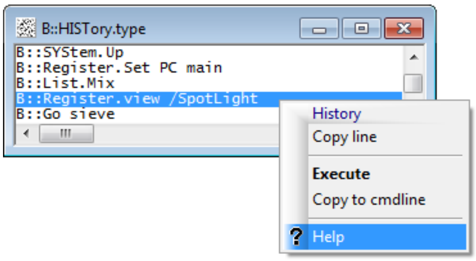
```
::B::                           ; select ICD Debugger

B::QUIT                         ; The QUIT command is a part of the
                                ; operating system and therefore, it is
                                ; recognized for all devices
```

CmdPOS	Controls the position of TRACE32 in **MWI** window mode
FramePOS	Controls the position of TRACE32 in **MDI** window mode

Command History

By clicking inside the command line and then pressing the «up» or «down» arrow keys, you can get back the previously executed commands. By entering just a keyword before pressing the «up» arrow key, it is possible to search for lines containing this word.

The command history is displayed with the command **HISTory.type**. Clicking with the mouse will copy one line to the command line.



```
B::SYStem.Up
B::Register.Set PC main
B::List.Mix
B::Register.view /SpotLight
B::Go sieve
```

HISTory.eXecute	Execute command history
HISTory.SAVE	Store command history log
HISTory.Set	History settings
HISTory.SIZE	Define command history log size
HISTory.type	Display command history log

Parameters are required for an exact definition of the operation.
Every parameter is separated from the next by spaces or a comma. .

```
; parameter separation by space
FramePOS 10. 10. 80. 50. Top WHITE

; parameter separation by comma
FramePOS 10.,10.,80.,50.,Top,WHITE
```

Omitting parameters is only possible if commas are used to separate parameters. Additionally, existing spaces are simply ignored.

```
FramePOS 10.,10.,80.,50.,Top,WHITE

FramePOS 10.,10., , , ,WHITE

FramePOS 10.,10.,,,,WHITE
```

Spaces are not allowed within parameters!
White spaces before or after operators are interpreted as separators of consecutive expressions

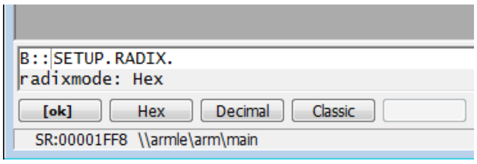
Wrong:	**Correct:**
`0y 1000`	`0y1000`
`0x1000 *0x3`	`0x1000*0x3`
`(0x1000+0x3 )*0x3`	`(0x1000+0x3)*0x3`

TRACE32 supports the following syntax for the command parameters.

Classic	Number base is hex.
Decimal	Number base is decimal, C-like operators are used.
Hex (default)	Number base is hex, C-like operators are used.

If **SETUP.RADIX.** is entered at the command line, the currently used RADIX mode is displayed in the message line.

```
B::SETUP.RADIX.
```



```
B::SETUP.RADIX.
radixmode: Hex
```

Parameter Types

Numerical values are limited to 64-bit values, strings are limited to 4095 characters. Depending on the particular command or function, the following parameters are valid:

Parameter Type	**Examples**	**RADIX.Classic**

Address	`<address> = [<access_mode>:]`
	`[[<machine_id>:::]`
	`<space_id>::]`
	`<constant>`
	UD:0x1000
	D:0x1000
	NSP:1:::0x0000::0xffff000008080004
		
		or UD:1000
		or D:1000

Address Range For details, see **Address Range** below.	D:0x4040--0x406F D:0x4040..0x406F NSP:1:::0x2000::0x8080004++0xffff func11--sYmbol.END(func11)	
ASCII value	'A'	
Binary mask or bit mask	0yX111XXX	or 0X111XXX!
Binary value	0y1 0y0 0y100010001	or 1! or 0! or 0100010001!
Boolean	*<operand1><compare_operator><operand2>* or any function returning a boolean expression, such as the functions **TRUE()** and **FOUND()**.	
Decimal value	1. 123445.	
Floating point number	1.3 1.3e+34 0.123	
Hex mask	0xFX 0xff1cxxxx	or 0FX or 0ff1cxxxx
Hex value	0x0 0xd344 0x1234 0xEEEE	or 0 or 0d344 or 1234 or 0EEEE
Range	0x10..0x20 10.--30. 0x10--0xed 'A'--'Z'	 or 10--0ed

Parameter Type	Examples	RADIX.Classic
String (*with* quotation marks)	"name" "abc""def" - string literal value: abc"def	
String (*without* quotation marks)	Strings *without* quotation marks are only used in PRACTICE functions for parameters such as: **HLL expressions** • **Var.FVALUE(ast.left->x)** **Keywords** • **TASK.STRUCT(queue)** • **WINdow.POSition(WinTOP,LEFT)** The notation format *with* quotation marks is accepted for this kind of function parameters as well.	
Time range	10us--2ms 10us..20ms Units of measurement: • **s** (seconds) • **ms** (milliseconds) • **us** (microseconds) • **ns** (nanoseconds)	
Time value	10s or 10.s are equivalent. 23.24ms 75.0ns Units of measurement: • **s** (seconds) • **ms** (milliseconds) • **us** (microseconds) • **ns** (nanoseconds)	

Parameter Type	Examples
File name MS-DOS	TEST TEST.CMM A:\FOLDER\TEST.CMM objs\abc.abs **NOTE:** 'C:TEST.C' is not valid name!
File name special (for 3rd-party tool integration e.g. Eclipse)	\\program\"C:\folder\helloworld.c" \"/home/myuser/examples/demo1.cpp"
File name UNIX/OS-9	objs/abs.abs ../src/abc.def ~/proj/src/main.c ~~~~/demo/analyzer/perf.ts
File name VMS	[.objs]abs.abs [-.src]abc.def DISK$DISK2:[t32.font]abc.d;4

PRACTICE Function	Register(PC)
	FOUND()
	OS.ENV(HOME)
	Data.Word.BigEndean(MX:0x1234)

Line numbers	\100 \MCC\100 \module\100 \\program\"C:\folder\helloworld.c"\100
Column numbers	\100\15 \MCC\100\15 \module\100\27 \\program\"C:\folder\helloworld.c"\100\15
Instance numbers	\100\15**\1** \MCC\100\15**\0** \module\100\27**\2** \\program\"C:\folder\helloworld.c"\100\15**\1** \module\100**\\2** ; default column used `(anonymous namespace)::sieve`\8**\\1** ; default column used

Parameter Type	Examples

Symbol	;*<symbol_name>*
	main
	SIEVE
	
	;\*<module_name>\<symbol_name>*
	\MCC\sieve
	
	;\\\*<machine_name>\\<program_name>\<module_name>\<symbol_name>*
	\\\1\\linux\do_mounts\load_ramdisk
	\\\Dom0\\linux\do_mounts\load_ramdisk
	
	;\\*<program_name>\<empty>\<symbol_name>* results in
	\\linux\\load_ramdisk ;2 backslashes before the *<symbol_name>*
	
	;\\\*<machine_name>\\<empty>\<empty>\<symbol_name>* results in
	\\\Dom0\\\\load_ramdisk ;4 backslashes before the *<symbol_name>*
	
	Symbol Syntax:
	`<symbol> =     \\\<machine_name>`
	`                  \\[<program_name>]`
	`                      \[<module_name>]`
	`                         \<symbol_name>`
	`                            [\<symbol_name>]...`
	`          ||`
	`              [   [\\<program_name>]`
	`                      \[<module_name>]\   ]`
	`                          <symbol_name>`
	`                            [\<symbol_name>]...`
	
	NOTE: The use of `\\\<machine_name>` requires that the machine spaces are enabled with the command **SYStem.Option.MACHINESPACES ON**.

Symbol with special chars	\`nestxf1::~nestxf1\` \\\`ops::operator<<=\`

The HLL debugger commands (all commands beginning with **Var.**) have their own syntax, which is identical to the syntax of the used high-level language.

Details about the Parameter Type Address Range

An address range consists of a start address, an operator, and an end address. The following operators between the start and end address are permissible: two dots (`..`) or two dashes (`--`) or two plus signs (`++`). If the radix mode is set to **Classic** with the command **SETUP.RADIX.Classic**, then only two dashes and two plus signs can be used.

NOTE:	The address range always includes the last byte too.

Example 1:

```
;Address range
 Data.List SP:0x0..0xFFF
;Address range
 Data.List SP:0x0--0xFFF

;RADIX.Classic: in RADIX classic you can leave out the "0x"
 Data.List SP:0--0FFF
```

Example 2: All four **Data.SAVE.Binary** commands save 0x30 bytes beginning from D:0x4040

```
;Address range --
 Data.SAVE.Binary file1.bin D:0x4040--0x406F
;Address range ..
 Data.SAVE.Binary file2.bin D:0x4040..0x406F

;Offset ++
 Data.SAVE.Binary file3.bin D:0x4040++0x2F
;Range computed with offset
 Data.SAVE.Binary file4.bin D:0x4040--(0x4040+0x2F)
```

White spaces before or after operators are interpreted as separators of consecutive expressions.
Values can be linked by operators.

Type	Example	RADIX.Classic
Brackets	**(**main+1**)***20	
Range (with borders)	0x1000**..**0x1fff 0x1000**--**0x1fff teststart**--**testend (-1000.)**--**(-50.) 'a'**--**'f' 'a'**..**'f'	or 1000**--**1FFF
Range (with offset)	0x1000**++**0x33 teststart**++**0xff	or 1000**++**33 or teststart**++**0FF
Negation	**-**1 **-**0x1 **-**0y10000	or **-**1 or **-**10000!
Binary NOT	~2e ~0x2e	or **N#**2e
Logical NOT	**!**(i<20.) **!**('a'--'z'||'A'--'Z' ||0x20||0x9||'0'--'9') **!**0x10	or **N:**(i<20.) or **N:**('a'--'z':O:'A'--'Z' :O:20:O:9:O:'0'--'9') or **N:**10
Shift left	0x10**<<**2. *result: 0x40* 0x10**<<**0x2 *result:* 0x40 0x1000--0x1fff**<<**0x4 *result:* 0x1000--0x1FFF0 "abc"**<<**3. *result:* "abcccc" "-"**<<**10. *result:* "-----------"	or 10**<<**2 or 1000--1fff**<<**4

Type	Example	RADIX.Classic
Shift right	"abc">>3. *result: "aaaabc"* 0x10>>2. *result: 0x04* 0x1000--(0x1ffff>>0x2) *result: 0x1000--0x7fff* 0x1000--0x1fff>>0x10 *result: 0xff0--0x1fef*	or 10>>2. or 1000--1ffff>>2.
Multiplication	1000.*0x2e 1000.*0y10	or 1000.*2e or 1000.*10!
Division	1000./0x2e 1000./0y10	or 1000./2e or 1000./10!
Addition **Concatenation**	0x1000+0x03 sieve+0x33 "abc"+"def" or "abc" "def" *result: "abcdef"*	or 1000+3 or sieve+33
Subtraction	0x1000-0x34 1000.-0x34	or 1000-34 or 1000.-34
Comparisons	sieve>0x1000 sieve<0x1000 sieve==0x1000 sieve!=0x1000 sieve>=0x1000 sieve<=0x1000 Data.Byte(my_char)==('a'--'z'||'0'||'1') *result: TRUE() when value is a lower alphabet character or a binary digit character “0“ or “1”* Register(PC)!=(P:0x1000|| sYmbol.RANGE(func2)|| P:0x20..P:0x2ff) *result: TRUE() when the actual program counter register value is not covered from the address ranges.*	 or sieve<>1000 or sieve=>1000 or sieve=<1000
Binary AND	mask&0x1000	or mask#A#1000
Binary XOR	mask^0x1000	or mask#X#1000

Type	Example	RADIX.Classic
Binary OR	mask|0x1000	or mask**#O#**1000
Binary Complement	~mask	or **N#**mask
Logical AND	flag0**&&**flag1 (r(D0)>d.l(i))**&&**(d.b(x)<=0x0f)	or flag0**:A:**flag1 or (r(D0)>d.l(i))**:A:**(d.b(x)<=0f)
Logical XOR	flag0**^^**flag1	or flag0**:X:**flag1
Logical OR	flag0**||**flag1 'a'--'z'**||**'0'--'9'**||**0x20**||**9.	or flag0**:O:**flag1 or 'a'--'z'**:O:**'0'--'9'**:O:**20**:O:**9
Logical NOT	**!**FOUND()	or **N:**found

Arithmetic Rules and Operator Precedence

The arithmetic hierarchy is similar to that found in most other programming languages, whereby a difference is made between boolean and arithmetic operators of logical relations. Expressions of the same priority are evaluated from left to right.

Precedence	Operands	Meaning
1.	( ) { }	Brackets (highest priority)
2.	-- ++ ..	Ranges
3.	+ - ~ ! *N# N:*	Signs, Binary NOT, Logic NOT
4.	<< >>	Shift operations
5.	* / %	Multiplication, Division, Modulo
6.	+ - +	Addition, Subtraction, Concatenation
7.	== != >= <= > <	Comparisons
8.	& *#A#*	Binary AND
9.	^ *#X#*	Binary XOR
10.	| *#O#*	Binary OR
11.	&& *:A:*	Logical AND
12.	^^ *:X:*	Logical XOR
13.	|| *:O:*	Logical OR (lowest priority)

Parentheses and Braces

The braces '{' and '}' have the same mathematical function as the parentheses '(' and ')', except that the braces additionally convert a variable expression into a constant expression.

```
B::Data.dump Register(PC)        ; The Data.dump window displays a hex dump
                                 ; of the memory range indicated by the PC.
                                 ; Whenever the PC changes the
                                 ; corresponding memory range is displayed.

B::Data.dump {Register(PC)}      ; The Data.dump window displays a hex dump
                                 ; of the memory range indicated by the PC.
                                 ; Since the current contents of the PC is
                                 ; converted to a constant expression, the
                                 ; same memory range is displayed all the
                                 ; time, even when the PC changes.
```

Parameter History

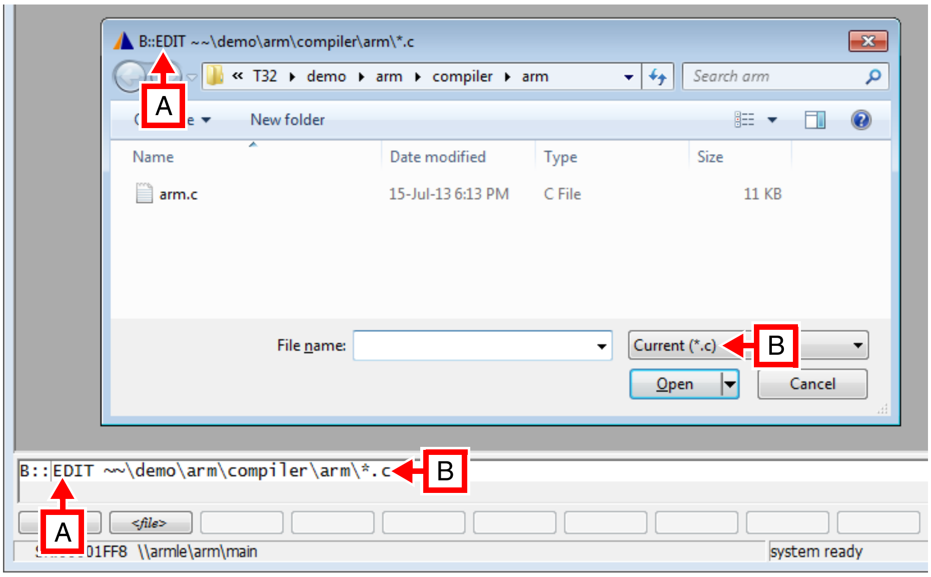
For most parameters (e.g. addresses, file names) the previous parameter entered may be recalled by using the appropriate softkey. Only one entry is stored for each parameter type.

File Names

TRACE32-PowerView supports the input of file names as follows:

- File names can be entered *without* extensions (*.xyz). The valid extension is added automatically (see **SETUP.EXTension**).
- Wildcard characters ('*****' or '**?**') are supported in file names. In this case, a file selection or folder picker dialog opens, from which you can select the file you want. See [**A**] in screenshot below.
- The file type filter can be set to automatically show the desired file types, for example `c`, `cmm`, `txt`, etc.

  In the example below, the file type filter is set to `c` files, i.e. the other files are temporarily hidden in the file selection dialog.



A The command you have used to open a window is shown as the window caption.

B Filter by file type.

For MS-DOS/Windows applications, only one working directory is supported. To access a file on another drive, the full path name must be used. Prepending the **ChDir** command before the command causes the new directory to become the current working directory.

Examples:

```
Data.LOAD  *.abs

DO \practice\*                  ; execute a PRACTICE script file from
                                ; another directory,
                                ; keep current working directory

ChDir.DO \practice\*            ; execute a script file from another
                                ; directory and make this directory to the
                                ; current working directory
```

```
EDIT  a?.c

DIR  *.obj

;inside a PRACTICE script file only, no macro replacement in command line
&practice_dir=OS.PresentPracticeDirectory()
ChDir &practice_dir
```

Path Prefixes

Tildes and periods can be used as path prefixes. There are five special path prefixes:

Linux	Windows	Function
./	.\	Current working directory
../	..\	Parent directory
~/	~\	Home directory of the user (from $HOME)
~~/	~~\	System directory of TRACE32. Default: c:\t32 on MS Windows
~~/	~~~\	Temporary directory for TRACE32
~~/	~~~~\	Directory where the currently executed PRACTICE script is located

Example:

```
;step through this PRACTICE script file (*.cmm) in the PSTEP window
PSTEP ~~/demo/arm/compiler/arm/arm9.cmm
```

NOTE:	•	In the command line, please use the path prefixes instead of the functions, e.g. **`CD ~~~~/`** instead of **OS.PresentPracticeDirectory()**.
	•	TRACE32 can handle forward slashes **`/`** on all operating systems.

General Command Parameter Parser - Behavior in the Different Radix Modes

Parser Changes in Version V2.00 and Higher

A. Object of Description

The general parameter parser for commands is the TRACE32 parser which is used for command line input, the batch language PRACTICE, the analyzer programming language, the peripheral description language and the menu programming. The parser version V2.X was introduced May 1999.

Only the **command group “Var”** which handles HLL debugging does not use the TRACE32 parser. For HLL debugging a special programming language aware parser is used. This allows the user to enter HLL expression like the following example:

```
Var.View *((long*)p_firstelement->next))
```

Different HLL parsers are implemented (e.g. for C, C++, JAVA, Ada, ...).

This description is **not intended** for these kind of special HLL parsers.

Examples of using the general TRACE32 parameter parser:

Command line:

```
Break.Set sieve /Alpha      ; sets alpha breakpoint at function begin
                            ; of sieve
Data.List P:0x1ACE          ; opens source list window at address
                            ; program 0x1ACE
Data.dump P:0x10--0x200     ; opens data dump window from address 0x10
                            ; to 0x200
DUMP mcc.abs 0xC00          ; displays file dump with file offset
                            ; of 0xC00
```

PRACTICE script files:

```
; check whether program stopped at correct address (0x1000)
IF  Register(PC)!=0x1000
(
     PRINT "Program stopped at address: " Register(PC)
     ; loads program counter with address of symbol startaddress
     ; and restart program
     Register.Set  PC  startaddress
     Go
)
ENDDO
```

Analyzer programming files:

```
TIMECOUNTER delay_counter 100ns--2ms              ; defines counter time
                                                  ; window

ADDRESS AlphaBreak func1--sYmbol.END(func3)       ; defines address event
                                                  ; from start address of
                                                  ; func1 to end address of
                                                  ; function func3
```

B. Support of C Language Expressions

Parser version V2.00 and higher supports a command parameter syntax that is similar to C language expressions.

Please be aware, it isn't a full C expression implementation, which is only available for the command group "Var" (e.g. `Var.view *(&flags+20)`).

Restrictions:

1. **Not implemented:**

   sizeof(), (typename), assignment operator (=,+=,-=,*=,/=,%=,>>=,<<=, &=,|=,^=), array[], pointer->element, structure.element, *p_value, &flags[20], (a==2)?1:2
   e.g. `a += b+3;`

2. **Different meaning:**

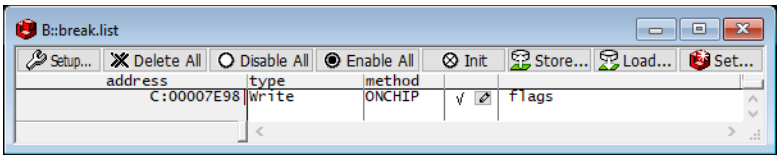
   ++ (prefix and postfix; e.g. `i++`) will be used for range offset input
   e.g. `1234.++1000.,`

   -- (prefix and postfix; e.g. `i--`) will be used for range offset input
   e.g. `100ns--200ns,`

   Symbol names will be interpreted always as &symbolname (start address of symbol) and not as name or value for the complete symbol.

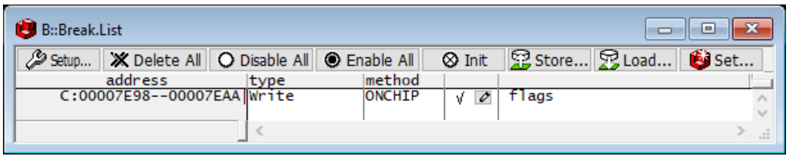
   Example:

```
Break.Set flags /Write              ; Sets a write breakpoint to the
                                    ; start address of the variable flags
```



```
B::break.list
Setup...  Delete All  Disable All  Enable All  Init  Store...  Load...  Set...
             address |type   |method
          C:00007E98 |Write  |ONCHIP  √    flags
```

```
Var.Break.Set flags /Write          ; Sets a write breakpoint to the
                                    ; complete address range used for
                                    ; the variable flags
```



```
B::Break.List
Setup...  Delete All  Disable All  Enable All  Init  Store...  Load...  Set...
             address |type   |method
C:00007E98--00007EAA |Write  |ONCHIP  √    flags
```

   The character **&** is used to mark **PRACTICE macros** (e.g. `&cpu="MPC860"`)

3. **Extensions:**

   logical XOR (^^), data type boolean, bit constants, bit masks, hex masks, ranges, addresses, address ranges, times, time ranges can use**..** to define a range (e.g. `123..456`)

C. Radix Mode Support

Parser version V2.00 and higher supports radix (number base) switching.

Depending on the selected radix the written values are interpreted in a different way.

E.g. `123` could be meant as 123 decimal or 123 hexadecimal depending on the used radix mode.

RADIX Modes

RADIX Radix mode

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.

Classic	number base is hex - all input formats for operands and operators permitted
Decimal	number base is decimal - **old-fashioned operators** and **operands** are locked
Hex	number base is hex - old-fashioned operators and operands are locked - default

If **RADIX.** is entered in the command line, the currently used RADIX mode is displayed in the state line.

```
E::RADIX.
radixmode: Hex
```

To use PRACTICE and analyser programming files written in old-fashioned format please insert the command **RADIX.Classic** in the first line of the start up PRACTICE script.

Written Value	**Interpreted Value in Radix Mode**		
	Classic	**Decimal**	**Hex**
1000	1000**h**==4096**d**	1000**d**==1000**d**	1000**h**==4096**d**
P:1234	P:1234**h**==P:4660**d**	P:1234**d**==P:1234**d**	P:1234**h**==P:4660**d**
1000.	1000**d**	1000**d**	1000**d**
1234.	1234**d**	1234**d**	1234**d**

’**d**’: decimal value - ’**h**’: hexadecimal value.

D. Incompatibilities in RADIX.Classic Mode

This section describes incompatibilities between radix mode Classic of the parser version 2 and higher compared to older versions. The parser version V2.X was introduced May 1999.

Written Value	**Interpreted Value**		**New Syntax for V2.00 and later**
	<=V1.90	**>=V2.00**	
0xx13 (see 1.)	xx13	x13	**0X**xx13
0xxxx1100! (see 1.)	xxxx1100!	syntax error	**0Y**xxxx1100
0x23af (see 1.)	**x**23af	23af **!!**	**0X**x23af
0x23x56x (see 1.)	**x**23x56**x**	23**x**56**x !!**	**0X**x23x56x
.symbolname (symbol names with **.** as prefix, see 2.))	**.symbolname**	syntax error	**‘.symbolname‘** or **symbolname** and **sYmbol.PREFIX .**

1. Now bit- and hex masks which start with 0x (e.g. `0xx10`) will always be interpreted as a hex constant (only one ’`x`’ in value - e.g. `0x23`) or hex mask (more than one ’`x`’ - e.g. `0x2x4`). The result will be a different and wrong constant value or an error message compared to versions <= V1.90.

NOTE:	In consequence existing PRACTICE files and especially analyser trigger programs written in old-fashioned syntax will not run correctly or will generate an error messages. Please refer to the **Operand transition table**.

2. Now symbol names which begin with the prefix character ’.’ will generate an error message. This could be avoided if:

   - the whole symbol name is put into quotation marks (e.g ’`.start`’)

   - the automatic symbol prefix is set to ’.’ with the command **sYmbol.PREFIX .** Then the symbol name has to be entered without prefix (e.g. `start`).

Operands

Examples for operands:

```
Break.Set sieve /Alpha      ; sets alpha breakpoint at function begin of
                            ; sieve
Data.List P:0x1AF           ; opens source list window at program address
                            ; 0x1AF
Data.dump P:0x10--0x1ff     ; opens a data dump window from address 0x10
                            ; to 0x1ff
```

Restriction:

Not all operand formats could be used in all radix modes. Please refer to the **Operand Format Table**.

Operand	Meaning	Radix mode		
		Classic	Decimal	Hex
0y1010	binary constant	X	X	X
1010!	binary constant	X		
0x12af	hex constant	X	X	X
1234	hex constant	X		X
1234	decimal constant		X	
1000.	decimal constant	X	X	X
1.2	float constant	X	X	X
0y10xx10	bitmask constant	X	X	X
10xx10!	bitmask constant	X		
0x12axfx	hexmask constant	X	X	X
12axfx	hexmask constant	X		
'a'	ASCII constant	X	X	X
"abcdef"	string constant	X	X	X
"abc""def"	string constant with escape sequence for using string delimiter inside string literals string value: abc”def	X	X	X
\`main\`	backticks for HLL symbols	X	X	X
1000--2000	numeric range constant	X	X	X
1000..2000	numeric range constant	X	X	X
P:0x1af	address constant (hex)	X	X	X
P:1234	address constant (hex)	X		X
P:1234	address constant (decimal)		X	
P:1234.	address constant (decimal)	X	X	X
P:0x1000--0x1fff	address range constant	X	X	X
P:0x1000..0x1fff	address range constant	X	X	X
123ms	time constant	X	X	X
123ns--4.25s	time range constant	X	X	X
123ns..4.25s	time range constant	X	X	X

Operand Transition Table

Please use the table below for converting the old-fashioned data type format into the new-fashioned formats.

Old-fashioned **<= V1.90**	**New-fashioned** **>= V2.00**	**Meaning**
1010**!**	**0y**1010	bit constant
12af	**0x**12af	hex constant
10xx10**!**	**0y**10xx10	bitmask constant
12axfx	**0x**12axfx	hexmask constant

Examples for the use of operators:

Command line:

```
Data.dump P:0x10++(Register(D0)%4)          ; open data dump window from
                                            ; address 0x10 to offset value
                                            ; in register D0 modulo 4
```

PRACTICE script files:

```
IF  Register(PC)!=0x1000                    ; check whether program
                                            ; stopped at the correct
                                            ; address
```

Analyzer programming files:

```
DATA.BYTE ascii  'a'--'z'||'A'--'Z'             ; define data event with
                                                ; the alphabet as valid
                                                ; values

ADDRESS AlphaBreak !(fct1--sYmbol.END(fct3))    ; define an address event
                                                ; over the whole 4 giga
                                                ; address space without
                                                ; the address range from
                                                ; start address of func1
                                                ; to end address of func3
```

Not all operators could be used in all radix modes. Please refer the **Operand Format Table**.

Operator	**Meaning**	**Radix mode**		
		Classic	**Decimal**	**Hex**
!	logical NOT	X	X	X
&&	logical AND	X	X	X
^^	logical XOR	X	X	X
||	logical OR	X	X	X
N:	logical NOT	X		
:A:	logical AND	X		
:X:	logical XOR	X		
:O:	logical OR	X		
~	binary NOT	X	X	X
&	binary AND	X	X	X
^	binary XOR	X	X	X
|	binary OR	X	X	X
N#	binary NOT	X		
#A#	binary AND	X		
#X#	binary XOR	X		
#O#	binary OR	X		
-	negation or minus	X	X	X
+	plus	X	X	X
*****	multiplication	X	X	X
/	division	X	X	X
%	modulo (reminder)	X	X	X
<<	shift left	X	X	X
>>	shift right	X	X	X

Operator	**Meaning**	**Radix mode**		
		Classic	**Decimal**	**Hex**
<	smaller than	X	X	X
>	greater than	X	X	X
<=	smaller or equal than	X	X	X
=<	smaller or equal than	X		
>=	bigger or equal than	X	X	X
=>	bigger or equal than	X		
==	equal to	X	X	X
<>	not equal	X		
><	not equal	X		
!=	not equal	X	X	X
() {}	parenthesis	X	X	X
--	range with borders	X	X	X
..	range with borders	X	X	X
++	range with offset	X	X	X

Operator Transition Table

Please use the following table to convert old-fashioned operators into the new format.

Old-fashioned	New-fashioned	Meaning
N#	~	bitwise negation
#A#	&	bitwise and
#X#	^	bitwise xor
#O#	|	bitwise or
N:	!	logical negation
:A:	&&	logical and
:X:	^^	logical xor
:O:	||	logical or
<>	!=	not equal comparator
><	!=	not equal comparator
=<	<=	smaller or equal comparator
=>	>=	bigger or equal comparator
not available	%	modulo (remainder)

Window System

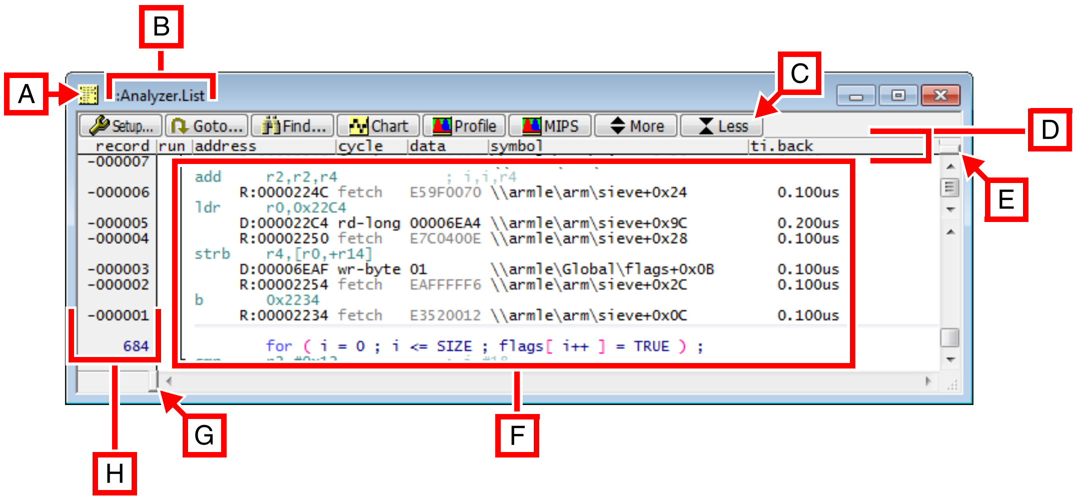
Windows

[Back to Top]

All outputs of the TRACE32 system are displayed in windows. Usually, all windows display current data because they are updated periodically.

Windows can be closed by the **esc** key. This allows to temporarily display some information and quickly close the window again.

TRACE32 windows typically consist of some or all of the following components:



```
 record |run |address          |cycle   |data     |symbol                             |ti.back
-000007
          add     r2,r2,r4                ; i,i,r4
-000006       R:0000224C fetch   E59F0070 \\arm1e\arm\sieve+0x24            0.100us
          ldr     r0,0x22C4
-000005       D:000022C4 rd-long 00006EA4 \\arm1e\arm\sieve+0x9C            0.200us
-000004       R:00002250 fetch   E7C0400E \\arm1e\arm\sieve+0x28            0.100us
          strb    r4,[r0,+r14]
-000003       D:00006EAF wr-byte 01       \\arm1e\Global\flags+0x0B         0.100us
-000002       R:00002254 fetch   EAFFFFF6 \\arm1e\arm\sieve+0x2C            0.100us
          b       0x2234
-000001       R:00002234 fetch   E3520012 \\arm1e\arm\sieve+0x0C            0.100us

    684           for ( i = 0 ; i <= SIZE ; flags[ i++ ] = TRUE ) ;
```

A **Window manager menu**: Clicking the icon lets you open the window manager menu.

B **Window caption**: It displays the TRACE32 command that was used to open the window.

C **Local buttons** of a window.

D **Scale area**: Column headers of a window.

E **Slider control** (top).

F **Data area**: It contains the actual values or information.

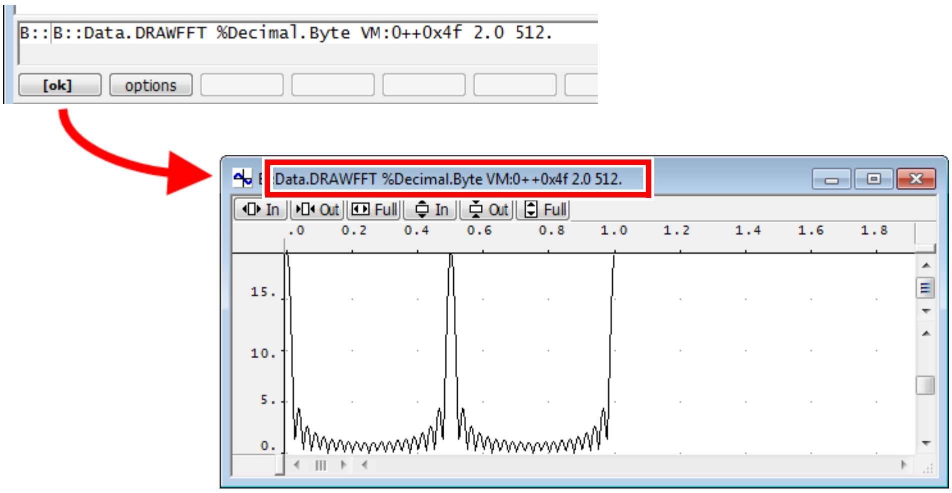
G **Slider control** (bottom).

H **Scale area**: Additional information about lines, such as line numbers, record numbers, addresses, breakpoints, bookmarks, etc.

Window Captions - What Makes Them Special in TRACE32

[Back to Top]

The command you have used to open a window is shown as the window caption. The parameters and options are also included in the window caption.



```
B::B::Data.DRAWFFT %Decimal.Byte VM:0++0x4f 2.0 512.
```

```
Data.DRAWFFT %Decimal.Byte VM:0++0x4f 2.0 512.
```

In addition, you can easily modify the window caption with a simple mouse-click. For more information, refer to "Modifying and Re-using Commands Shown In Window Captions".

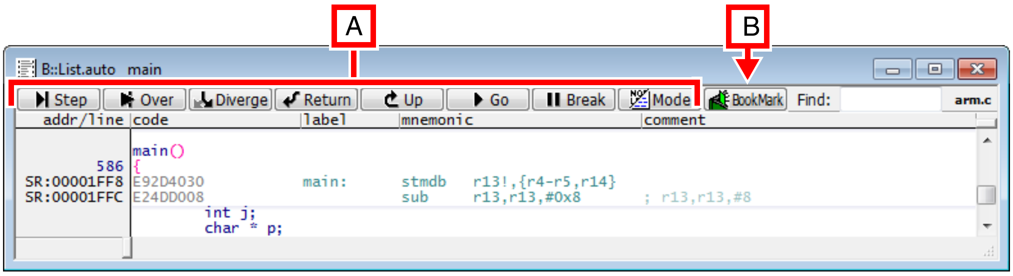
Example: This script allows you to reproduce the above **Data.DRAWFFT** window:

```
;set a test pattern to the virtual memory of TRACE32
Data.Set AVM:0--0x4f %Byte 1 0 0 0

Data.dump AVM:0x0 ;open the Data.dump window

;visualize the contents of the TRACE32 virtual memory as a graph
Data.DRAWFFT %Decimal.Byte AVM:0++0x4f 2.0 512.
```

Local Buttons

[Back to Top]

Many TRACE32 windows have built-in local buttons [**A**]. In addition, you can extend TRACE32 windows with user-defined local buttons [**B**].



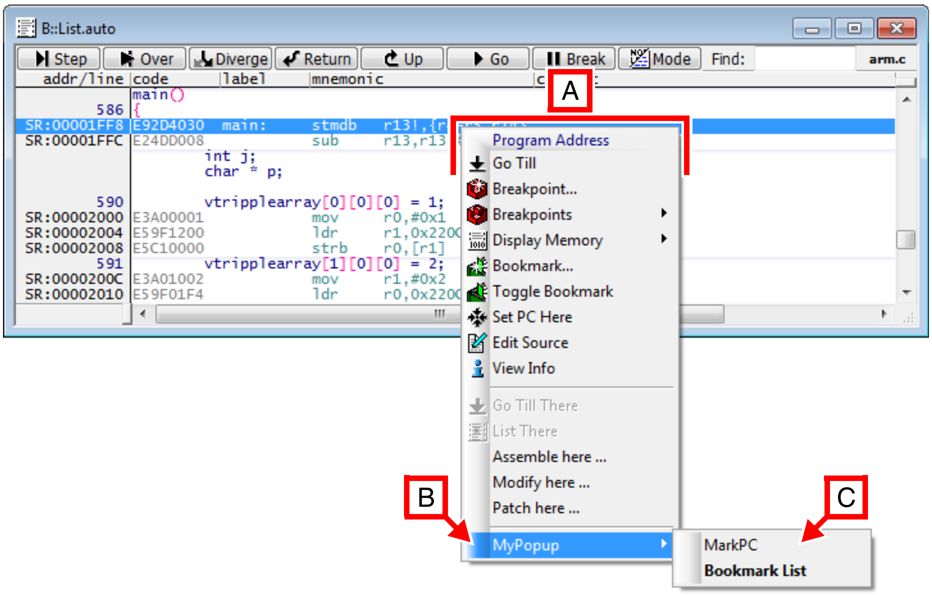
```
  addr/line code          label      mnemonic                  comment
            main()
        586 {
SR:00001FF8 E92D4030      main:      stmdb   r13!,{r4-r5,r14}
SR:00001FFC E24DD008                 sub     r13,r13,#0x8      ; r13,r13,#8
                  int j;
                  char * p;
```

For an example of how to program your own local buttons in TRACE32 windows, see the **BUTTONS** command.

Local Popup Menus

[Back to Top]

You can extend the built-in local popup menus of TRACE32 windows with your own local popup menus and menu items, as shown in this example of a **List.auto** window:



```
addr/line  code      label    mnemonic
                main()
       586      {
SR:00001FF8 E92D4030  main:    stmdb    r13!,{r
SR:00001FFC E24DD008           sub      r13,r13
                  int j;
                  char * p;
       590        vtripplearray[0][0][0] = 1;
SR:00002000 E3A00001           mov      r0,#0x1
SR:00002004 E59F1200           ldr      r1,0x220C
SR:00002008 E5C10000           strb     r0,[r1]
       591        vtripplearray[1][0][0] = 2;
SR:0000200C E3A01002           mov      r1,#0x2
SR:00002010 E59F01F4           ldr      r0,0x220C
```

A Built-in local popup menu named **Program Address**.

B User-defined local popup menu.

C User-defined menu items.

There are two ways to add your own menu items to popup menus in TRACE32 windows:

- You can assign your own menu items to the command short form of a TRACE32 window, e.g. to the command short form **L.** for the **List.auto** window. As a result, your own menu items are only visible in the **List.auto** window, but not in the **List.Mix** nor the **List.Asm** window nor any other window.
  For information about command short forms, see **"Long Form and Short Form of Commands and Functions"**, page 29.

- You can assign your own menu items to the built-in popup menus **Program Address** and **Variable**. As a result, your own menu items are visible in all TRACE32 windows that have these popup menus, such as the following windows: **List.auto**, **List.Mix**, **List.Asm**, **Data.dump**, **Var.Watch**, etc.

For examples of how to programmatically extend a TRACE32 window with your own menu items, refer to the menu programming command **MENU**.

Slider Controls

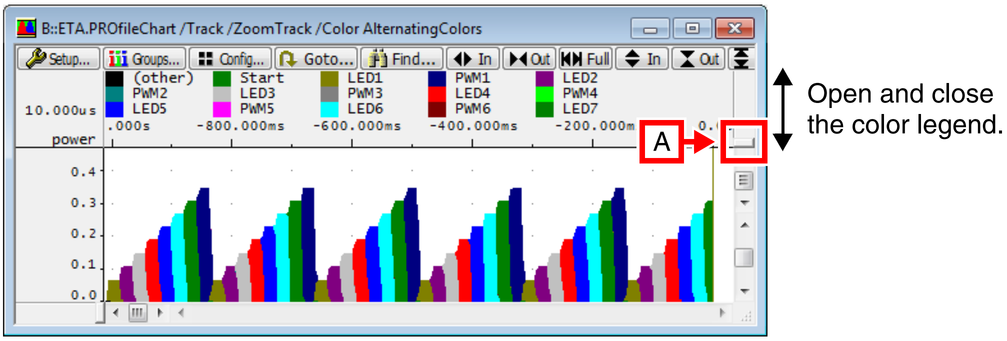
Most windows that output data have slider controls. By dragging the slider controls, you can:

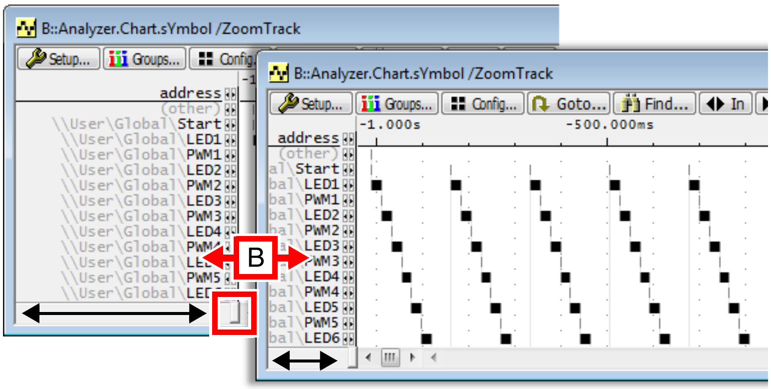
1. Open and close legends, e.g. the color legends of charts in **ProfileChart** windows, see [**A**].

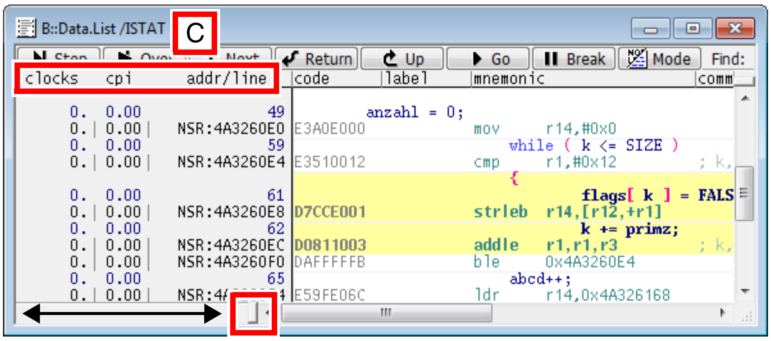
2. Resize the scale area, see [**B**].

3. Display new columns after modifying a command on the fly. In example [**C**], the **Data.List** command is modified by adding **ISTAT**. To display the new columns, drag the slider control to the right.

   For information about how to modify a command displayed in a window caption, see “Modifying and Re-using Commands Shown In Window Captions”.







```
clocks    cpi        addr/line  code      label     mnemonic                comm
     0.   0.00              49                        anzahl = 0;
     0. | 0.00 |  NSR:4A3260E0  E3A0E000            mov      r14,#0x0
     0.   0.00              59                            while ( k <= SIZE )
     0. | 0.00 |  NSR:4A3260E4  E3510012            cmp      r1,#0x12            ; k,
                                                          {
     0.   0.00              61                                flags[ k ] = FALS
     0. | 0.00 |  NSR:4A3260E8  D7CCE001            strleb   r14,[r12,+r1]
     0.   0.00              62                                k += primz;
     0. | 0.00 |  NSR:4A3260EC  D0811003            addle    r1,r1,r3            ; k,
     0. | 0.00 |  NSR:4A3260F0  DAFFFFFB            ble      0x4A3260E4
     0.   0.00              65                            abcd++;
     0. | 0.00 |  NSR:4A      4  E59FE06C            ldr      r14,0x4A326168
```

Window Operations

Basic Operations

All basic operations (e.g. move window, iconize window) are fully compatible with the host operating system.

Old Position, Bookmarks, and Current Selection

You can place visible bookmarks and one hidden bookmark in TRACE32 windows that output data, e.g. in **Trace.List** or **List** windows. Using bookmarks, you can navigate between bookmarked locations.

Visible bookmarks	**View** menu > **Bookmarks** opens the **Bookmark.List** window. The steps below describe how to place visible bookmarks. For more information about visible bookmarks and the difference between the bookmark colors yellow and green, see **BookMark**.
Hidden bookmark	**Recall Position** returns to the position you have previously saved with **Store Position**. The steps below describe how to place a hidden bookmark.
Current Selection	**Goto Selection** returns you to the currently selected position or last active view (in case the selection is no longer active).

To place visible bookmarks in a window:

1. Choose **View** menu > e.g. **Trace List** to open a **Trace.List** window.
2. Right-click where you want to place a visible bookmark, and then select **Toggle Bookmark**.
   - Scroll somewhere else within the same window, and then place another bookmark.
3. To view the bookmark list, choose **View** menu > **Bookmarks**.

```
BookMark.List ;alternatively use the TRACE32 command line to open
              ;the BookMark.List window
```

Tips: To go to a bookmark location, you have the following options:

- Double-click a bookmark in the **BookMark.List** window. A new window opens, displaying the bookmark location.
- Open a new window with the **Track** option, for example:

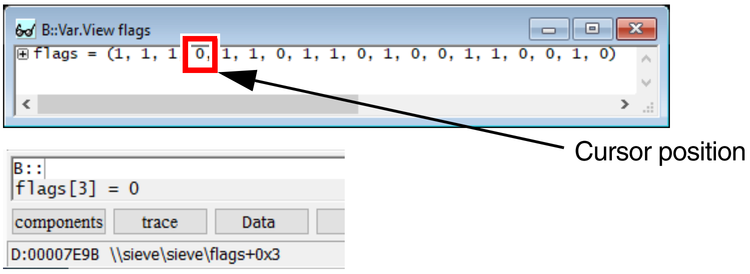
```
Trace.List /Track

BookMark.List ;now click the bookmark you want in the BookMark.List
              ;window to jump to that bookmark location in the
              ;Trace.List /Track window
```

To place a hidden bookmark in a window:

1. Choose **View** menu > **Trace** to open a **Trace.List** window.
2. Click where you want to place the hidden bookmark.
3. Choose **Edit** menu > **Store Position**.
   - Scroll somewhere else within the same window.
4. To return to the last stored position, choose **Edit** menu > **Recall Position**.

Getting Information

If the left mouse button is held down, additional information will be displayed concerning the field addressed by the cursor position.



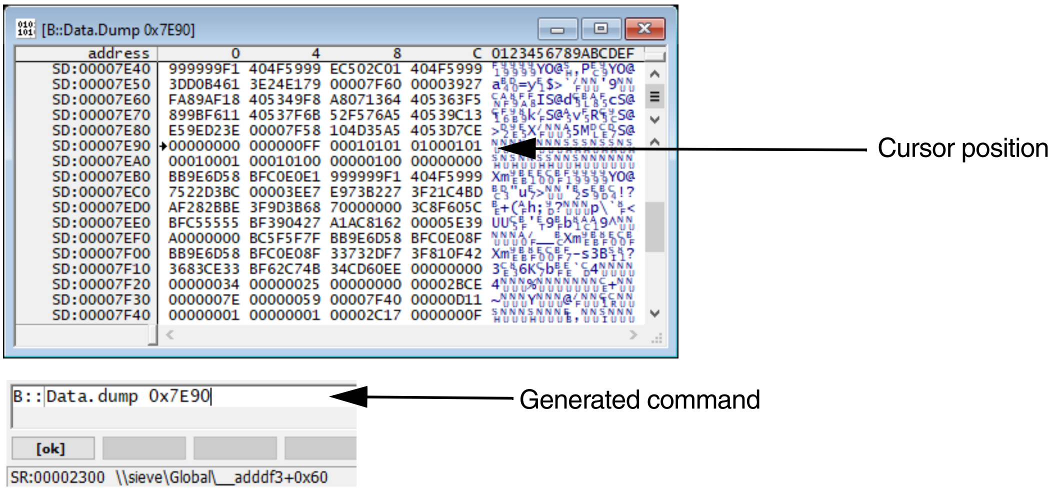
```
flags = (1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0)
```

```
B::
flags[3] = 0
```

```
D:00007E9B  \\sieve\sieve\flags+0x3
```

Changing Data or Setups

A double click to a field with the left mouse key will invoke a change command such as «Data.Set» or «Register.Set».



```
         address          0        4        8        C 0123456789ABCDEF
     SD:00007E40  999999F1 404F5999 EC502C01 404F5999
     SD:00007E50  3DD0B461 3E24E179 00007F60 00003927
     SD:00007E60  FA89AF18 405349F8 A8071364 405363F5
     SD:00007E70  899BF611 40537F6B 52F576A5 40539C13
     SD:00007E80  E59ED23E 00007F58 104D35A5 4053D7CE
     SD:00007E90 +00000000 000000FF 00010101 01000101
     SD:00007EA0  00010001 00010100 00000100 00000000
     SD:00007EB0  BB9E6D58 BFC0E0E1 999999F1 404F5999
     SD:00007EC0  7522D3BC 00003EE7 E973B227 3F21C4BD
     SD:00007ED0  AF282BBE 3F9D3B68 70000000 3C8F605C
     SD:00007EE0  BFC55555 BF390427 A1AC8162 00005E39
     SD:00007EF0  A0000000 BC5F5F7F BB9E6D58 BFC0E08F
     SD:00007F00  BB9E6D58 BFC0E08F 33732DF7 3F810F42
     SD:00007F10  3683CE33 BF62C74B 34CD60EE 00000000
     SD:00007F20  00000034 00000025 00000000 00002BCE
     SD:00007F30  0000007E 00000059 00007F40 00000D11
     SD:00007F40  00000001 00000001 00002C17 0000000F
```

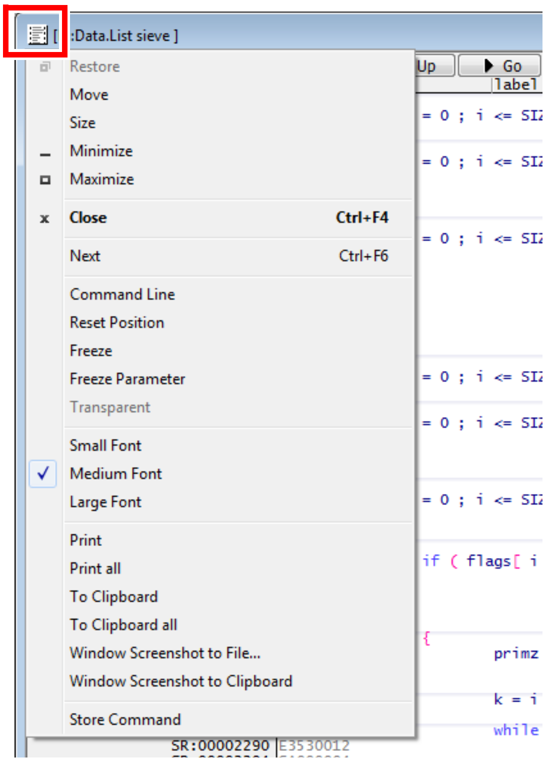
```
B::Data.dump 0x7E90
```

```
SR:00002300  \\sieve\Global\__adddf3+0x60
```

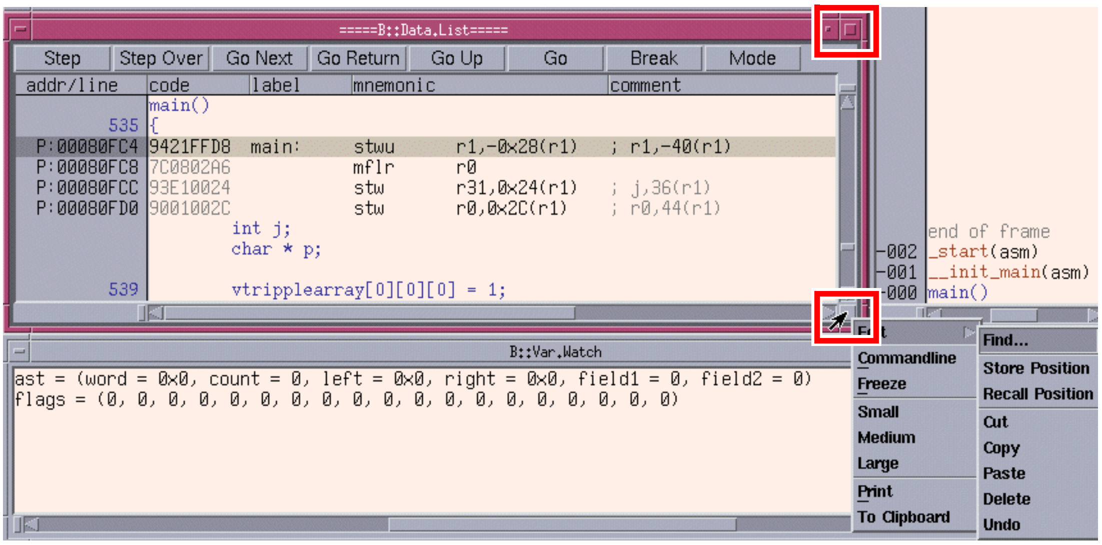
Window Manager Menu

The windows in TRACE32 provide a window manager menu with special commands. For a short description of the these commands, see below.

- **Windows GUI**: To access the window manager menu, click the icon in the top left corner of a window:



- **Motif GUI**: To access the window manager menu, right-click the window manager button. The window manager button is located on the right upper or right lower corner of a Motif window.



```
addr/line   code      label     mnemonic            comment
            main()
      535   {
P:00080FC4  9421FFD8  main:     stwu      r1,-0x28(r1)     ; r1,-40(r1)
P:00080FC8  7C0802A6            mflr      r0
P:00080FCC  93E10024            stw       r31,0x24(r1)     ; j,36(r1)
P:00080FD0  9001002C            stw       r0,0x2C(r1)      ; r0,44(r1)
                      int j;
                      char * p;

      539             vtripplearray[0][0][0] = 1;
```

```
end of frame
-002 _start(asm)
-001 __init_main(asm)
-000 main()
```

```
ast = (word = 0x0, count = 0, left = 0x0, right = 0x0, field1 = 0, field2 = 0)
flags = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
```

Next	Jump to next window.
Command Line	Inserts the window caption (= command) in the command line. • On a Windows GUI, right-click the window caption. • On a Motif GUI, click the window manager button, and then select **Command line**. You can now modify and run the command again or re-use it in a PRACTICE script (*.cmm). See also "**Window Captions - What makes them special in TRACE32**".
Reset Position	Returns to the position specified in the window caption. Examples of window captions: **B::Data.dump (0x100)** => Returns to address 0x100 **B::Data.List func9** => Returns to symbol func9 **B::Trace.List -000212.** => Returns to record -000212.
Freeze	Freezes the window contents. Executing the function again will change back to a cyclic update of the window.
Freeze Parameter	Freezes the window parameters. **Example:** Data.Dump Var.Value(MyVar) If Freeze Parameter is used, the dumped memory addresses are not updated if the variable value will change.
Small, Medium, Large Font	Changes the size of the font for the window. Switching to **Large Font** is very useful in presentations before large audiences. See also **WinSmall**, **WinMid**, **WinLarge**.
Transparent	Makes the window transparent (only available for MWI interface of Windows 2000 and later). These kind of external windows will allow windows in the background to shimmer through. See also **WinTrans**.

Print **Print All**	The result of **Print** or **Print All** depends on the output medium you have selected in the **PRinTer** dialog: 1. Choose **File** menu > **Printer Settings** to open the **PRinTer** dialog. 2. Select the output medium you want: **printer**, **ClipBoard**, **FILE**, or **Area**. Depending on your selection, the window contents can now a) be sent to the printer or b) copied to the clipboard or c) saved to file or d) printed to an **AREA** window. • **Print** prints only the visible window contents to the selected output medium • **Print all** behaves within a TRACE32 window as if you scroll to the top of the terminal buffer and choose **Print**, then scroll down one visible terminal page and do the next **Print**, and so on. **NOTE**: To process huge amounts of data, e.g. from a **List.auto** window, we recommend that you redirect the output to a file instead. See **PRinTer.FILE** example. See also **PRinTer** and **PRinTer.select**.
To Clipboard	**To Clipboard** copies the visible window contents as text to the clipboard. See also **PRinTer**.
To Clipboard all	**To Clipboard All** behaves within a TRACE32 window as if you scroll to the top of the terminal buffer and choose **To Clipboard**, then scroll down one visible terminal page and do the next **To Clipboard**, and so on. **NOTE**: To process huge amounts of data, e.g. from a **List.auto** window, we recommend that you redirect the output to a file instead. See **PRinTer.FILE** example. See also **PRinTer**.
Window Screen-shot to File	Captures a screenshot of the active window and opens the **Save Window Screenshot** dialog. Enter file name and select file type (PNG, GIF etc.) See also **SCreenShot**.
Window Screen-shot to Clipboard	Copies a screenshot of the visible part of the window to the clipboard.
Store Command	Saves the window caption (= command) as a PRACTICE script (*.cmm). The position, size, and name of the window as well as column widths are also included in the script. See also **STOre**.

Window Position and Name

The size and position of a window generated by a command can be predefined by the command **WinPOS**. A name can be specified for this window. This command in mainly used in PRACTICE scripts (*.cmm), which were generated by the **STOre** command.

AutoSTOre	Store settings automatically
STOre	Generate a script that allows to reproduce the current setting or settings
ClipSTOre	Store settings to the clipboard
WinPOS	Define position, size, and name of the next window
WinOverlay	Pile up windows on top of each other

Example:

```
;         <x>   <y>   <w>   <h>          <window_name>
WinPOS    5.0   5.0   58.   8.   , , ,      TEXT1
TYPE ~~~\test.txt /LineNumbers
```

Freezing a Window

A window is frozen by choosing the **Freeze** command of the window manager menu. A frozen window is no longer updated with the current state. Therefore, it can no longer be scrolled, because the required data are missing. The pre-command **WinFreeze** will generate a frozen window from the command line.

Erasing a Window

Windows are deleted like any other window on the host. All windows can be deleted without loss of data, e.g. when using the editor. The command **WinCLEAR** without parameters deletes all windows on the current window page. All window pages are deleted by the **WinPAGE.RESet** command.

WinCLEAR	Erase all windows on one page or a named window
WinPAGE.RESet	Erase all pages

Window Scroll Bars

In the case of most windows with a finite size, the relationship between the displayed section and the entire size of the window is represented in the scroll bars located at the borders of the window. Infinite windows, like a hex dump, have no moving scroll bar.

Printing Window Contents

To print a hardcopy of the active window, select the **Print** command from the window manager menu. Larger areas can be printed by adding the pre-command **WinPrint**.

Printers must be configured in the config file (default config.t32). The installation of printers is described in the **INSTALLATION GUIDE**.

WinPrint.<*command*>	Print one window (full printer size) to file
WinPRT	Make hardcopy of existing window
PRinTer.HardCopy	Print all windows on screen
PRinTer.select	Select type of printer
PRinTer.ClipBoard	Re-route printer output to clipboard in specified format
PRinTer.Area	Re-route printer output to **AREA** window in specified format

Saving Window Contents

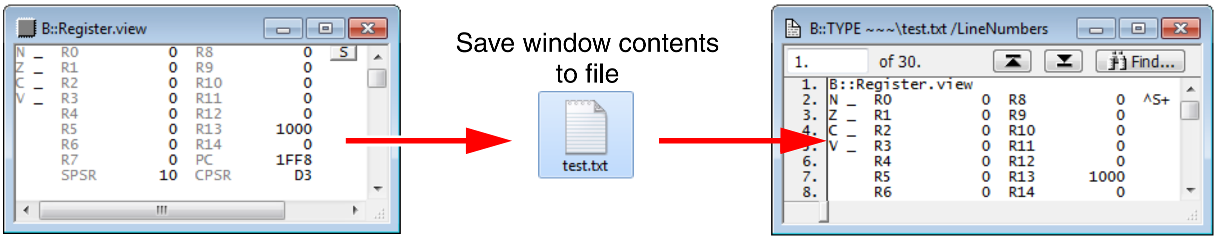
[Example]

The **PRinTer** commands can be used to redirect and save window contents to a file. The output file can either contain one printout or combine multiple printouts in one file. The output format of the file can be either a plain ASCII format for postprocessing or POSTSCRIPT for use in document processing tools.

PRinTer.OPEN	Open file and re-route multiple printer outputs to this file
PRinTer.FILE	Define file for single printer output and select output format for file
PRinTer.CLOSE	Close file after multiple printer outputs
WinPrint	Print one window (full size) to file
PRinTer.EXPORT	Export CSV-formatted printer output to file

Example: The contents of the **Register.view** window are saved to file, which is then opened in the **TYPE** window. The path prefix ~~~ expands to the temporary directory of TRACE32.

```
Register.view                 ;optional step: open the window
PRinTer.FILE ~~~\test.txt     ;create and open a file for writing
WinPrint.Register.view        ;print the window contents to file
TYPE ~~~\test.txt /LineNumbers  ;open the file in the TYPE window
```


B::Register.view
Save window contents to file
test.txt
B::TYPE ~~~\test.txt /LineNumbers
1.
of 30.
Find...

```
N _   R0           0  R8           0  S
Z _   R1           0  R9           0
C _   R2           0  R10          0
V _   R3           0  R11          0
      R4           0  R12          0
      R5           0  R13       1000
      R6           0  R14          0
      R7           0  PC       1FF8
      SPSR        10  CPSR        D3
```

```
 1. B::Register.view
 2. N _   R0           0  R8           0  ^S+
 3. Z _   R1           0  R9           0
 4. C _   R2           0  R10          0
 5. V _   R3           0  R11          0
 6.       R4           0  R12          0
 7.       R5           0  R13       1000
 8.       R6           0  R14          0
```

Special Window Options

Windows with some special behavior can be created by the following commands:

WinBack	Generates a window on background
WinDuplicate	Duplicates window
WinExt	Generates an external separate window (MWI like)
WinFreeze	Generates a frozen window
WinLarge	Generate window with large font
WinMid	Generate window with regular font
WinOverlay	Pile up windows on top of each other
WinPAGE	Window pages
WinPAGE.Create	Create page
WinPAGE.Delete	Delete page
WinPAGE.List	List pages
WinPAGE.select	Select page
WinResist	Generates a window which cannot be erased by **WinCLEAR**
WinRESIZE	New size for window
WinSmall	Generate window with small font
WinTABS	Define TABs
WinTOP	Bring window to top
WinTrans	Generate transparent window

Examples:

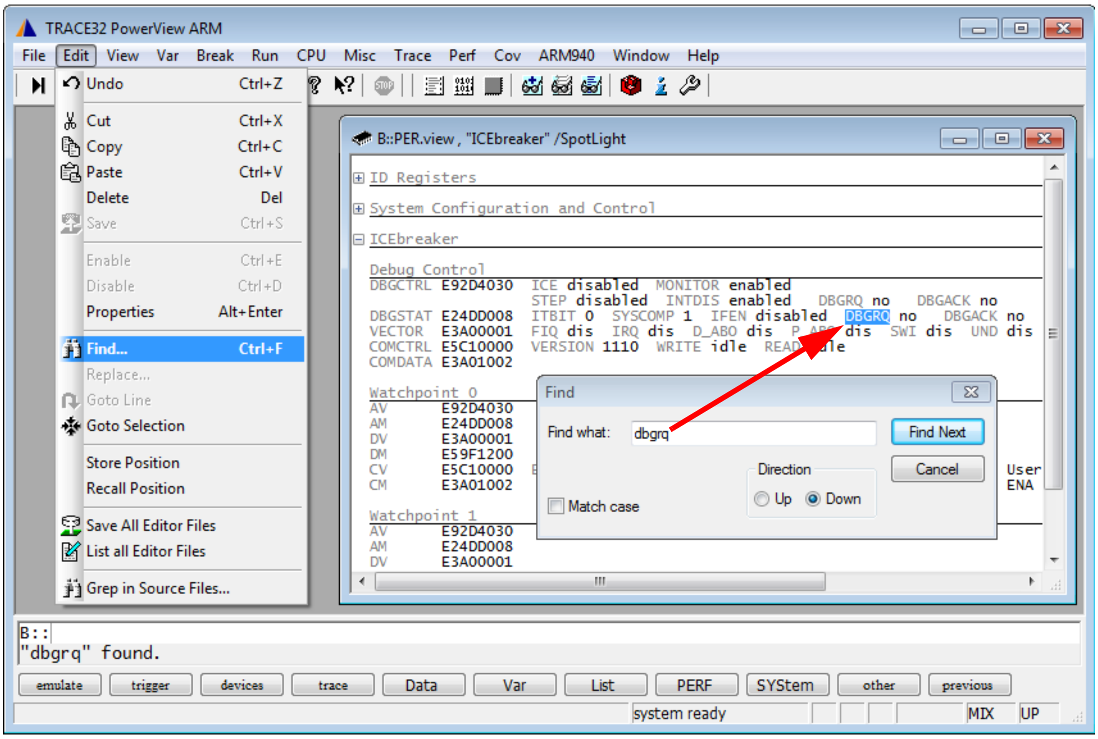
```
WinBack.AREA error

WinFreeze.Data.dump 0x1000

WinResist.PEDIT test
```

Text-based Functions

The text-based functions are available in all windows. They allow searching for text and control the display excerpt of the window.

WinFIND	Search for a text string in a window
WinPAN	Scroll window

The **Find** function can be accessed from the **Edit** menu window (Windows) or from the window manager menu (Motif). This example shows that you can search for text in a peripherals file (**PER.view** window).



```
⊞ ID Registers
⊞ System Configuration and Control
⊟ ICEbreaker

  Debug Control
  DBGCTRL  E92D4030  ICE disabled  MONITOR enabled
                     STEP disabled  INTDIS enabled   DBGRQ no    DBGACK no
  DBGSTAT  E24DD008  ITBIT 0  SYSCOMP 1  IFEN disabled  DBGRQ no   DBGACK no
  VECTOR   E3A00001  FIQ dis  IRQ dis  D_ABO dis  P_ABO dis  SWI dis  UND dis
  COMCTRL  E5C10000  VERSION 1110  WRITE idle  READ idle
  COMDATA  E3A01002

  Watchpoint 0
  AV       E92D4030
  AM       E24DD008
  DV       E3A00001
  DM       E59F1200
  CV       E5C10000
  CM       E3A01002

  Watchpoint 1
  AV       E92D4030
  AM       E24DD008
  DV       E3A00001
```

```
B::dbgrq
"dbgrq" found.
```

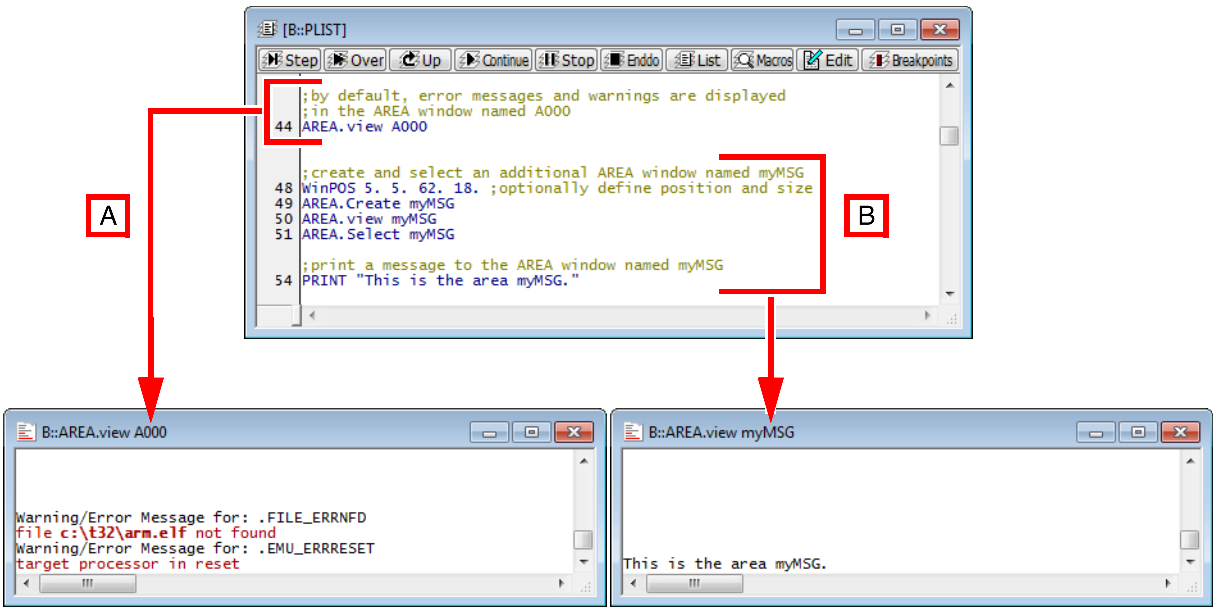
Selection Service

The selection service allows 'drag and drop' and 'cut and paste' features between applications. Drag and drop is started by pressing the left mouse button on a selection and then moving the mouse. Cut and Paste can be done either with the ***Copy*** command in the window manager menu or by using the **Edit** menu or the appropriate accelerator key (i.e. ^C on Windows).

Message Windows

By default, all information is displayed in the message line. To get a more terminal-like output and input, you can create multiple named message areas and display the information output to the various message areas in **AREA** windows. Information is printed to the **AREA** windows with the **PRINT** command. Interactive keyboard input on an **AREA** window can be made with the **ENTER** command.

- Error messages and warnings will always be displayed in the default **AREA** window **A000**. **A000** is the name of the default message area. See [**A**].
- User-defined messages can be output to the same default **AREA** window **A000**, or to extra **AREA** windows having user-defined names, see [**B**].



```
;by default, error messages and warnings are displayed
;in the AREA window named A000
AREA.view A000

;create and select an additional AREA window named myMSG
WinPOS 5. 5. 62. 18. ;optionally define position and size
AREA.Create myMSG
AREA.view myMSG
AREA.Select myMSG

;print a message to the AREA window named myMSG
PRINT "This is the area myMSG."
```

```
Warning/Error Message for: .FILE_ERRNFD
file c:\t32\arm.elf not found
Warning/Error Message for: .EMU_ERRRESET
target processor in reset
```

```
This is the area myMSG.
```

AREA.CLEAR	Clear area
AREA.CLOSE	Close output file
AREA.Create	Create in/out area
AREA.OPEN	Open output file
AREA.RESet	Delete all in/out areas
AREA.SAVE	Save contents of the **AREA** window to file. In this simple save operation, the commands **AREA.OPEN** and **AREA.CLOSE** are not required.
AREA.Select	Select a message area for **PRINT** and **ENTER**
AREA.view	Display in/out area
LOG.toAREA	Log commands by writing them to an **AREA** window
OS.Area	Call host operating system with output in a TRACE32 **AREA** window
PRinTer.Area	Re-route printer output to **AREA** window in specified format

Window Tracking

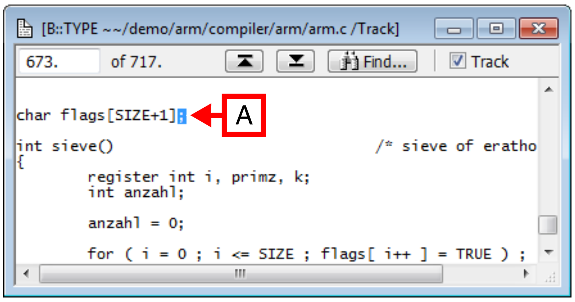
Windows may be coupled by a global reference indicator, generated either by the mouse position within a window or by the result of a search or goto operation. The global reference indicator can be one of the following:

- The line number for text windows, see example 1.
- The address, see example 2.
- Or the absolute time, see example 3.
- Trace record numbers.

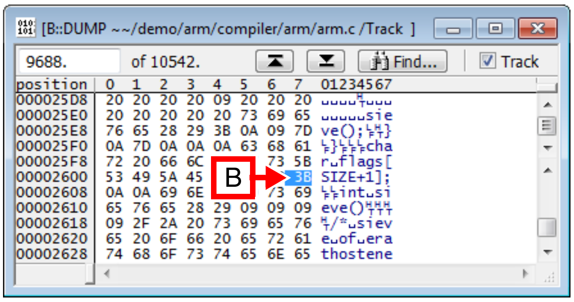
Window tracking is possible between different types of windows, like source text, analyzer listings or timing diagrams. Every window which is set to track mode by the option **/Track** will follow the global reference indicator.

Some windows are temporarily set to tracking when search functions are executed (e.g. the analyzer list window during a find operation).

Example 1 - Tracking in two text windows using the mouse: The cursor position of the mouse pointer [**A**] can be tracked in the other window [**B**], provided path and file name are identical in both windows.



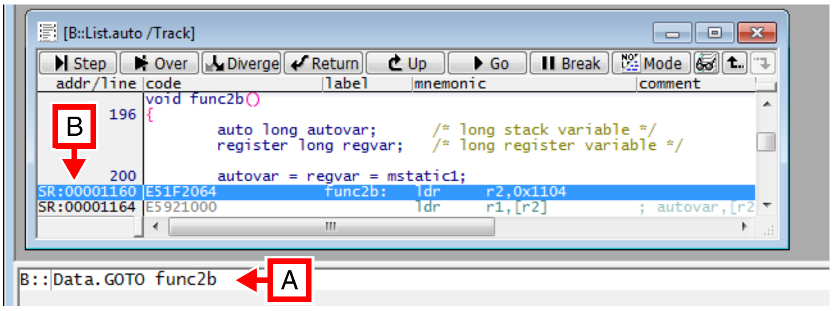
```
char flags[SIZE+1];

int sieve()                                  /* sieve of eratho
{
        register int i, primz, k;
        int anzahl;

        anzahl = 0;

        for ( i = 0 ; i <= SIZE ; flags[ i++ ] = TRUE ) ;
```



```
position  0  1  2  3  4  5  6  7  01234567
000025D8 20 20 20 20 09 20 20 20  ␣␣␣␣ ␣␣␣
000025E0 20 20 20 20 20 73 69 65  ␣␣␣␣␣sie
000025E8 76 65 28 29 3B 0A 09 7D  ve();  }
000025F0 0A 7D 0A 0A 0A 63 68 61   }   cha
000025F8 72 20 66 6C       73 5B  r␣flags[
00002600 53 49 5A 45          3B  SIZE+1];
00002608 0A 0A 69 6E       73 69    int␣si
00002610 65 76 65 28 29 09 09 09  eve()
00002618 09 2F 2A 20 73 69 65 76   /*␣siev
00002620 65 20 6F 66 20 65 72 61  e␣of␣era
00002628 74 68 6F 73 74 65 6E 65  thostene
```

B Tracking pointer

Example 2 - Tracking by going to an address:

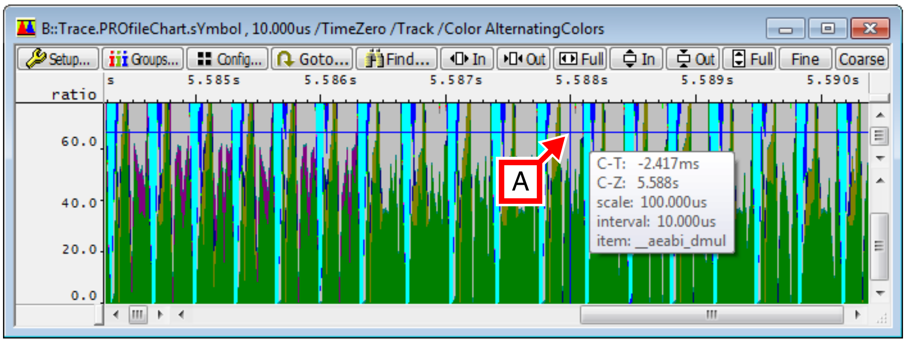


```
 addr/line |code              |label      |mnemonic          |comment
           |void func2b()
       196 |{
           |        auto long autovar;       /* long stack variable */
           |        register long regvar;    /* long register variable */

       200 |        autovar = regvar = mstatic1;
SR:00001160|E51F2064           func2b:     ldr       r2,0x1104
SR:00001164|E5921000                       ldr       r1,[r2]          ; autovar,[r2

B::Data.GOTO func2b
```

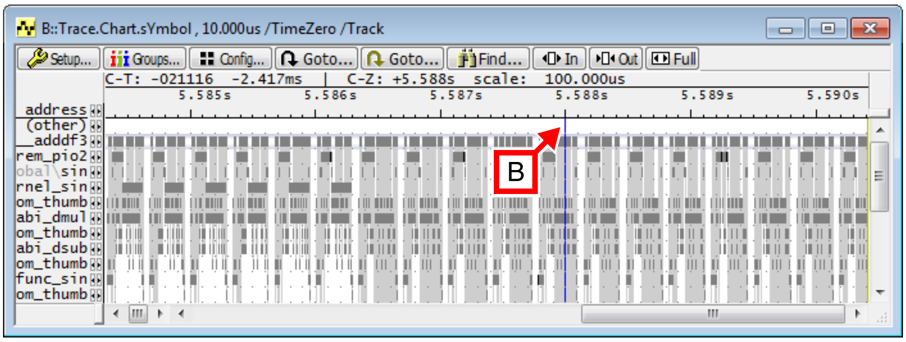
A The **Data.GOTO** command is used to go to the address of `func2b`.

B In the **List.auto** window, the corresponding position is highlighted because of the use of the **Track** option.

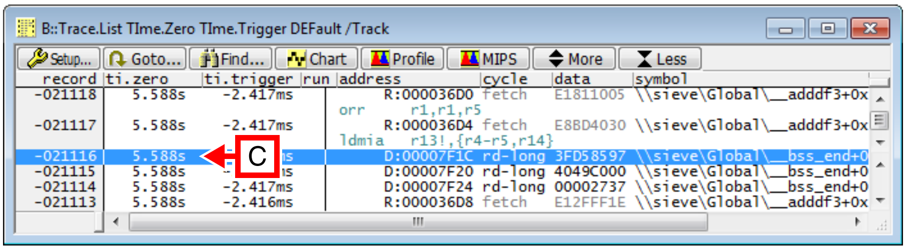
Example 3 - Tracking based on absolute time: After recording trace data, the same data is displayed in three different **Trace.*** windows. Each **Trace.*** window is opened with the **Track** option.



```
C-T:   -2.417ms
C-Z:   5.588s
scale:  100.000us
interval:  10.000us
item:  __aeabi_dmul
```



```
C-T: -021116   -2.417ms  |  C-Z: +5.588s  scale:  100.000us
```



```
 record ti.zero     ti.trigger run address         cycle   data     symbol
-021118    5.588s    -2.417ms            R:000036D0 fetch   E1811005 \\sieve\Global\__adddf3+0x
                              orr      r1,r1,r5
-021117    5.588s    -2.417ms            R:000036D4 fetch   E8BD4030 \\sieve\Global\__adddf3+0x
                              ldmia    r13!,{r4-r5,r14}
-021116    5.588s                        D:00007F1C rd-long 3FD58597 \\sieve\Global\__bss_end+0
-021115    5.588s                        D:00007F20 rd-long 4049C000 \\sieve\Global\__bss_end+0
-021114    5.588s    -2.417ms            D:00007F24 rd-long 00002737 \\sieve\Global\__bss_end+0
-021113    5.588s    -2.416ms            R:000036D8 fetch   E12FFF1E \\sieve\Global\__adddf3+0x
```

A By clicking inside the **Trace.PROfileChart.sYmbol** window, a fine blue graticule marks the cursor position. A tooltip displays more information about the selected position, including the absolute time, here 5.588s.

B A the same time, a fine blue vertical line highlights the corresponding position in the **Trace.Chart.sYmbol** window thanks to the **Track** option.

C The corresponding record is also highlighted in the **Trace.List** window, again thanks to the **Track** option.

File and Folder Operations

TRACE32 provides standard operating system commands for fast execution of file and folder operations. The commands are implemented in the TRACE32 software, they don't execute operating system commands on the host.

ChDir	Change directory
ComPare	Compare files
COPY	Copy file
DEL	Delete file
DIR	List subdirectories and files
DUMP	Display binary file
EDIT	Edit text file in the TRACE32 editor
FIND	Find in text or binary file
LS	Display directory
MKDIR	Create directory
MV	Rename file
PACK	Compress file (with LZW algorithm)
PATCH	Modify binary file
PATH.Set	Define search path
PEDIT *<file>*	Open *<file>* with the PRACTICE script editor
PWD	Change directory
REN	Rename file
RM	Delete file
RMDIR	Delete directory
SETUP.EDITEXT	Define an external editor
TAR	Pack files into an archive without compression
TYPE	Display text file
UNARchive	Extract files from Linux and Microsoft libraries
UNPACK	Expand packed file (with LZW algorithm)
UNZIP	Expand GZIP archive file (with DEFLATE algorithm)
ZIP	Compress files to GZIP archive (with DEFLATE algorithm)

For information about wildcard characters and path prefixes supported with the file and folder handling commands, see **"File Names"**, page 44 and **"Path Prefixes"**, page 45.

File Contents

TRACE32 provides a number of commands for writing data from TRACE32 to file and reading data from files. The following list is a selection of commands:

CLOSE	Close file
Data.WRITESTRING	Write string from target memory to PRACTICE file
OPEN	Open data file
READ	Read data from file
Var.EXPORT	Export variables in CSV format to file
Var.WRITE	Write variables to file
WinPrint	Print window
WRITE	Write data to file
WRITEB	Write binary data to file
APPEND	Append data to file

Encrypt/Execute Encrypted Files

You can encrypt PRACTICE script files (*.cmm) and PER files (*.per) in TRACE32 with user-defined keys. This encryption is useful if you do not want other people to view your source code in human readable form.

Other users can execute any encrypted file (*.cmm or *.per) in TRACE32, provided the encrypted file is unlocked with the same key you have defined for this file.

NOTE:	With the correct key, an encrypted file can be executed in TRACE32, but the source code itself remains encrypted.

PRACTICE script files (*.cmm):

ENCRYPTDO	Encrypt a PRACTICE script file.
DODECRYPT	Execute the encrypted PRACTICE script file. **NOTE**: The PRACTICE script source code itself remains encrypted, i.e. it is *not* human readable.

PER files (peripheral register definition file, *.per):

ENCRYPTPER	Encrypt a PER file.
PER.viewDECRYPT	Execute and view the encrypted PER file in a **PER** window. **NOTE**: The PER file source code itself remains encrypted, i.e. it is *not* human readable.

Text and binary files:

ENCRYPT	Encrypt a text or binary file.
DECRYPT	Decrypt the text or binary file. The text is displayed in human readable form again.

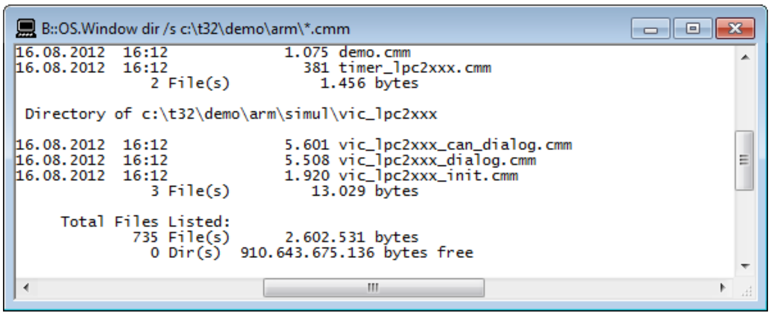
Host Commands

Operations of the host system may be executed directly on the TRACE32 command line.

OS.screen	Execute host command
OS.Area	Call host operating system with output in a TRACE32 **AREA** window
OS.Command	Execute host command
OS.Window	Call host operating system with output in a TRACE32 window
OS.Hidden	Call host operating system without output
OS.OPEN	Open any file type in its default application

Example 1: The TRACE32 commands and functions are formatted in bold. The host command is formatted in regular font.

```
;list all PRACTICE script files (*.cmm) in the TRACE32 ~~/demo/arm/
;folder and all subfolders

LOCAL &files

&files=OS.FILE.ABSPATH(~~/demo/arm/)+"*.cmm"

OS.Window dir /s &files
```



```
16.08.2012  16:12             1.075 demo.cmm
16.08.2012  16:12               381 timer_lpc2xxx.cmm
               2 File(s)          1.456 bytes

 Directory of c:\t32\demo\arm\simul\vic_lpc2xxx

16.08.2012  16:12             5.601 vic_lpc2xxx_can_dialog.cmm
16.08.2012  16:12             5.508 vic_lpc2xxx_dialog.cmm
16.08.2012  16:12             1.920 vic_lpc2xxx_init.cmm
               3 File(s)         13.029 bytes

     Total Files Listed:
             735 File(s)      2.602.531 bytes
               0 Dir(s)  910.643.675.136 bytes free
```

Example 2: This script line opens a *.csv file in your favorite spreadsheet application.

```
OS.OPEN ~~/demo/etc/trace/export.taskevents/temp.csv
```

Printer Operations

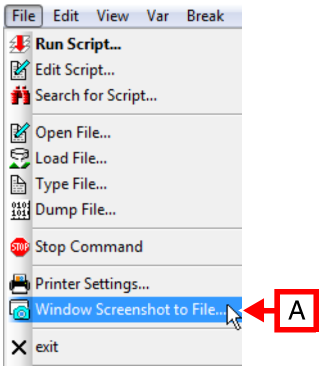
You can send every window or the complete screen from TRACE32 to:

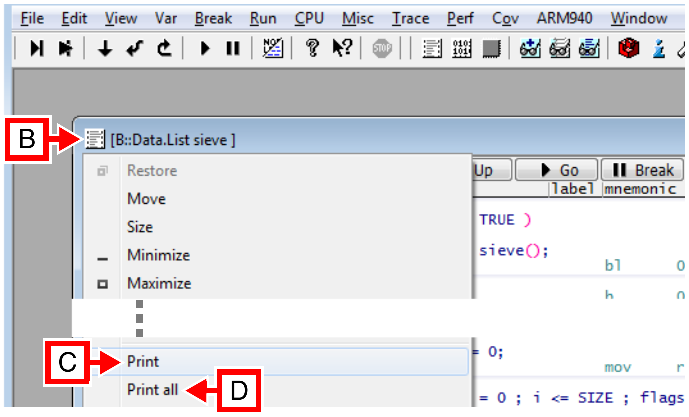
- The default printer
- The clipboard
- A file
- The default **AREA** window **A000**

For each output medium, you can define the format, e.g. font, font size, ASCII, enhanced ASCII, XML, or a more complex format, like POSTSCRIPT or WORDSTAR. When printing to file, you can specify path and file name or browse for an existing file.

You have the following options to send information from TRACE32 to a printer or save TRACE32 windows to file:

- Choose **File** menu > **Window Screenshot to File** to capture the TRACE32 main window and all other TRACE32 windows displayed within the TRACE32 main window [**A**].
- Click the top left icon in any window to open the window manager menu [**B**].





  - The **Print** option prints just the visible contents of the active window [**C**].
  - The **Print all** option prints more than the visible contents of the active window [**D**].

- Use the TRACE32 command line and PRACTICE scripts (*.cmm); commands for printing and saving TRACE32 windows to file are listed in the table below.
- Extra commands are provided for saving the code coverage database and the instruction statistics database to XML files.

PRinTer.select	Select printer type and output style
PRinTer.FILE	Define file for single printer output and select output format for file (ASCII, CSC, XML, etc.)
PRinTer.OPEN	Open file and re-route multiple printer outputs to this file
PRinTer.CLOSE	Close file after multiple printer outputs

PRinTer.EXPORT	Export CSV-formatted printer output to file
PRinTer.HardCopy	Print all windows on screen
PRinter.SIZE	Define layout
PRinter.OFFSET	Define left and top border
WinPRT *<window_name>*	Prints just the visible contents of the active window
WinPrint.*<command>*	The **WinPrint** *pre*-command prints more than the visible contents of the active window
ISTATistic.EXPORT	Export instruction statistics to an XML file
COVerage.EXPORT	Export code coverage information to an XML file

Examples:

```
; example for print operation
PRinTer.select      IBM            ; select IBM printer
PRinter.SIZE        80. 65.        ; select lines and columns
PRinter.OFFSET       5.  5.        ; select border
WinPrint.Data.dump  0x0--0xfff     ; print window
```

```
; example for copy to file operation
PRinTer.FILE       test.lst        ; open file for printing
WinPrint.Data.dump 0x0--0xfff      ; print Data.dump window to file
PRinTer.select     IBM             ; switch back to line printer
```

```
; example for generating a POSTSCRIPT file
PRinTer.FILE test.ps  PSPS12       ; open file for printing and select
                                   ; POSTSCRIPT, Portrait,
                                   ; Helvetica, 12cpi
WinPrint.Data.dump 0x0--0xfff      ; print Data.dump window to file
```

System Setup and Configuration

Many system configuration options are set with the **SETUP** command. For more information refer to the **“PowerView Command Reference”** (ide_ref.pdf) and the manual of the devices.

SETUP.ASCIITEXT	Configure ASCII text display
SETUP.BAKfile	Set backup file mode
SETUP.COLOR	Configure colors
SETUP.DEVNAME	Set logical device name
SETUP.EDITOR	Configure the TRACE32 editors
SETUP.EDITEXT	Define an external editor
SETUP.EXTension	Set default file name extensions
SETUP.HOLDDIR	Configure working directory
SETUP.ICONS	Display icons in popup menus
SETUP.QUITDO	Define a PRACTICE file that is executed automatically when you quit a TRACE32 session
SETUP.ReDraw	Update whole screen
SETUP.SOUND	Set sound generator mode
SETUP.TabSize	Configure tab width
SETUP.TIMEFORM	Select scientific time format
SETUP.URATE	Limit window update rate
SETUP.WARNSTOP	Configure PRACTICE stops

ZERO	Set time reference

Logging Commands

You can log the command inputs and the call hierarchy of PRACTICE scripts (*.cmm) with the commands listed below.

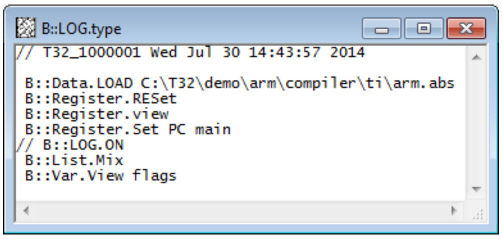
The logging of the command input generates a file which has the structure of a PRACTICE script (*.cmm). This file can be edited and started with the **DO** command. The command log includes all commands entered in the TRACE32 command line and all mouse commands. Every operation on TRACE32 can be referred to a single-line command. The mouse click to a screen-based button will be stored as a single line command. Command inputs which lead to a syntax error are not logged.

To generate a command log, the log file must be opened first. Then all executed commands are written to this file. There is no limitation by an internal buffer size. The file can be viewed in a window, while it is being filled. By closing the file the logging process is terminated. Only one file may be opened at the same time. The logging may be interrupted temporarily by an OFF and ON sub-command.

LOG.OPEN	Generate command log file and start logging
LOG.CLOSE	Terminate logging and close command log file
LOG.type	Show contents of the command log file
LOG.ON	Resume logging
LOG.OFF	Pause logging
LOG.toAREA	Log commands by writing them to an AREA window
LOG.DO	Log the call hierarchy of PRACTICE scripts (*.cmm)
	For more information, see **“Logging the Call Hierarchy of PRACTICE Scripts”** (practice_user.pdf).

Example:

```
LOG.OPEN                        ; opens file 't32.log'
;...                            ; commands are logged

LOG.OFF                         ; switch off log function
;...                            ; commands are not logged

LOG.ON                          ; switch on log function
;...                            ; commands are logged
LOG.CLOSE                       ; close file and terminate log function
```



```
// 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
```

Dialog Programming

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**

In this section:

- Dialog Syntax and File Types
- Comments in Dialogs
- Dialog Commands - an Overview
- Dialog Elements - an Overview
- Return Values and Labels
- PRACTICE Macros in Dialog Programming

For information about built-in and user-defined icons, see **"Built-in Icons and Icon Library"**, page 118.

Dialog Syntax and File Types

The syntax of a dialog definition is line oriented. Blanks and empty lines can be inserted to structure and indent the dialog definition. Single and multi-line programs can be assigned to dialog elements.

Single-line scripts are enclosed in straight quotation marks `"..."`; multi-line scripts are enclosed in parentheses `(...)`.

Single-line script Click Me

```
BUTTON "Click Me"  "Data.List"
```

Multi-line script Click Me

```
BUTTON "Click Me"
       (
         Data.List
         Register.view
       )
```

The opening parenthesis of a multi-line script *must* immediately follow after a dialog element. If an empty line is erroneously inserted after a dialog element, the TRACE32 message bar displays the error message `nesting block open missing`. This error message is displayed when you try to execute the defective dialog.

There are two file types where you can store custom dialogs. The syntax slightly varies depending on the file type you have chosen:

1. Embedded in PRACTICE script files with the extension ***.cmm**. The dialog definition is placed in parentheses after the **DIALOG** command. See example 1.
2. In extra files with the extension ***.dlg**. They are called by the **DIALOG.view** command. See example 2.

Example 1: The dialog is embedded in a PRACTICE script file with the extension *.cmm:

```
LOCAL &file                     ;declare PRACTICE macro
        DIALOG.view             ;start the dialog definition
        (
         POS 1. 1. 15.
myLabel: EDIT "" ""
         POSX 1.  6.
         BUTTON "[:edit]File"  "DIALOG.SetFile.SAVE myLabel ~~~\*.cmm"
         POSY 1.
         DEFBUTTON "OK" "CONTinue"
        )
STOP                            ;wait for the user's response to the dialog
&file=DIALOG.STRing(myLabel) ;get return value of EDIT box by label
DIALOG.END

OPEN  #1 &file /Create
WRITE #1 "Begin of file"
CLOSE #1
ENDDO
```

Example 2: The dialog is in an extra file with the extension *.dlg:

PRACTICE script file (*.cmm)

```
LOCAL &file

DIALOG.view dialog.dlg




STOP
&file=DIALOG.STRing(LAB)
DIALOG.END

OPEN  #1 &file /Create
WRITE #1 "Begin of file"
CLOSE #1
ENDDO
```

Contents of dialog.dlg

```
     POS 1. 1. 10.
LAB: EDIT "" ""
     POS 11. 1. 5.
     BUTTON "File"
    (
     DIALOG.SetFile.SAVE LAB *.cmm
     )
     POS 1. 3. 5.
     DEFBUTTON "OK" "CONTinue"
```

Comments in Dialogs

Comment lines start with a semicolon and must be placed in separate lines.

Correct comment position

```
DIALOG.view
(
 ;your comment
  ICON ":objects"
 ;your comment
  TEXT "Hello World!"
)
```

Wrong comment position

```
DIALOG.view
(
 ICON ":objects"     ;your comment
 TEXT "Hello World!" ;your comment
)
```

If a comment is erroneously placed in the same line as a dialog element, the TRACE32 message bar displays the error message `no more arguments expected`. This error message is displayed when you try to execute the defective dialog.

Dialog Commands

Using the **DIALOG** command group you can (a) control your custom dialogs, (b) control the behavior of an individual dialog element on a custom dialog, (c) interact with the file system of the operating system (OS), and (d) display OS message boxes.

Control Your Custom Dialogs

DIALOG.AREA	Adds an output **AREA** to a custom dialog
DIALOG.END	Close the dialog window
DIALOG.Program	Editor to write a custom dialog.
DIALOG.ReProgram	Dialog programming
DIALOG.SELect	Programmatically focus on this dialog
DIALOG.view	Show dialog window

Control Behavior of Individual Dialog Elements on Custom Dialogs

DIALOG.Disable	Disable dialog elements
DIALOG.Enable	Enable dialog elements
DIALOG.EXecute	Execute a dialog button
DIALOG.Set	Set the value of a dialog element

Interact with the File System

DIALOG.DIR	Display a folder picker dialog and pass the return value of the selected folder to your PRACTICE script (*.cmm).
DIALOG.File	Open an OS file dialog and pass the name of the selected file to your PRACTICE script (*.cmm).
DIALOG.SetDIR	Browse for folder. The selected folder an be displayed in the **EDIT** box of your custom dialog.
DIALOG.SetFile	Open an OS file dialog and pass the name of the selected file to a custom dialog. The selected file can be displayed in the **EDIT** box of your custom dialog.

Display Message Boxes of the Operating System

DIALOG.MESSAGE	Create dialog box with an information icon (OK button only)
DIALOG.OK	Create dialog box with an exclamation mark (OK button only)
DIALOG.YESNO	Create dialog box with YES and NO buttons

Dialog Elements

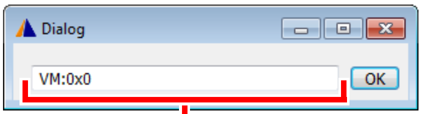
Dialog elements allow you to place edit boxes, buttons, drop-down lists, etc. on your custom dialogs. TRACE32 provides the following dialog elements for programming custom dialogs:

BAR	Define a progress bar
BOX ["*<text>*"]	Define a decorative border
BUTTON "*<text>*" [*<command>*]	Define a button
CHECKBOX "*<text>*" [*<command>*]	Define a check box
<label> **CHOOSEBOX** "*<text>*" [*<command>*]	Define a choose box
CLOSE [*<command>*]	Catch window close
COMBOBOX "*<list_items>*" [*<command>*]	Define a combo box
DEFBUTTON "*<text>*" [*<command>*]	Define the default button
DEFCOMBOBOX "*<list_items>*" [*<command>*]	Define a combo box
DEFEDIT "*<initial_text>*" [*<command>*]	Define an edit control
DEFHOTCOMBOBOX "*<list_items>*" [*<command>*]	Define a default hot combo box
DEFHOTEDIT "*<initial_text>*" [*<command>*]	Define a hot edit control
DEFMEDIT "*<initial_text>*" [*<command>*]	Define a default multiline edit control
DLISTBOX "*<list_items>*" [*<command>*]	Define a default list box
DYNAMIC "*<initial_text>*"	Define a dynamic, single-line area
DYNCOMBOBOX "*<list_items>*" [*<command>*]	Define a dynamic combo box
DYNDEFCOMBOBOX "*<list_items>*" [*<command>*]	Define a default dynamic combo box
DYNDEFHOTCOMBOBOX "*<list_items>*" [*<command>*]	Define a dynamic default hot combo box
DYNHOTCOMBOBOX "*<list_items>*" [*<command>*]	Define a dynamic hot combo box
DYNLTEXT "*<initial_text>*"	Define a dynamic single-line text area in bold and large font size
DYNPULLDOWN "*<list_items>*" [*<command>*]	Define a dynamic pull-down list
DYNTEXT "*<initial_text>*"	Define a dynamic, single-line text area in regular font size
EDIT "*<initial_text>*" [*<command>*]	Define an edit control
HEADER "*<text>*"	Define window header
HELP *<name>*	Define a help icon
HOTEDIT "*<initial_text>*" [*<command>*]	Define a hot edit control

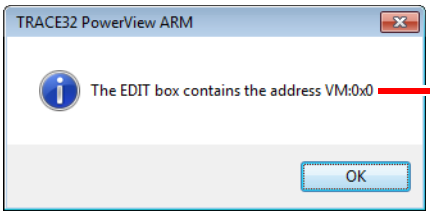
HOTCOMBOBOX "*<list_items>*" [*<command>*]	Define a hot combo box
ICON "*<built_in_icon_name>*" | "*<user_defined_icon>*"	New icon in top left corner of dialog
INFOTEXT "*<msg_text>*" [*<formatting>*]	Define a multiline info text box on a dialog
LEDIT "*<initial_text>*" [*<command>*]	Define an edit control
LINE "*<text>*"	Define a decorative line
LISTBOX "*<list_items>*" [*<command>*]	Define a list box
LTEXT "*<text>*"	Static, single-line text area in bold and large font size
MEDIT "*<initial_text>*" [*<command>*]	Define a multiline edit control
MLISTBOX "*<list_items>*" [*<command>*]	Define a multiline list box
NAME "*<text>*"	Define an internal dialog name
POS *<x> <y> <width> <height>*	Define position and size
POSX *<increment> <width> <height>*	Define position and size on the x-axis
POSY *<increment> <width> <height>*	Define position and size on the y-axis
PULLDOWN "*<list_items>*" [*<command>*]	Define a static pull-down list
SPACE	Define space
STATIC "*<built_in_icon_name>*" | "*<user_defined_icon>*"	Place an icon in a dialog
TEXT "*<text>*"	Define a text item
TEXTBUTTON "*<text>*" [*<command>*]	Define a flat button with text only
TREEBUTTON "" [*<command>*]	Define a +/- toggle button
UPDATE ["*<command_string>*"] [*<update_interval>*]	Executes commands periodically
VLINE ""	Define a decorative vertical line

Return Values and Labels

Dialog elements, such as an **EDIT** box or a **LISTBOX**, can have a user-defined label in front of the command. Labels must start in the first column and are always followed by a colon. Together with the **DIALOG.STRing()** or **DIALOG.STRing2()** function, a label can be used to access the return value of a dialog element.



&retVal=DIALOG.STRing(*<label>*)

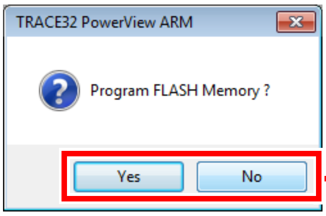


The EDIT box contains the address VM:0x0

&retVal

```
LOCAL &retVal                  ;declare a PRACTICE macro

DIALOG.view                    ;start the dialog definition
(
          POS 1.5    0.75   30.
myLAB:    EDIT "x" ""

          POSX 1.   5.
          DEFBUTTON "OK" "CONTinue"
)
STOP                           ;wait for the user's response to the dialog

&retVal=DIALOG.STRing(myLAB) ;get return value by label

DIALOG.END
DIALOG.MESSAGE "The EDIT box contains the address &retVal"
```

The return values of built-in dialog boxes, e.g. the **DIALOG.YESNO** message box or the **DIALOG.DIR** folder picker dialog, can be accessed with the **ENTRY** command. Here is an example of a simple “yesno” input:



ENTRY &*<practice_macro>*

```
LOCAL &result                  ;declare a PRACTICE macro

DIALOG.YESNO "Program FLASH Memory ?"
ENTRY &result                  ;get return value of DIALOG.YESNO

IF !&result
    ENDDO
ELSE
(
   ;your code...
)
```

PRACTICE Macros inside Dialog Definitions

Two PRACTICE macros, highlighted in blue, are used in the following dialog definition. For activating PRACTICE **macro expansion inside a DIALOG definition**, the following prerequisites have to be fulfilled:

1. “(&” must be used - instead of just “(“
2. “**(&**” must begin in the **first column** of the line

```
      ENTRY &flashno &default_flash_firmware_file
      LOCAL &file    &header_text
      &header_text="program dialog for "+FORMAT.Decimal(1,&flashno)
      &header_text="&header_text"+". flash"

      DIALOG.view
(&
      HEADER "&header_text"
      POS 1. 1. 30.
LAB:  EDIT "&default_flash_firmware_file" ""
      POS 31. 1. 5.
      BUTTON "File"
      (
           DIALOG.SetFile LAB *.bin
      )
      POS 1. 3. 5.
      DEFBUTTON "OK" "CONTinue"
)
      STOP
      &file=DIALOG.STRing(LAB)
      PRINT "selected firmware file: &file"
      PRINT "for flash:"+FORMAT.Decimal(1,&flashno)
      DIALOG.END
      ENDDO
```

See also: **“Switching PRACTICE Macro Expansion ON or OFF”** (practice_user.pdf)

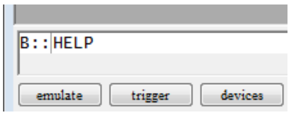
HELP System

In this section:

Ways to Get Help

There are several methods to get help information:

- **Help** menu in the menu bar
- Context-sensitive help
- **HELP** via the TRACE32 command line



```
B::HELP
```

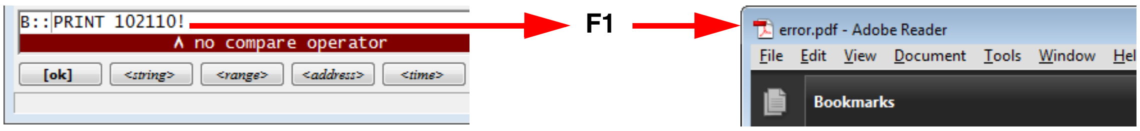
- The following **HELP** commands are available:

HELP.Bookmark	Bookmark PDF files
HELP.checkUPDATE	Enable the automatic help update via internet
HELP.command	Command related support
HELP.FILTER	Filtering all documents with the used hardware and software
HELP.Find	Full-text search across all help documents
HELP.Index	Search within indexed terms, commands, functions
HELP.PDF	Open PDF file in a PDF viewer
HELP.PICK	Context-sensitive help
HELP.PRinT	Print PDF files
HELP.Topics	Display help content list
HELP.TREE	Display command tree

Context-Sensitive Help

You can call up the **HELP** window via the help key. On Windows, the help key is **F1**. The **HELP** window then displays information about the current context, with the current context being determined by the cursor position.

- To get context-sensitive help on a window or dialog, click the window or dialog, and then press **F1**.
- To get context-sensitive help for a command, type the command at the TRACE32 command line, append an empty space, and then press **F1**.
- To get context-sensitive help on an individual item such as a button, check box, or menu item, click the context help button on the toolbar (**HELP.PICK**).
- If an error occurs, a short error help message will be displayed.
  Press **F1** to access the complete error help message in the error.pdf.



```
B::PRINT 102110!
          ∧ no compare operator
```

Structure of the Help System

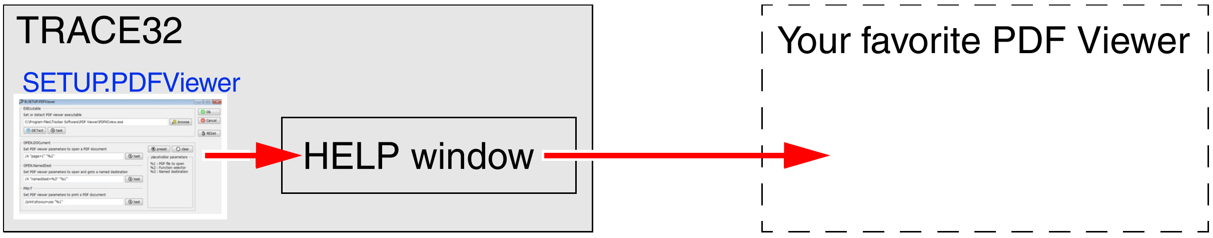
The TRACE32 help system is divided in two parts:

- 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.

New TRACE32 Releases

[Software Releases 02/2016 and higher]

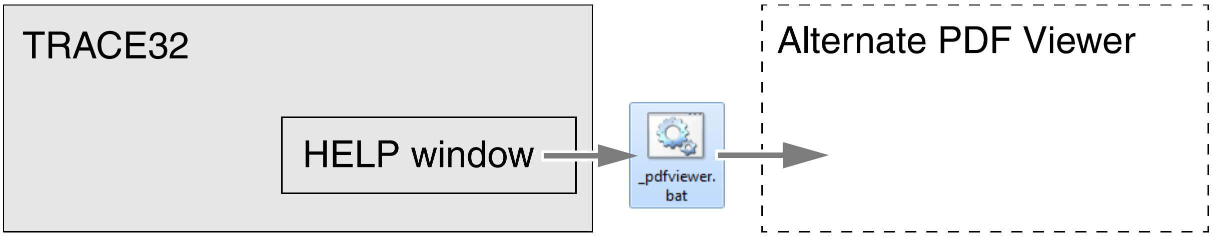
Your favorite PDF viewer: It takes only a few mouse-clicks to configure the TRACE32 help system to relay context-sensitive help calls to your favorite PDF viewer.



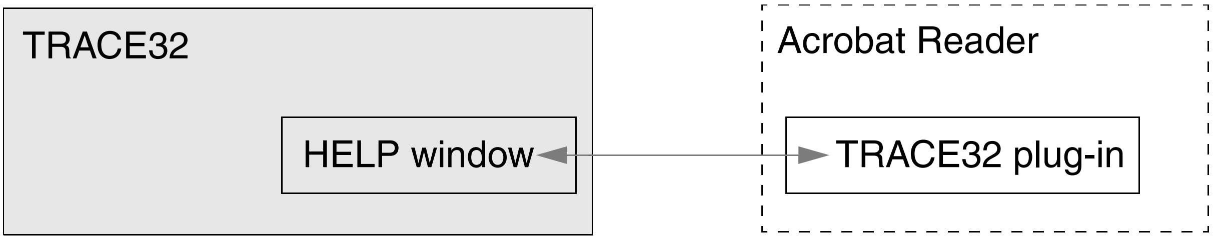
For a step-by-step procedure, see **"Configure the Help System"**, page 94.

Previous TRACE32 Releases

Alternate PDF viewers: TRACE32 relays context-sensitive help calls to a batch file, which then calls the desired topic in the PDF file. The script is a *.bat file under Windows, or an*.sh file under Linux and MacOS.



Acrobat Reader: TRACE32 communicates with the TRACE32 plug-in to jump directly to the desired topic in the PDF file.



Configure the Help System

[Recommendations] [Software Releases 02/2016 and higher]

This section describes how to proceed after you have successfully installed the TRACE32 software and the help system.

Upon completion of the installation:

- Each TRACE32 executable (t32m<*architecture*>.exe) provides the **HELP** window.
- The file help.t32, which has to be in the system path (e.g. c:\t32), enables all help functions in TRACE32, like context-sensitive help and full-text search. When TRACE32 is started, the file help.t32 is loaded. If not, you receive an error message, saying that the help.t32 file cannot be loaded.
- The PDF help files are in the subfolder **pdf** of the TRACE32 system path (e.g. c:\t32\pdf). This path can be changed in the config file.

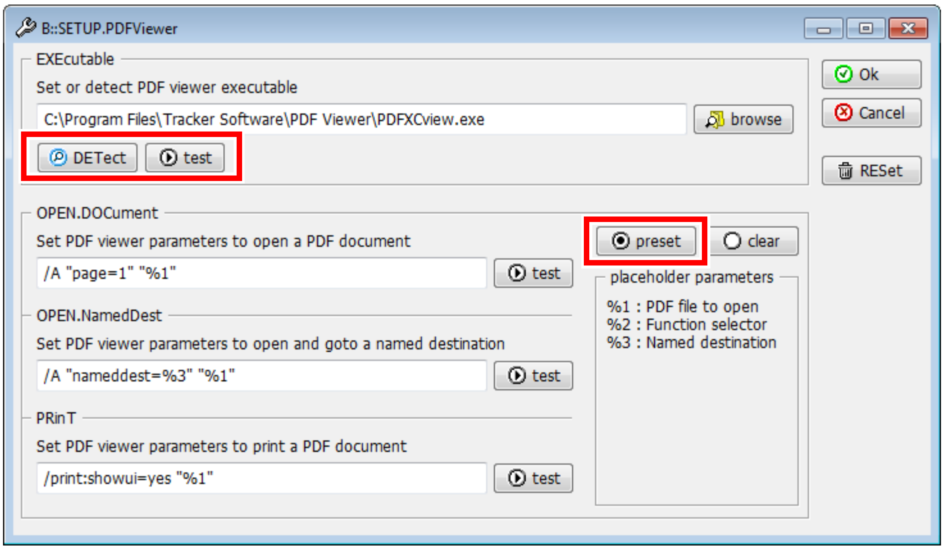
Your next step:

- Configure the TRACE32 help system with a few mouse-clicks to display the PDF help files in your favorite PDF viewer; see step-by-step procedure below.

To configure the TRACE32 help system:

1. Choose **Help** menu > **Setup PDF Viewer**, or at the TRACE32 command line, type: **SETUP.PDFViewer**

   The **SETUP.PDFViewer** dialog window opens.



B::SETUP.PDFViewer

EXEcutable
Set or detect PDF viewer executable
C:\Program Files\Tracker Software\PDF Viewer\PDFXCview.exe

OPEN.DOCument
Set PDF viewer parameters to open a PDF document
/A "page=1" "%1"

OPEN.NamedDest
Set PDF viewer parameters to open and goto a named destination
/A "nameddest=%3" "%1"

PRinT
Set PDF viewer parameters to print a PDF document
/print:showui=yes "%1"

placeholder parameters
%1 : PDF file to open
%2 : Function selector
%3 : Named destination

2. Do one of the following:

   - Click **DETect** if you want to use your default PDF viewer.
     The remaining input boxes are populated with the pre-configured command line parameters for the selected PDF viewer.
   - Click **browse** if you want to user a PDF viewer other than the default, e.g. a portable PDF viewer.

Then click **preset** to populate the remaining input boxes with the pre-configured command line parameters for the selected PDF viewer.

3. Click **test** to verify that the selected PDF viewer can be started from within TRACE32.

4. Click the remaining three **test** buttons to verify that your selected PDF viewer passes the following basic tests:

   - The PDF viewer opens our test document on page 1.

   - The PDF viewer jumps to a named destination on another page in the same test document.

   - The PDF viewer prints our test document or opens the **Print** dialog.

5. If the selected PDF viewer has passed all tests, click **Ok**.

6. **Optional test** - online help call via the TRACE32 command line:

   - Type the following command at the TRACE32 command line: `List.Mix`

   - Add a space, and then press **F1**.
     Result: TRACE32 help system displays the description of the **List.Mix** command in your favorite PDF viewer.

NOTE:	You do **not** need to re-start TRACE32 because your settings take immediate effect.

Recommendations for Choosing a PDF Viewer

1. You should choose a PDF viewer for use with the TRACE32 help system that provides the following features:

   - Tabbed document view for files opened via the command line.

   - Command line argument for passing file names to the PDF viewer (e.g. debugger_arm.pdf).

   - Command line argument for passing named destinations to the PDF viewer (e.g. line IDs).

   - One and the same PDF viewer instance allows an unlimited number of context-sensitive jumps to named destinations within one and the same PDF file instance.

   - A **Back** button that allows you to re-trace your navigation steps across PDF documents, and not just the navigation steps within the same PDF document.

   - A PDF viewer that is quick to start.

2. Install the latest version of the PDF viewer, in which you want to display the files of the TRACE32 help system.

Bookmarks for Help Topics

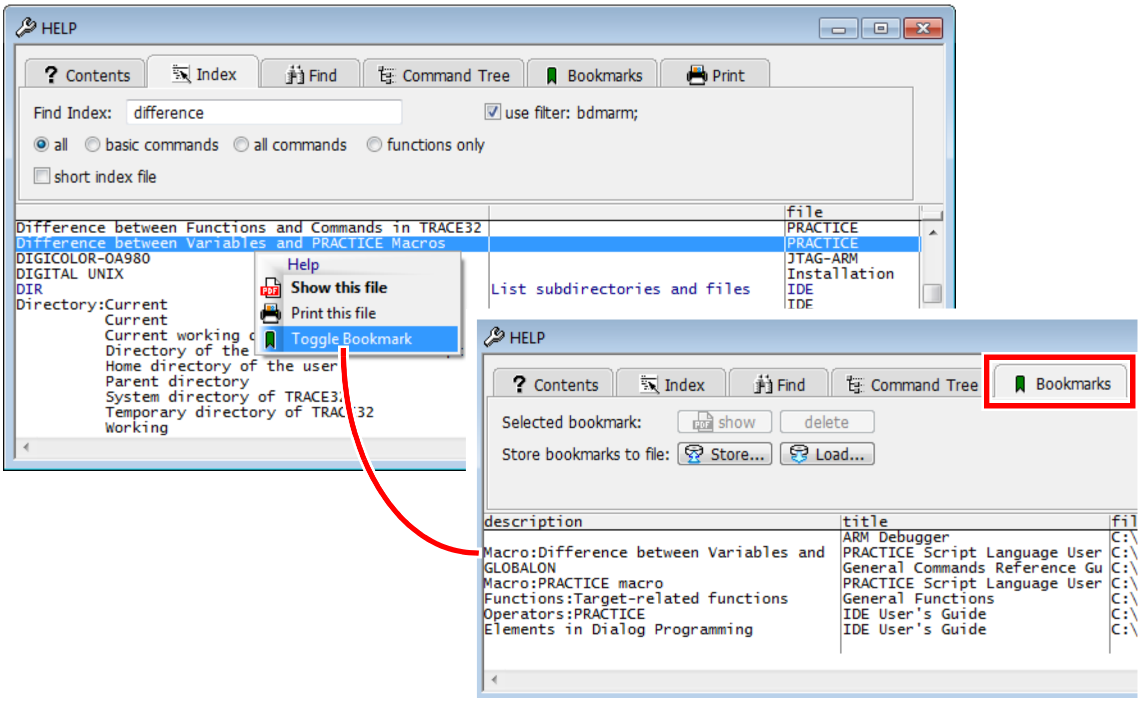
In this section:

- Create Help Bookmarks
- Store and Load Help Bookmarks Manually
- Store and Load Help Bookmarks Automatically

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. See **“Store and Load Help Bookmarks Automatically”**, page 97.

Create Help Bookmarks

- Right-click any topic in the **HELP** window, and then select **Toggle Bookmark**.
  - A green rectangle indicates the bookmarked help topic.
  - The help bookmark itself is added to the **Bookmark** tab of the **HELP** window.



```
                                                                     file
Difference between Functions and Commands in TRACE32                 PRACTICE
Difference between Variables and PRACTICE Macros                     PRACTICE
DIGICOLOR-OA980                                                      JTAG-ARM
DIGITAL UNIX                                                         Installation
DIR                                        List subdirectories and files IDE
Directory:Current                                                    IDE
          Current
          Current working d
          Directory of the
          Home directory of the user
          Parent directory
          System directory of TRACE32
          Temporary directory of TRACE32
          Working
```

```
description                                  title                           fil
                                             ARM Debugger                    C:\
Macro:Difference between Variables and       PRACTICE Script Language User   C:\
GLOBALON                                     General Commands Reference Gu   C:\
Macro:PRACTICE macro                         PRACTICE Script Language User   C:\
Functions:Target-related functions           General Functions               C:\
Operators:PRACTICE                           IDE User's Guide                C:\
Elements in Dialog Programming               IDE User's Guide                C:\
```

Store and Load Help Bookmarks Manually

Use the steps described below if you want to transfer your bookmarks from one computer to another computer.

To store help bookmarks:

1. On the **Bookmark** tab of the **HELP** window, click **Store**.
2. Enter a file name, and then click **Save**.
   The bookmarks displayed on the **Bookmarks** tab are saved as a PRACTICE script (*.cmm).

To load help bookmarks:

1. On the **Bookmark** tab of the **HELP** window, click **Load**.
2. Browse for the PRACTICE script (*.cmm) containing the bookmarks.
3. Click **Open** to load the help bookmarks into the **Bookmark** tab.

Store and Load Help Bookmarks Automatically

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

1. Close TRACE32.
2. Add the **AutoSTOre** command to your PRACTICE start-up script (*.cmm):

   ```
   AutoSTOre , HELP
   ```

   If the **AutoSTOre** command is already used in your start-up script, then add just the keyword `HELP` as shown in the example below.

   ```
   AutoSTOre , HISTory HELP
   ```

3. Restart TRACE32.

   The help bookmarks you create are now automatically stored when you close TRACE32. In addition, the bookmarks are automatically loaded back into the **Bookmark** tab of the **HELP** window when you start TRACE32 again.

This script line returns the path and file name where TRACE32 auto-stores your help bookmarks:

```
PRINT VERSION.ENVironment(AUTOSTORE)
```

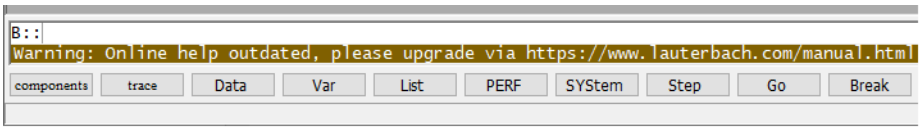
[Software Releases 02/2016 and higher]

Loads only old Online Help

Verify if the file help.t32 is in the TRACE32 system path (by default c:\t32), and if you have rights to read this file.

Warning: Online help outdated, please upgrade via https://www.lauterbach.com/manual.html

Situation: The TRACE32 message line displays this warning message:



```
B::
Warning: Online help outdated, please upgrade via https://www.lauterbach.com/manual.html
```

Cause: The TRACE32 help system is at least 2 software releases older than the TRACE32 executable (t32m*<architecture>*.exe).

Remedy:

1. Open the **VERSION.ENVironment** window, and then make a note of the paths shown in the lines **`SYS:`** and **`HELP:`**
2. Close TRACE32.
3. Download the zipped help system of the most recent TRACE32 software release from **www.lauterbach.com/manual.html**
4. Unzip the downloaded file.
5. Copy the pdf files to the **`HELP:`** folder.
6. Copy the help.t32 file to the **`SYS:`** folder, i.e. the TRACE32 system directory.
7. Restart TRACE32. The TRACE32 help system is now up to date again.

Change the Installation Path of the PDF Files

The PDF files of the TRACE32 help system are installed to the TRACE32 system path, subfolder **pdf**. But sometimes it may be necessary to change this path - for example, if you want different TRACE32 installations to share the same HELP= path.

There are two possibilities to change the installation path for the PDF files:

1. Add it to the configuration file in the OS= part:

```
OS=
SYS=c:\t32          ; system directory for TRACE32
TMP=c:\tmp          ; temporary directory for TRACE32
HELP=c:\t32\pdf     ; help directory for TRACE32 (default: c:\t32\pdf)
```

2. Set the environment variable “T32HELP” to the pdf installation path.

NOTE:	The help directory for the PDF files can be a local folder or a network folder, e.g. g:\trace32-help-files\pdf The file help.t32 must reside in the system directory. A network folder is **not** supported.

Winhelp Compatibility

To provide backward compatibility, the main Winhelp functions will still work.

On unix, additionally the environment variable “HHHOME” has to be set to the directory for hyperhelp (used for displaying the online manual).

Winhelp Files:

man.t32	Online manual (error messages)
man.*	Online manual (WINDOWS help)
manhh.*	Online manual (UNIX hyperhelp)

Previous Releases - HELP System

Previous Releases - HELP Installation and Setup

[up to Software Release 09/2015]

The installation of the help system is normally done by the software installation program, but here the complete online help installation is described if any problem occurred:

1. The **HELP** window is included in the TRACE32 executable (t32*.exe).

2. TRACE32 help loads the file help.t32, which has to be in the system path, e.g. C:\T32\
Only this file enables all help functions in TRACE32, like context-sensitive help and full-text search.

3. Acrobat Reader should be installed on the computer, and to use the TRACE32 plug-in, the version has to be 4.0 or higher.

4. Acrobat loads the TRACE32 plug-in (trace32.api) which has to be in the “plug_ins” directory. If the plug-in is loaded correctly, you can find the menu entry **About TRACE32** in the **Help** menu.

5. The PDF help files are in the TRACE32 system path in the subfolder “pdf”, e.g. “C:\T32\pdf”. This path can be changed in the config file.

6. On Unix you have to do manually:

   The environment variable “ACROBAT_PATH” has to be set to the path where acroread is installed, Use the setenv command or add it to your .profile - file.

   ```
   >setenv ACROBAT_PATH=/opt/Acrobat5
   ```

   Copy the TRACE32 plug-in to the Acrobat plug_ins folder

   ```
   >cp cdrom/bin/suns/trace32.api
   /opt/Acrobat5/Reader/sparcsol/plug_ins/
   ```

Previous Releases - Configuring an Alternate PDF Viewer

[only Software Releases 09/2014, 02/2015, and 09/2015]

By default, the help system of TRACE32 uses Adobe Reader as PDF viewer. But, as of the release in November 2014, the help system of TRACE32 supports any PDF viewer that can handle file names and named destinations.

Please consult the help of the PDF viewer you want to use about how to pass file name and named destination as command line arguments to that PDF viewer; for some examples, see below.

The following step-by-step procedure assumes that you have installed TRACE32 in the default system directory `c:\t32` on Windows, or `$HOME/t32` on Linux.

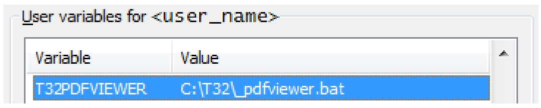
To configure an alternate PDF viewer for the help system of TRACE32:

1. Close TRACE32.

2. Create an OS environment variable called `T32PDFVIEWER`

3. Assign the following value to the variable:

   - **For Windows users**: `c:\t32\_pdfviewer.bat`
     The resulting entry in the Windows **Environment Variables** dialog looks as follows:

     
     User variables for <user_name>

Variable	Value
T32PDFVIEWER	C:\T32_pdfviewer.bat

   - **For Linux and Mac users**: `$HOME/t32/_pdfviewer.sh`
     The resulting entry in the Linux `$HOME/.profile` file looks as follows:

```
  export T32PDFVIEWER=$HOME/t32/_pdfviewer.sh
```

4. To make the new OS environment variable and its value available to TRACE32, log out of your Windows or Linux session, and then log in again.

5. In the TRACE32 system directory, create the file `_pdfviewer.bat` or `_pdfviewer.sh`

6. **For Linux users**: Make sure that you have execute permission for the script file, e.g.
   `chmod +x _pdfviewer.sh`

7. Enter a script which calls the PDF viewer you want to use and passes file names and named destinations as arguments from TRACE32 to your PDF viewer:

   - Examples for Windows users

   - Example for Linux users

8. Start TRACE32.

9. To test the alternate online help call, type the following command at the TRACE32 command line:
   `List.Mix`

10. Add a space, and then press **F1**. **Result**: TRACE32 help system displays the description of the **List.Mix** command in your favorite PDF viewer.

Previous Releases - Examples for Windows and Linux Users

[only Software Releases 09/2014, 02/2015, and 09/2015]

The argument `%1` or `${1}` in the script examples below takes the pdf file names, the argument `%3` or `${3}` takes the named destinations within a pdf file.

Examples for Windows Users

PDF-XChange Viewer:

```
@echo off
set reader="C:\Program Files\Tracker Software\PDF Viewer\PDFXCview.exe"
start "Launch PDF" %reader% /A nameddest=%3 %1
```

SumatraPDF:

```
@echo off
start "Launch PDF" "C:\T32\bin\SumatraPDF.exe" ^
                               %1 ^
                               -nameddest %3 ^
                               -reuse-instance
```

The caret sign ^ serves as a line continuation character in Windows batch files (*.bat). White space characters after ^ are NOT permissible.

Foxit Reader:

```
@echo off
set reader="C:\Program Files (x86)\Foxit Software\Foxit Reader\Foxit
Reader.exe"
start "Launch PDF" %reader% /A "nameddest=%3" %1
```

Adobe Acrobat X Pro:

```
@echo off
start acrobat.exe  /n /A "nameddest=%3" %1
```

Example for Linux Users

evince (as of version 3.x; earlier versions do not support the `-n` option):

```
#!/bin/bash
/usr/bin/evince ${1} -n ${3} &
```

evince or **xpdf** or **Firefox**: This Linux shell script displays the pdf of the TRACE32 help system in the first available pdf viewer:

```
#!/bin/bash
evince "${1}" -n ${3}                      || \
xpdf -remote t32xpdf -raise "${1}" +${3}  || \
firefox file:///${1}#nameddest=${3}       &
```

The backslash \ serves as a line continuation character in Linux shell scripts (*.sh).

Previous Releases - HELP Installation Problems

[up to Software Release 09/2015]

Some common installation problems are described here.

Loads only old Online Help

Verify if help.t32 is in TRACE32 system path (by default c:\t32), and if you have rights to read this file.

Alternate Call for Adobe Reader

[only Software Releases 09/2014, 02/2015, and 09/2015]

By default, the trace32.api file relays the call for a particular help topic from TRACE32 to Adobe Reader. However, if you encounter problems after updating your Adobe Reader version, you can bypass the trace32.api file with the code shown below. For a step-by-step procedure, see **“Previous Releases - Configuring an Alternate PDF Viewer”**, page 100.

Adobe Reader:

```
@echo off
start acrord32.exe /n /A "nameddest=%3" "%1"
```

Acrobat does not start automatic

Reinstall Acrobat Reader, verify if everybody can write to Acrobat subfolder “plug_ins” – if not, copy “trace32.api” manually to this folder

Acrobat opens File, but does not jump to the right Chapter

Verify if there is a Acrobat menu entry “Help->About plug-ins->About Trace 32” – if not copy “trace32.api” to Acrobat subfolder “plug_ins”

Warning “Communication with Acrobat Reader failed” always when using the Help

Copy “trace32.api” to Acrobat subfolder “plug_ins”

Warning “Communication with Acrobat Reader failed” only at first Acrobat Startup

Acrobat starts too slow.

Good trick to improve Acrobat startup time is to delete never needed plug_ins:

rename folder “plug_ins” to “plug_ins_bak”

then create empty “plug_ins” folder and copy there only “trace32.api” and other really needed plug_ins

Warning “Please install Acrobat Reader to see pdf help files!”

This message is displayed if the Acrobat installation could not be found on windows systems. Download the Acrobat Reader software from www.adobe.com and install it.

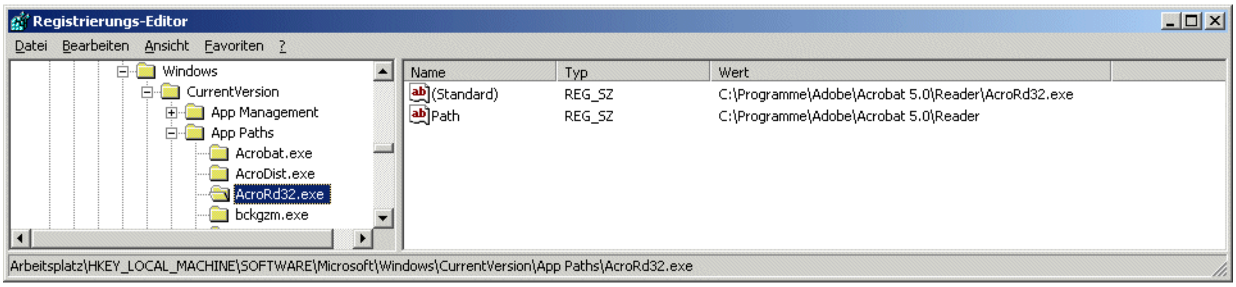
If you installed Acrobat already and this message is displayed anyway, check if one of the following registry entries exist (execute regedit.exe):

- HKEY_LOCAL_MACHINE: SOFTWARE\\ Microsoft\\ Windows\\ CurrentVersion\\ App Paths\\ Acrobat.exe
- HKEY_LOCAL_MACHINE: SOFTWARE\\ Microsoft\\ Windows\\ CurrentVersion\\ App Paths\\ AcroRd32.exe

If none of these keys exist, remove your current installation and install it again. You can also start Acrobat manually before using the online help and ignore the error message.

If you have the rights and if you are skilled to change registry entries, you can add it manually. But you have to be sure what you are doing - changing registry entries can affect the whole behavior of the Windows system!

Add the key “AcroRd32.exe” as shown below, change the Acrobat installation where it is installed on your system.



```
Name         Typ      Wert
(Standard)   REG_SZ   C:\Programme\Adobe\Acrobat 5.0\Reader\AcroRd32.exe
Path         REG_SZ   C:\Programme\Adobe\Acrobat 5.0\Reader
```

Arbeitsplatz\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths\AcroRd32.exe

Warning “Error occurred while trying to start Acrobat Reader!”

Check the registry entries as described above - check if the (Standard) entry is really the correct installation path

Warning “Acrobat Reader could not be started” (Unix only)

Check if environment variable “ACROBAT_PATH” is set correctly to the Acrobat installation path.

InterCom

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

...
```

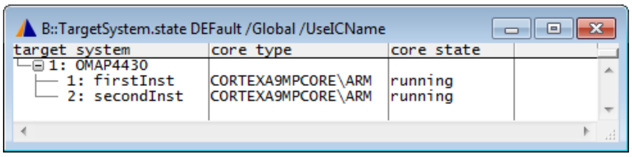
NOTE:	If multiple TRACE32 systems are used on one host, the port numbers must differ!

A good way to familiarize yourself with the **InterCom** command group is to start with the example below.

InterCom.Evaluate	Evaluates InterCom
InterCom.execute	Remote execute command line
InterCom.PING	Test the InterCom system
InterCom.PipeCLOSE	Close named pipe
InterCom.PipeOPEN	Open named pipe
InterCom.PipeREAD	Read from named pipe
InterCom.PipeWRITE	Write to named pipe
TargetSystem.state	Show overview of multicore system
TargetSystem.NewInstance	Start new TRACE32 PowerView instance

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
```



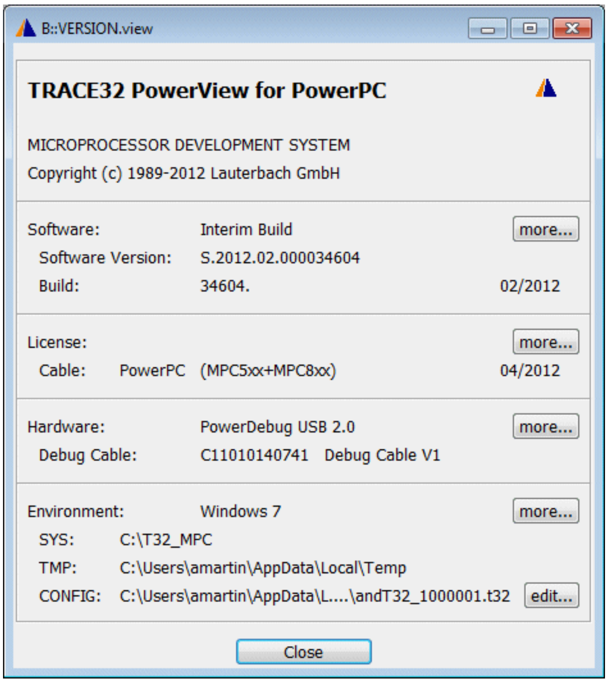
```
target system         core type            core state
└⊟1: OMAP4430
   ├── 1: firstInst   CORTEXA9MPCORE\ARM   running
   └── 2: secondInst  CORTEXA9MPCORE\ARM   running
```

Version Management and Licensing

The **VERSION.view** window provides information about the TRACE32 software version and licenses as well as TRACE32 hardware and environment information.

1. To open the **VERSION.view** window, choose **Help** menu > **About TRACE32**.

2. For details, click the **more** buttons.



TRACE32 PowerView for PowerPC

MICROPROCESSOR DEVELOPMENT SYSTEM
Copyright (c) 1989-2012 Lauterbach GmbH

```
Software:               Interim Build                              more...
  Software Version:     S.2012.02.000034604
  Build:                34604.                                     02/2012

License:                                                           more...
  Cable:      PowerPC   (MPC5xx+MPC8xx)                            04/2012

Hardware:               PowerDebug USB 2.0                         more...
  Debug Cable:          C11010140741   Debug Cable V1

Environment:            Windows 7                                  more...
  SYS:        C:\T32_MPC
  TMP:        C:\Users\amartin\AppData\Local\Temp
  CONFIG:     C:\Users\amartin\AppData\L....\andT32_1000001.t32    edit...
```

For more information on finding serial numbers, see **"Serial Numbers"** in Software Updates, page 12 (updates.pdf).

The following commands are described in the **"PowerView Command Reference"** (ide_ref.pdf).

VERSION.HARDWARE	Displays the version of the used debug hardware
VERSION.SOFTWARE	Displays detailed information about the used TRACE32 software
VERSION.view	Displays window with version info
LICENSE.state	Displays the currently used maintenance contract
LICENSE.List	Displays all license information
LICENSE.UPDATE	Updates the maintenance contract inside a debug cable

Text Editors

This chapter describes how TRACE32 PowerView supports editing text files.

Built-in Editors

TRACE32 PowerView includes two built-in editor types.

1. OS-Native Editor: This editor has a limited feature set as provided by the GUI framework.
2. PowerView Editor: Advanced editor with syntax highlighting and context specific features.

The editor type can be selected using **SETUP.EDITOR.TYPE**.

OS-Native Editor

The features of this editor are limited to the features provided by the GUI framework / OS API that provides the edit control. There are no configurable options for highlighting, indentation etc. The are keyboard shortcuts available as provided by the GUI framework (Windows API, Qt, MOTIF).

PowerView Editor

This editor has a variety of features that are available on all supported host operating systems:

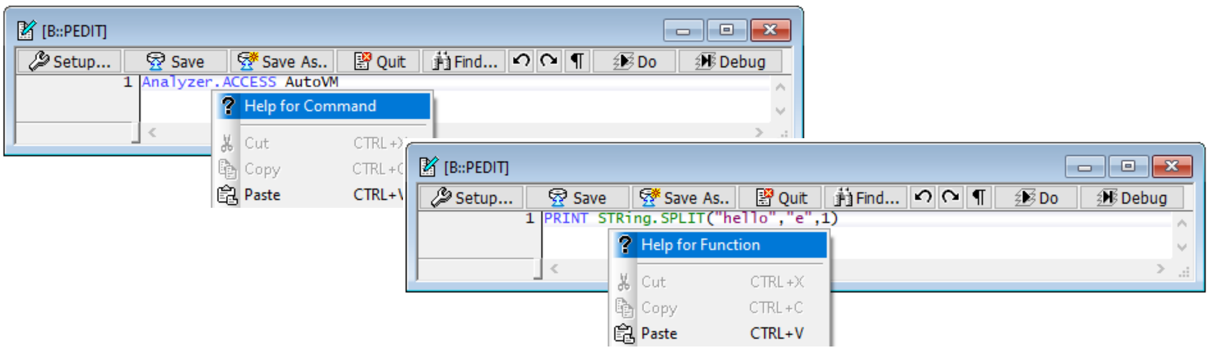
- Syntax Highlighting for all TRACE32 programming languages
- Nesting-aware editing
- Highlighting of matching block, braces, current cursor line, keywords and matching selection
- Configurable visible whitespace and ASCII view
- Automatic indentation during input
- Individual settings for tab size, indentation size and type for several file types (PRACTICE, C, Python, ASM Text and TRACE32 trigger languages.
- Context sensitive context menus (e.g. Help for command / function, Goto label)
- Input suggestions and automatic input completion, see also **SETUP.EDITOR.AutoSuggest**
- Automatic script formatting and command expansion, see **EDIT.FORMAT**.

For a detailed description of available configuration options see the **SETUP.EDITOR** command group.

Context Sensitive Context Menu

Below is an overview of the context sensitive context menu features The context menu is opened by right-click on the item of interest.

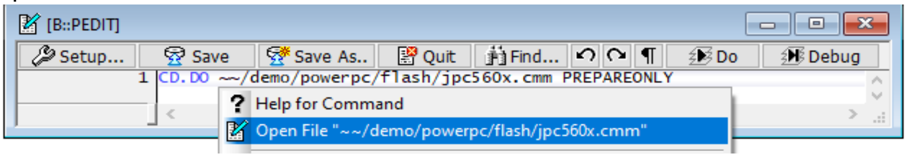
Context menu of command or PRACTICE function leads to link info TRACE32 Help:



```
Analyzer.ACCESS AutoVM
```

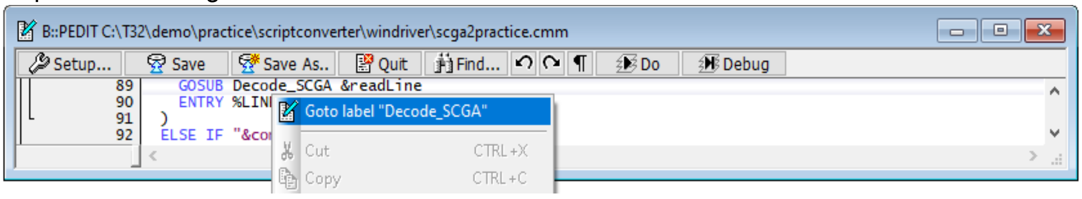
```
PRINT STRing.SPLIT("hello","e",1)
```

Open destination text file addresses in a command in new edit window:


[B::PEDIT]
Setup... Save Save As.. Quit Find... ¶ Do Debug
Help for Command
Open File "~~/demo/powerpc/flash/jpc560x.cmm"

```
CD.DO ~~/demo/powerpc/flash/jpc560x.cmm PREPAREONLY
```

Jump to labels of e.g. subroutines:


B::PEDIT C:\T32\demo\practice\scriptconverter\windriver\scga2practice.cmm
Setup... Save Save As.. Quit Find... ¶ Do Debug
Goto label "Decode_SCGA"
Cut CTRL+X
Copy CTRL+C

```
    GOSUB Decode_SCGA &readLine
    ENTRY %LIN
  )
  ELSE IF "&co
```

Keyboard Shortcuts

Below is a list of keyboard shortcuts supported by the PowerView editor:

CTRL+SPACE	Open Auto Completion box. See also **SETUP.EDITOR.AutoSuggest**.
CTRL+A	Select all text
CTRL+C	Copy selected text to clipboard
CTRL+D	Comment line or selection (D = Disable)
CTRL+E	Uncomment line or selection (E = Enable)
CTRL+F	Open *Find* dialog
CTRL+G	Open *Goto Line* dialog
CTRL+H	Open *Replace* dialog
CTRL+S	Save document

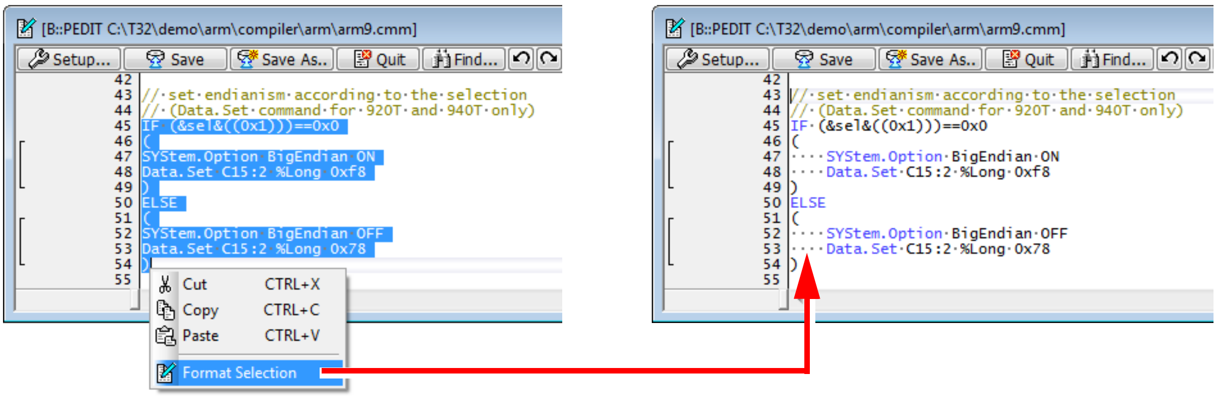
CTRL+V	Paste from clipboard
CTRL+X	Cut (copy to clipboard and delete)
CTRL+Y	Redo
CTRL+Z	Undo
CTRL+ <cursor key>	Navigate cursor word-wise
SHIFT+ <cursor key>	Select text
CTRL+SHIFT +<cursor key>	Select text word-wise
CTRL+ BACKSPACE	Delete text word-wise

Automatic Formatting

Automatic formatting is available for PRACTICE (**PEDIT**), Menu (**MENU.Program**) and Peripheral View (**PER.Program**) and re-indents the selected file or block.

Automatic formatting is available from context menu and also from command line with additional features like bringing all commands into the correct camel cased form. See **EDIT.FORMAT** for more information.

Unformatted block: Formatted block:



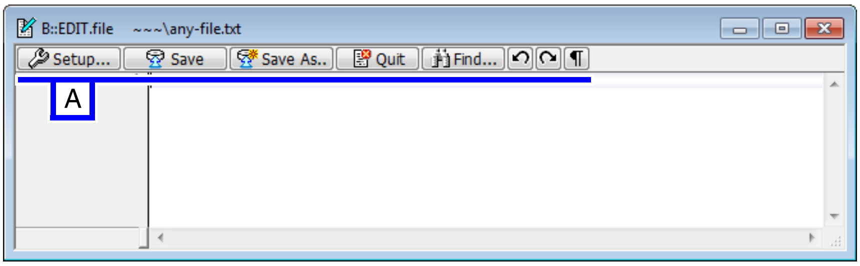
```
//·set·endianism·according·to·the·selection
//·(Data.Set·command·for·920T·and·940T·only)
IF (&sel&((0x1)))==0x0
(
SYStem.Option BigEndian ON
Data.Set C15:2 %Long 0xf8
)
ELSE
(
SYStem.Option BigEndian OFF
Data.Set C15:2 %Long 0x78
)
```

```
//·set·endianism·according·to·the·selection
//·(Data.Set·command·for·920T·and·940T·only)
IF·(&sel&((0x1)))==0x0
(
····SYStem.Option·BigEndian·ON
····Data.Set·C15:2·%Long·0xf8
)
ELSE
(
····SYStem.Option·BigEndian·OFF
····Data.Set·C15:2·%Long·0x78
)
```

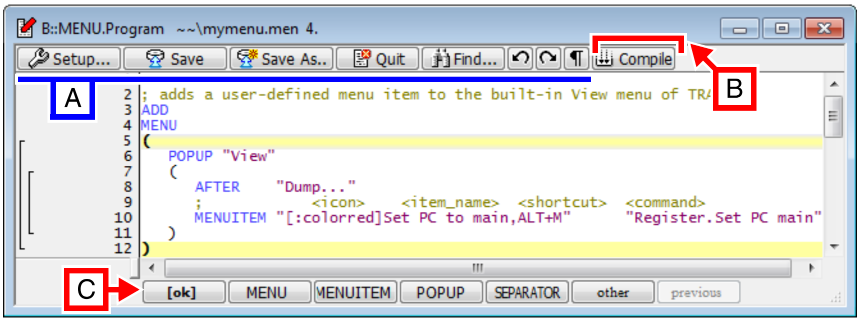
Special Purpose Editor Windows

TRACE32 includes editor windows that are specific for a certain programming language:

The special purpose editor windows allow easy access to commands that are specific to the respective editor and file type. For examples of editor-specific buttons, see [**B**] and [**C**].

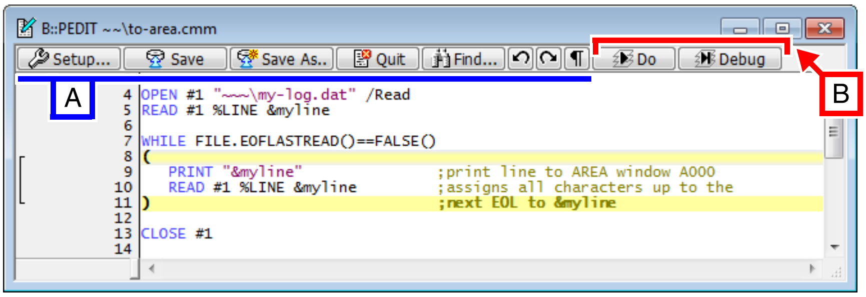


The general-purpose editor



```
; adds a user-defined menu item to the built-in View menu of TR/
ADD
MENU
(
   POPUP "View"
   (
      AFTER    "Dump..."
      ;            <icon>     <item_name>   <shortcut>   <command>
      MENUITEM "[:colorred]Set PC to main,ALT+M"       "Register.Set PC main"
   )
)
```

Editor for menu programming (special-purpose editor)



```
OPEN #1 "~~~\my-log.dat" /Read
READ #1 %LINE &myline

WHILE FILE.EOFLASTREAD()==FALSE()
(
   PRINT "&myline"              ;print line to AREA window A000
   READ #1 %LINE &myline        ;assigns all characters up to the
)                               ;next EOL to &myline

CLOSE #1
```

Editor for PRACTICE scripts (special-purpose editor)

In addition to, or as an alternative to the TRACE32 editors, you can configure an external editor for use in TRACE32 using the **SETUP.EDITEXT** command. For more information about the use of external editors in TRACE32, see **“External Editors”**, page 114.

General-purpose editor:

EDIT	Primarily used to create and edit text files, e.g. *.txt, *.log, *.dat, etc.

Special-purpose editors:

PEDIT [*<file>*]	Editor for PRACTICE scripts (*.cmm).
PER.Program [*<file>*]	Editor for programming the peripheral description files (*.per).
MENU.Program [*<file>*]	Editor for creating your own menus or customizing the TRACE32 menus (*.men).
DIALOG.Program [*<file>*]	Editor for creating your own dialogs (*.dlg).
BITMAPEDIT [*<file>*]	BItmap editor for drawing user-defined icons. They can be embedded in these TRACE32 file types: *.cmm, *.men, or *.dlg.
Data.PROGRAM [*<address>*] [*<file>*]	Editor for writing an assembler program (*.asm).

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

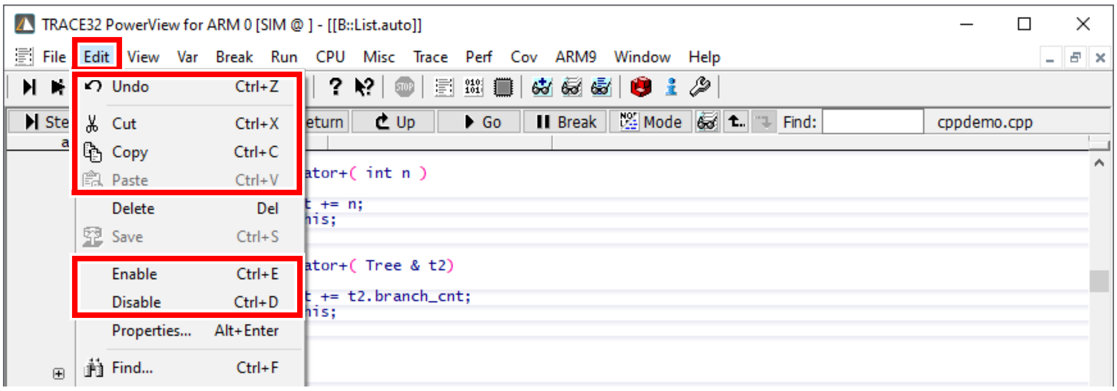
Break.Program [*<file>*]	Editor for on-chip breakpoint programs (*.ctl).
Integrator.Program [*<file>*]	Editor for Integrator trigger programs (*.tap).
Probe.Program [*<file>*]	Editor for PowerProbe trigger programs (*.tap).

Examples

```
EDIT config.t32                 ; edit one file

EDIT *.c                        ; the asterisk opens a file selection
                                ; dialog, where you can browse for the
                                ; file you want to edit

EDIT test.txt                   ; open file and edit
EDIT test.txt                   ; access the same file through a second
                                ; EDIT window
```

Edit Menu

Below is an overview of the Edit menu tools displayed in the menu bar.



- **Undo**: same action as pressing key Ctrl+z or command **EDIT.UNDO**.
- **Cut**: same action as pressing key Ctrl+x.

  Removes current selected content from text and transfer into clipboard.
- **Copy**: same action as pressing key Ctrl+c.

  Transfers current selected text into clipboard.
- **Paste**: same action as pressing key Ctrl+v.

  Inserts content of clipboard at current cursor position.
- **Delete**: same action as pressing key Del.

  Removes character right of current cursor position or delete selected text.
- **Enable**: changes current line to a comment line (prefix with //).
- **Disable**: removes comment from current line (remove prefix //).

External Editors

In addition to, or as an alternative to the built-in TRACE32 editors, you can configure an external editor for use in TRACE32. This allows you to take advantage of the combined features of both (a) the respective built-in TRACE32 editor you have selected and (b) the external editor - while you are working on the same file at the same time in both editors.

Syntax highlighting files for external editors reside under ~~/demo/practice/syntaxhighlighting and are available for the following external editors:

- TextPad
- UltraEdit
- Kate
- Notepad++

The syntax highlighting files for external editors cover the following TRACE32 file types:

- PRACTICE script files (*.cmm)
- Menu files (*.men)
- Dialog files (*.dlg)
- OCTL files (*.octl)

Configuring an External Editor

Before configuring an external editor for use in TRACE32, you should consult the online help of your favorite external editor for information about (a) syntax highlighting files or syntax definition files and (b) command line parameters for file name and line number.

Background information is also provided in the headers of the syntax files and in the readme.txt residing under ~~/demo/practice/syntaxhighlighting

To configure an external editor for TRACE32:

1. In your TRACE32 system directory, navigate to ~~/demo/practice/syntaxhighlighting/*<editor>*.
2. Copy the required syntax highlighting file, and paste it into the folder where your external editor expects syntax highlighting files.

   The remaining steps for registering a syntax highlighting file depend on the external editor.

   In TextPad, for example, you need to create a new document class and assign the TRACE32 file types *.cmm, *.men, and *.dlg to this new document class.
3. Look up the examples and description of the following TRACE32 command. They tell you how to replace the TRACE32 editor call with an external editor call.

SETUP.EDITEXT	Define an external editor

Working with TRACE32 and the External Editor

The interaction between a TRACE32 editor and an external editor allows you to take advantage of the combined features of both editors. For example, after saving your PRACTICE script in the external editor, you can immediately execute and/or debug your script in TRACE32.

Prerequisite:

- You have configured an external editor for use in TRACE32. If not, then the **EDIT** command will continue to open the TRACE32 **EDIT** window instead of your external editor.

To work with TRACE32 and the external editor in parallel:

1. Run the **PSTEP** and **EDIT** commands for the same PRACTICE script file. For example:

```
PSTEP c:\temp\demo.cmm      ;open file in the PSTEP window
EDIT  c:\temp\demo.cmm      ;open same file in the external editor
```

2. In the external editor, type your PRACTICE script. Let’s use this very simple demo script:

```
AREA.view                   ;display the default AREA window
AREA.RESet                  ;reset the AREA window

RePeaT 10.                  ;repeat the PRINT command
   PRINT "Hello TRACE32!"

ENDDO
```

3. In TRACE32, right-click the window caption of the **PSTEP** window, and then press **Enter**.
   - The saved PRACTICE script file is loaded into the **PSTEP** window.
   - You are now ready to step through your script line by line (**STEP** button), execute it (**Continue** button), set PRACTICE breakpoints (see **PBREAK**).

NOTE: If your script starts with a **WinCLEAR** command, you can prevent the **PSTEP** window from being erased by opening it with the **WinResist** pre-command.

In our example: `WinResist.PSTEP c:\temp\demo.cmm`

Icons

TRACE32 allows you to customize the user interface and add icons to your customized user interface. This chapter informs you about the supported icon types, tells you where you can add icons, and describes step-by-step how you can create your own icons.

TRACE32 supports two icon types:

- Built-in icons
- User-defined icons

Both icon types can be added to the following dialog, menu, and toolbar elements:

BUTTON	Define a button
DEFBUTTON	Define the default button
DYNAMIC	Define a dynamic, single-line area
ICON	Define an icon for the top left corner of a dialog
MENUITEM	Define an item in a menu, popup menu or a local button
STATIC	Define a non-dynamic area in a dialog
TOOLITEM	Define a button in the toolbar

Icon-capable elements can be used in the following TRACE32 file types:

- PRACTICE script files (*.cmm)
- Menu files (*.men)
- Dialog files (*.dlg)

Which Icon Type Do You Need for Your Project?

You can choose built-in icons from the icon library. For more information, see **“Built-in Icons and Icon Library”**, page 118.

You can create your own, user-defined icons with the TRACE32 bitmap editor. For step-by-step instructions, see **“Inserting a Placeholder for User-Defined Icons”**, page 119.

BITMAPEDIT *<file>*	Open the bitmap editor

Built-in Icons and Icon Library

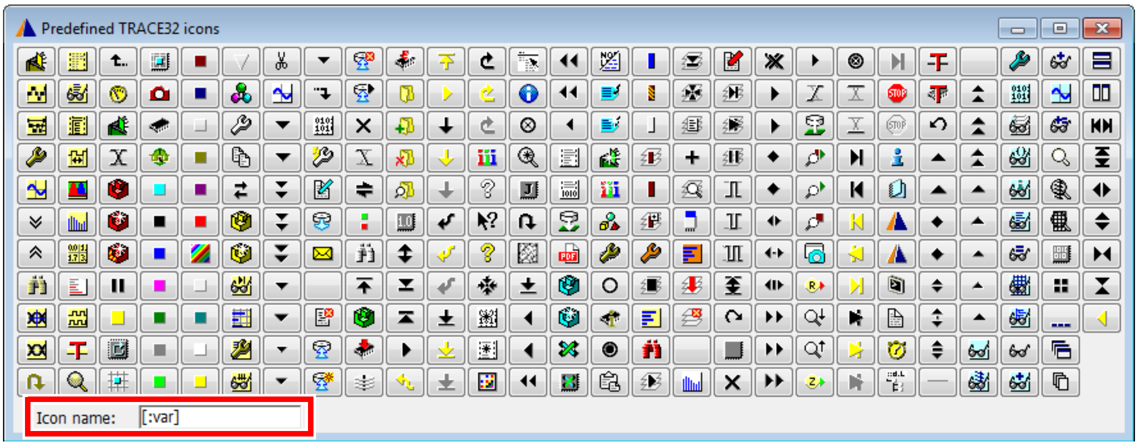
TRACE32 provides a number of built-in icons. You can easily include these built-in icons in icon-capable dialog, menu, and toolbar elements. Using the TRACE32 icon library, you can:

- Get an overview of all available icons
- Get the built-in name of each icon

To display the TRACE32 icon library:

1. Choose **Misc** menu > **Tools** > **Display internal icon library**.

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



Icon name: [:var]

2. Click any icon.
   The built-in icon name is displayed in the **Icon name** field.

3. Observe this syntax for built-in icons: **"[:**`<built_in_icon_name>`**]**`<your_text>`**"**

```
DIALOG.view
(; Assigns the icon to BUTTON
 BUTTON "[:var]Any text"  "Var.View"
)
STOP
DIALOG.END
```

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

Inserting a Placeholder for User-Defined Icons

Icon-capable dialog, menu, and toolbar elements require an icon placeholder in the form of two square brackets **[]**.

To insert an icon placeholder for a user-defined icon:

1. In TRACE32, open the file where you want to add an icon, e.g.:

```
PEDIT ~~/demo/practice/dialogs/dialog_edit.cmm ;PRACTICE script file
                                               ;or
MENU.Program ~~/demo/menu/demo.men             ;menu file
                                               ;or
DIALOG.Program <your_file>.dlg                 ;dialog file
```

2. Observe this syntax for the icon placeholder: **"[]**<your_text>**"**

   Examples:

```
;DIALOG element (*.dlg or *.cmm file)
BUTTON   "Cancel"   "GOTO cancel"
BUTTON   "[]Cancel" "GOTO cancel"
```

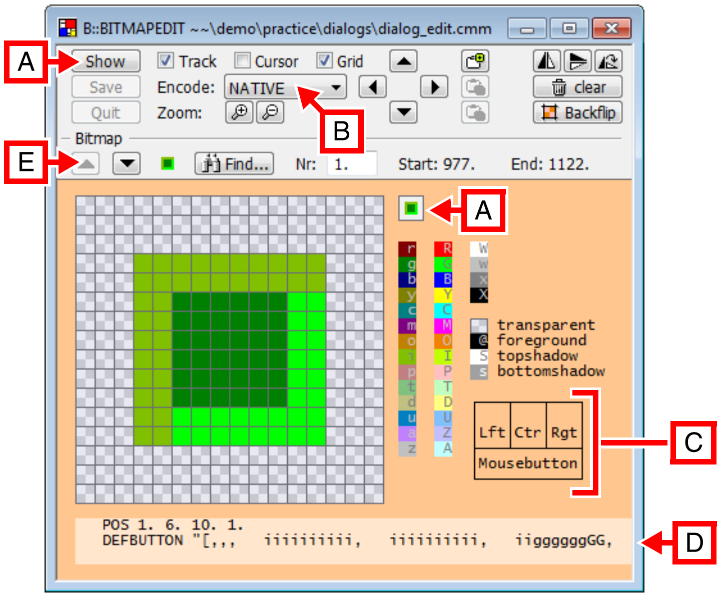
```
;MENU element   (*.men file)
MENUITEM "Edit .c File..."   "EDIT *.c"
MENUITEM "[]Edit .c File..." "EDIT *.c"
```

3. Click **Save**.

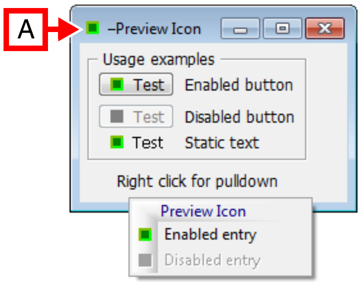
Next: **"Drawing Icons"**, page 120.

Drawing Icons

After inserting the icon placeholders (see previous section), you can open the file in the **BITMAPEDIT** window and draw an icon on the canvas.



```
POS 1. 6. 10. 1.
DEFBUTTON "[,,,    iiiiiiiii,    iiiiiiiii,    iigggggGG,
```



A Opens the **Preview Icon** window.

B Bitmap format (For more information, see the **BITMAPEDIT** command.)

C Assign a color to a mouse button by clicking a color in the palette with that mouse button.

D The source code of an icon is inserted between the corresponding icon placeholder `[]` while you are drawing the icon.

E Up and down arrow buttons let you navigate from one icon or icon placeholder to the next.

To draw an icon:

1. Open your file in two TRACE32 editor windows.

   - The first editor window contains the source code of your project.
   - The second editor window, i.e. the **BITMATEDIT** window, provides the icon drawing tools.

   Example:

```
PEDIT       ~~/demo/practice/dialogs/dialog_edit.cmm   ;1. editor
BITMAPEDIT ~~/demo/practice/dialogs/dialog_edit.cmm   ;2. editor
                                                       ;or
MENU.Program ~~/demo/menu/demo.men                     ;1. editor
BITMAPEDIT   ~~/demo/menu/demo.men                     ;2. editor
                                                       ;or
DIALOG.Program <file>.dlg                              ;1. editor
BITMAPEDIT     <file>.dlg                              ;2. editor
```

The same file should now be open in two TRACE32 editor windows.

2. In the **BITMAPEDIT** window, under **Bitmap**, click the up and down arrow to navigate to the icon placeholder you want [**E**].

3. From the color palette, choose the colors you want, and draw an icon.

4. When done, click **Save**.

5. Click in the window of the first editor: the **PEDIT** window or the **MENU.Program** window or the **DIALOG.Program** window. You are prompted to reload the file.

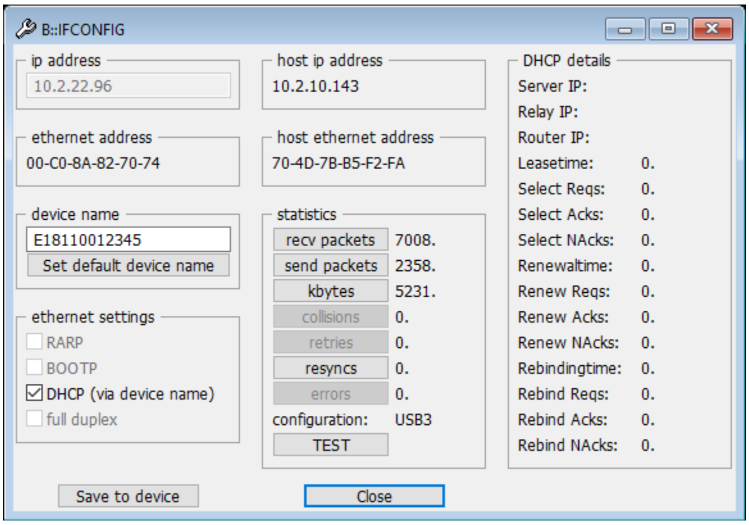
6. Click **Yes** to reload.

You are now ready to execute the file to view the finished icon.

- In the **PEDIT** window, click **DO**.
- In the **MENU.Program** window, click **Compile**.
- In the **DIALOG.Program** window, click **Comp+Show**.

Interface

For more information about the configuration of the interface, see **“TRACE32 Installation Guide”** (installation.pdf). Commands are described in the **“PowerView Command Reference”** (ide_ref.pdf).



B::IFCONFIG

ip address: 10.2.22.96
host ip address: 10.2.10.143
ethernet address: 00-C0-8A-82-70-74
host ethernet address: 70-4D-7B-B5-F2-FA
device name: E18110012345

ethernet settings:
- [ ] RARP
- [ ] BOOTP
- [x] DHCP (via device name)
- [ ] full duplex

statistics	
recv packets	7008.
send packets	2358.
kbytes	5231.
collisions	0.
retries	0.
resyncs	0.
errors	0.
configuration:	USB3

DHCP details	
Server IP:	
Relay IP:	
Router IP:	
Leasetime:	0.
Select Reqs:	0.
Select Acks:	0.
Select NAcks:	0.
Renewaltime:	0.
Renew Reqs:	0.
Renew Acks:	0.
Renew NAcks:	0.
Rebindingtime:	0.
Rebind Reqs:	0.
Rebind Acks:	0.
Rebind NAcks:	0.

IFCONFIG.state	Interface configuration
IFCONFIG.TEST	Test interface function and speed
IFCONFIG.PROfile	Display operation profiles

ALT+ Spacebar

Opens the window manager menu of the active window.

- Available for the TRACE32 window modes FDI and MTI.
- If the TRACE32 window mode is MDI, then the **ALT+spacebar** shortcut works only for windows preceded by the **WinExt** pre-command, e.g. **WinExt.Register.view** or **WinExt.List**

Alt+F4

- Closes the TRACE32 main window if no **WinExt**.*<window>* is selected.
- Closes the active **WinExt**.*<window>*. See also esc key.

Ctrl+A

Select all in the TRACE32 editors.

Ctrl+C

Copy in the TRACE32 editors.

Ctrl+F

- Opens the **Find** dialog window. TRACE32 searches in the active window.
- Find operation in the TRACE32 editors.

Ctrl+G

Go to *<line>* in the TRACE32 editors.

Ctrl+Left

Move to previous word in the TRACE32 editors.

Ctrl+H

Find-and-replace operation in the TRACE32 editors.

Ctrl+F4

Closes the active window (i.e. windows without the **WinExt**. pre-command).
See also esc key.

Ctrl+F6

Selects the next window (i.e. windows without the **WinExt**. pre-command).

Ctrl+Right

Move to next word in the TRACE32 editors.

Ctrl+S

Save a file in the TRACE32 editors.

Ctrl+V

Paste in the TRACE32 editors.

Ctrl+X

Cut in the TRACE32 editors.

Ctrl+Y

Redo in the TRACE32 editors.

Ctrl+Z

Undo in the TRACE32 editors.

Ctrl+D and Ctrl+E

Ctrl+D disables the selected breakpoint.

Ctrl+E enables the selected breakpoint.

esc

Closes the active window - regardless of whether the active window is preceded by the **WinExt**. pre-command or not.

F1

- To get context-sensitive help on a window or dialog, click the window or dialog, and then press **F1**.
- To get context-sensitive help for a command, type the command at the command line, append an empty space, and then press **F1**.

F2	Single step
F3	Step over function call or subroutines
F4	Step Diverge Path: Leave loops, go till something new happens.
F5	Go return / Go to the last instruction
F6	Go up / return to the caller function

F7	Go / Start real-time execution
F8	Break / Stop real-time execution
F9	Toggle between the debug modes HLL and MIX
Up and down arrow keys	Go up / down in the command line history.

Shift+Tab

Navigate back through the softkeys of a command group:

1. At the command line, type for example: `Data.LOAD`.
2. Press **Shift+Tab** to navigate *back* through the softkeys.
   (Alternatively, click previous.)

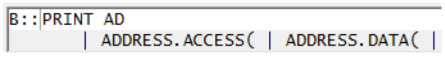
Tab and Tab Completion for Commands

(A) Navigate forward through the softkeys of a command group:

1. At the command line, type for example: `Data`.
2. Press **Tab** repeatedly to navigate *forward* through the softkeys.
   (Alternatively, click other.)

(B) Access the list of PRACTICE functions:

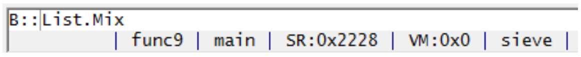
1. At the command line, type: `PRINT` or `Data.Print`
2. Type the first few letters of the PRACTICE function you want, e.g. `ad`
3. Press **Tab** repeatedly to cycle through the list of matching PRACTICE functions.



```
B::PRINT AD
         | ADDRESS.ACCESS( | ADDRESS.DATA( |
```

(C) Cycle through the list of symbols:

1. At the command line, type for example: `List.Mix`
2. Press **Tab** repeatedly to cycle through the list of recently used symbols.



```
B::List.Mix
            | func9 | main | SR:0x2228 | VM:0x0 | sieve |
```

   - Type the first few letters of the symbol you want.
   - Press **Tab** repeatedly to cycle through the list of matching symbols.

(D) Complete a command:

- At the command line, type for example `symb` and then press **Tab**.
  TRACE32 completes `symb` to the command `sYmbol`

(E) Display suggestions for completing a command:

1. At the command line, type for example `tr` and then press **Tab**.
   TRACE32 suggests `TrBus | TrOnchip | Trace | TRANSlation`
2. Press **Tab** repeatedly to cycle through the list of matching commands.

Pause / Break

Moves the insertion point from any TRACE32 window back to the TRACE32 command line.

Appendix - About the TRACE32 Software Version Numbers

The version number consists of the following building blocks:

Format:	*<type>*.*<year>*.*<month>*.*<build_number>*
<type>:	**R** | **P** | **N** | **S** | **F**

<type>	• **R**: release build • **P**: pre-release build • **N**: nightly build • **S**: interim build (“snapshot”) • **F**: feature build
<year>	• Year of the software version. This is the year in which a release or pre-release was branched from the development trunk. • Four-digit representation of a year (Return value example: 2018).
<month>	• Month of the software version. This is the month in which a release or pre-release was branched from the development trunk. • Two-digit representation of a month with leading zeros (Return values: 01 … 12).
<build_number>	Build number.

Examples of Version Numbers:

```
R.2018.09.000103893   ;release build

S.2018.12.000104075   ;interim build
```

Information about the version number is displayed in the following windows:

VERSION.SOFTWARE	Displays software version information.
VERSION.view	Displays software, license, hardware, and environment information.

Information about the version number can be returned with the following functions:

VERSION.SOFTWARE()	Returns the entire version number string.
VERSION.BUILD()	Returns the upper build number from the release branch.
VERSION.BUILD.BASE()	Returns the lower build number from the release branch.
VERSION.DATE()	Returns the software version date, e.g. 2018/09