## PowerView User’s Guide

**TRACE32 Online Help**

**TRACE32 Directory**

**TRACE32 Index**

**TRACE32 Documents**

**PowerView User Interface**

**PowerView User’s Guide**

**History**

**Structure and Contents of the Documentation**

- Online Documentation .......................... 5
- In-Circuit Debugger TRACE32-ICD ..... 7
- In-Circuit Emulator TRACE32-ICE .... 9
- Fully Integrated RISC Emulator TRACE32-FIRE .. 11

**Program Start**

- In-Circuit Debugger TRACE32-ICD .... 14
- In-Circuit Emulator TRACE32-ICE .... 15
- Fully Integrated RISC Emulator TRACE32-FIRE .. 15

**Program End**

**PowerView - Screen Display**

- Concept ........................................ 17
- Graphical User Interface - Window Modes ..... 17
  - MDI User Interface ......................... 18
  - MWI User Interface ....................... 19
- Main Menu Bar .......................... 20
- Accelerators ......................... 20
- Main Toolbar ........................ 21
- Work Area ............................ 21
- Message Line .......................... 22
  - Error Messages .......................... 22
  - General Messages ..................... 22
  - Additional Information on Cursor Position ... 22
- Softkeys .................................. 23
- State Line .................................. 24
- Cursor .................................. 24
- Cursor - In a Hypervisor Environment .... 25
- Debug .................................. 25
- Trace .................................. 26

©1989-2020 Lauterbach GmbH
History

28-Jun-19  Updated the entry of the parameter type “\texttt{symbol}” in section “Parameter Types”.

25-Jan-19  New chapter “Editing Files in the TRACE32 Editors - Syntax Highlighting”.

22-Jan-19  Placed chapter “External Editors - Syntax Highlighting” into this manual; previously the chapter was located in practice_user.pdf.

28-Aug-18 The title of the manual was changed from “IDE User's Guide” to “PowerView User's Guide”.

16-Mar-18 The new camel casing of the INTERCOM commands is now \texttt{InterCom}.

Structure and Contents of the Documentation

This chapter describes the structure of the TRACE32 documentation - printed as well as online.

Since the development of the TRACE32 software is always in progress, the online version of the manuals will always be more up to date than the printed version. The first pages of the printed documentation contain the issue date.

The release history in the online 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

The most recent version of the TRACE32 documentation is available online.

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, you can find a directory \texttt{PDF}. This directory contains the complete TRACE32 documentation in PDF format. Open \texttt{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:

- **“Training Simulator and Demo Software”** (demo.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. At the end of the manual, you can check your learning progress with an interactive mini quiz.

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

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

- **“Debugger Basics - SMP Training”** (training_debugger_smp.pdf) - An introduction to SMP debugging

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

- [https://www.lauterbach.com/reference_card_web.pdf](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](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](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_.pdf)  
  This part provides an alphabetical list of all debugger commands. All commands that are not available for TRACE32-ICD are marked with:
  -  (E) - TRACE32-ICE only
  -  (F) - TRACE32-FIRE only

- **“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.

- **“ICD Extensions”**
  Refer to this part if you want to use the TPU debugger or the PCP debugger.

- **“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.
TRACE32-ICE comes with 4 binders which provide documentation support for all its capabilities.

TRACE32-ICE is divided into two major functional units: the Emulator and the Analyzer.

- The emulator-part of TRACE32-ICE is responsible for carrying out debugging functions, for managing breakpoints, for displaying memory and high-level-language structures and for code coverage …
- The analyzer-part of TRACE32-ICE is responsible for displaying and managing the trace buffer, for programming the analyzer trigger unit to perform selective tracing and to carry out complex trigger sequences, for statistic functions and performance analyses …

The following list will give you an idea of the contents of each manual.

<table>
<thead>
<tr>
<th>Manual Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>“ICE User’s Guide” (ice_user.pdf)</td>
<td>This parts makes you familiar with the concept and the main features of the emulator-part of TRACE32-ICE.</td>
</tr>
<tr>
<td>“ICE Targets”</td>
<td>This part provides all CPU specific information for your TRACE32-ICE, chiefly how to set up TRACE32-ICE for your target. Here you will also find all extra features that are supported for your CPU.</td>
</tr>
<tr>
<td>“General Reference Guide”</td>
<td>This part provides an alphabetical list of all commands for the emulator-part of TRACE32-ICE. All commands that are not available for TRACE32-ICE are marked with (B) - TRACE32-ICD only or (F) - TRACE32-FIRE only.</td>
</tr>
<tr>
<td>“ICE/FIRE Analyzer User’s Guide” (analyzer_user.pdf)</td>
<td>This part makes you familiar with the concept and the main features of the analyzer-part of TRACE32-ICE.</td>
</tr>
<tr>
<td>“ICE/FIRE Analyzer Trigger Unit Programming Guide” (analyzer_prog.pdf)</td>
<td>This part makes you familiar with the concept of the trigger programming language. It contains examples for trigger programs and alphabetical lists of all actions and input events.</td>
</tr>
<tr>
<td>“PowerView User’s Guide” (ide_user.pdf)</td>
<td>All TRACE32 development tools share the common user interface TRACE32-PowerView. This part describes the basic user interface (command structure, online help, editing and managing files, printer operations etc.).</td>
</tr>
<tr>
<td>“PowerView Command Reference” (ide_ref.pdf)</td>
<td>This part provides an alphabetical list of all TRACE32 PowerView commands.</td>
</tr>
<tr>
<td>“PRACTICE Script Language User’s Guide” (practice_user.pdf)</td>
<td>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.</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>“PRACTICE Script Language Reference Guide”</strong></td>
<td>This part provides an alphabetical list of all PRACTICE commands.</td>
</tr>
<tr>
<td>(practice_ref.pdf)</td>
<td></td>
</tr>
<tr>
<td><strong>“ICD Extensions”</strong></td>
<td>Refer to this part, if you want to use the TPU debugger.</td>
</tr>
<tr>
<td><strong>“OS Awareness Manuals”</strong> (rtos_&lt;os&gt;.pdf)</td>
<td>Refer to this part if you want to use the TRACE32 OS Awarenesses (= RTOS Debuggers in previous TRACE32 releases).</td>
</tr>
<tr>
<td><strong>“Tools Integration”</strong></td>
<td>Refer to this part, if you want to run TRACE32-ICE from a 3rd-party user interface.</td>
</tr>
<tr>
<td><strong>“TRACE32 Installation Guide”</strong> (installation.pdf)</td>
<td>This part is the general Installation Guide for all TRACE32 development tools.</td>
</tr>
<tr>
<td><strong>“ICE Port Analyzer User’s Guide”</strong></td>
<td>Your TRACE32-ICE can be enhanced by a port analyzer to trace additional CPU lines. A description of the Port Analyzer and its features is provided in the “ICE Port Analyzer User’s Guide” (port_user.pdf)</td>
</tr>
<tr>
<td>(port_user.pdf)</td>
<td></td>
</tr>
<tr>
<td><strong>“Timing Analyzer”</strong></td>
<td>The Timing Analyzer provides timing and state analysis, a pattern generator and a serial line tester. A description of the Timing Analyzer and its features is provided in the “Timing Analyzer User’s Guide” (time_user.pdf), the “Timing Analyzer Reference Guide” (time_ref.pdf) and the “Timing Analyzer Trigger Unit Programming Guide” (time_prog.pdf)</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Fully Integrated RISC Emulator TRACE32-FIRE

TRACE32-FIRE comes with 4 binders which provide documentation support for all its capabilities.

The following list will give you an idea of the contents of each manual.

<table>
<thead>
<tr>
<th>Manual Title</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>“FIRE User’s Guide” (fire_user.pdf)</td>
<td>This parts makes you familiar with the concept and the main features of TRACE32-FIRE.</td>
</tr>
<tr>
<td>“FIRE Targets”</td>
<td>This part provides all CPU specific information for your TRACE32-FIRE, chiefly how to set up TRACE32-FIRE for your target. Here you will also find all extra features that are supported for your CPU.</td>
</tr>
<tr>
<td>“General Reference Guide”</td>
<td>This part provides an alphabetical list of all commands for debugging and trace with TRACE32-FIRE. All commands that are not available for TRACE32-FIRE are marked with (B) - TRACE32-ICD only or (E) -TRACE32-ICE only.</td>
</tr>
<tr>
<td>“ICE/FIRE Analyzer Trigger Unit Programming Guide” (analyzer_prog.pdf)</td>
<td>This part makes you familiar with the concept of the trigger programming language. It contains examples for trigger programs and alphabetical lists of all actions and input events.</td>
</tr>
<tr>
<td>“PowerView User’s Guide” (ide_user.pdf)</td>
<td>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.)</td>
</tr>
<tr>
<td>“PowerView Command Reference” (ide_ref.pdf)</td>
<td>This part provides an alphabetical list of all TRACE32 PowerView commands.</td>
</tr>
<tr>
<td>“PRACTICE Script Language User’s Guide” (practice_user.pdf)</td>
<td>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.</td>
</tr>
<tr>
<td>“PRACTICE Script Language Reference Guide” (practice_ref.pdf)</td>
<td>This part provides an alphabetical list of all PRACTICE commands.</td>
</tr>
<tr>
<td>“ICD Externsions”</td>
<td>Refer to this part, if you want to use the TPU debugger and PCP debugger.</td>
</tr>
<tr>
<td>“OS Awareness Manuals”</td>
<td>Refer to this part if you want to use the TRACE32 OS Awarenesses (= RTOS Debuggers in previous TRACE32 releases).</td>
</tr>
<tr>
<td>'Tools Integration’</td>
<td>Refer to this part, if you want to run TRACE32-FIRE from a 3rd-party user interface.</td>
</tr>
<tr>
<td><strong>“TRACE32 Installation Guide” (installation.pdf)</strong></td>
<td>This part is the general Installation Guide for all TRACE32 development tools.</td>
</tr>
<tr>
<td><strong>“FIRE Port Analyzer User’s Guide” (fireport_user.pdf)</strong></td>
<td>Your TRACE32-FIRE can be enhanced by a port analyzer to trace additional CPU lines. A description of the Port Analyzer and its features is provided in the <strong>“FIRE Port Analyzer User’s Guide” (fireport_user.pdf)</strong>.</td>
</tr>
</tbody>
</table>
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 “**Command Line Arguments for Starting TRACE32**” (debugger_user.pdf).

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

- `--t32-help`
- `--t32-safestart`
- `--t32-logautostart`
| t32m<cpu>.exe | Windows version for TRACE32-ICD.  
TRACE32-ICD system software is running on PC. |
| t32win.exe (former t32w95.exe) | Windows version for TRACE32-ICD with PODBUS Ethernet Controller (not for PowerTrace/PowerDebugEthernet/PowerDebugUsb/PowerNexus).  
TRACE32-ICD system software is downloaded to the PODBUS Ethernet Controller and runs there. |
| t32m<cpu> | Workstation version for TRACE32-ICD.  
TRACE32-ICD system software is running on workstation. |
| t32cde | Workstation version for TRACE32-ICD with PODBUS Ethernet Controller (not for PowerTrace/PowerDebugEthernet/PowerDebugUsb/PowerNexus).  
TRACE32-ICD system software is downloaded to the PODBUS Ethernet Controller and runs there. |
## In-Circuit Emulator TRACE32-ICE

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>t32win.exe (former t32w95.exe)</td>
<td>Windows version for TRACE32-ICE. TRACE32-ICE system software is downloaded to the System Controller Unit and runs there.</td>
</tr>
<tr>
<td>t32cde</td>
<td>Workstation version of TRACE32-ICE. TRACE32-ICE system software is downloaded to the System Controller Unit runs and there.</td>
</tr>
</tbody>
</table>

## Fully Integrated RISC Emulator TRACE32-FIRE

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>t32m&lt;cpu&gt;.exe</td>
<td>Windows version for TRACE32-FIRE. TRACE32-FIRE system software is running on PC.</td>
</tr>
<tr>
<td>t32win.exe (former t32w95.exe)</td>
<td>Windows version for TRACE32-FIRE with PODBUS Ethernet Controller. TRACE32-FIRE system software is downloaded to the PODBUS Ethernet Controller and runs there.</td>
</tr>
<tr>
<td>t32m&lt;cpu&gt;</td>
<td>Workstation version for TRACE32-ICD. TRACE32-ICD system software is running on workstation.</td>
</tr>
<tr>
<td>t32cde</td>
<td>Workstation version of TRACE32-FIRE with PODBUS Ethernet Controller. TRACE32-FIRE system software is downloaded to the PODBUS Ethernet Controller and runs there.</td>
</tr>
</tbody>
</table>
Getting back to the operating system command level is possible by using the command **QUIT** or by choosing **File menu > Exit**.

<table>
<thead>
<tr>
<th>ABORT</th>
<th>Abort driver program</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUIT</td>
<td>Return to operating system</td>
</tr>
</tbody>
</table>

`: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 the **Targets Guides** for your CPU.
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:

<table>
<thead>
<tr>
<th></th>
<th>Windows</th>
<th>Linux/ Motif</th>
<th>Linux/ Qt</th>
<th>SunOS/ Solaris</th>
<th>HP-UX</th>
<th>OS X/ Motif</th>
<th>OS X/ Qt</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MDI</strong></td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><strong>MWI</strong></td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>
After starting TRACE32, the main window of the TRACE32 PowerView GUI is displayed.

For more information, click the blue GUI terms.
After starting TRACE32, the main window of the TRACE32 PowerView GUI is displayed.

For more information, click the blue GUI terms.
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.

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.

Example:

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

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.

![Main Toolbar](image)

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MENU.AddTool</code></td>
<td>Add a temporary button to the main toolbar, i.e. the button is available only for the current TRACE32 session</td>
</tr>
<tr>
<td><code>TOOLBAR</code></td>
<td>Toggle main toolbar</td>
</tr>
<tr>
<td><code>MENU.Program</code></td>
<td>Editor to write a program that customizes the TRACE32 menu</td>
</tr>
<tr>
<td><code>MENU.ReProgram</code></td>
<td>Menu programming</td>
</tr>
<tr>
<td><code>MENU.RESet</code></td>
<td>Restore default menu and configuration of main toolbar</td>
</tr>
</tbody>
</table>

**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

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

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.

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.

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.
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

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

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

- **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**

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.)
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.

![Trace](image)

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

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.
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.
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**

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].
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**

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

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).

---

**Show/Hide State Line**

<table>
<thead>
<tr>
<th>STATUSBAR ON</th>
<th>Show state line.</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATUSBAR OFF</td>
<td>Hide state line.</td>
</tr>
</tbody>
</table>
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.

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:

```plaintext
WinResist.WinPAGE.List ;keep the table of contents visible on all pages
WinPAGE.Create ;new page with auto-incremented page name
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
```

Right-click the toolbar to create a new page or switch to another page.
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WinPAGE.select</td>
<td>Select page</td>
</tr>
<tr>
<td>WinPAGE.Create</td>
<td>Create page with a user-defined name</td>
</tr>
<tr>
<td>WinPAGE.Delete</td>
<td>Delete page</td>
</tr>
<tr>
<td>WinPAGE.List</td>
<td>List pages</td>
</tr>
<tr>
<td>WinPAGE.REName</td>
<td>Rename page</td>
</tr>
<tr>
<td>WinPAGE.RESet</td>
<td>Reset window system</td>
</tr>
</tbody>
</table>

## Colors

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SETUP.COLOR</td>
<td>Change colors</td>
</tr>
<tr>
<td>sYmbol.List.ColorDef</td>
<td>List keyword colors</td>
</tr>
<tr>
<td>sYmbol.ColorDef</td>
<td>Modify keyword colors</td>
</tr>
<tr>
<td>CmdPOS</td>
<td>Toolbar and/or background color for multicore debugging</td>
</tr>
<tr>
<td>FramePOS</td>
<td>Toolbar and/or background color for multicore debugging</td>
</tr>
<tr>
<td>CORE.SHOWACTIVE</td>
<td>Show active cores in an SMP system. Each core has its own color.</td>
</tr>
<tr>
<td>and CORE.List</td>
<td></td>
</tr>
<tr>
<td>SETUP.COLORCORE</td>
<td>Enable coloring for core-specific info in SMP systems</td>
</tr>
<tr>
<td>&lt;trace&gt;.STATistic.COLOR</td>
<td>Assign colors to function for colored graphics</td>
</tr>
<tr>
<td>GROUP.COLOR</td>
<td>Define color for group indicator</td>
</tr>
</tbody>
</table>
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)

Resulting command syntax for use in a PRACTICE script:

<table>
<thead>
<tr>
<th>Solution 1</th>
<th>Solution 2</th>
<th>Solution 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITM.ON</td>
<td>itm.on</td>
<td>itm.on</td>
</tr>
<tr>
<td>ITM.DataTrace DataPC</td>
<td>itm.datatrace datapc</td>
<td>itm.dt dpc</td>
</tr>
<tr>
<td>ITM.PCSampler OFF</td>
<td>itm.pcsampler off</td>
<td>itm.pcs off</td>
</tr>
<tr>
<td>ITM.TTimeMode External</td>
<td>itm.ttimemode external</td>
<td>itm.tim e</td>
</tr>
<tr>
<td>ITM.CyclePrescaler 1/1</td>
<td>itm.cycleprescaler 1/1</td>
<td>itm.cp 1/1</td>
</tr>
<tr>
<td>ITM.TimeStampMode ALL</td>
<td>itm.timestampmode all</td>
<td>itm.tsm all</td>
</tr>
<tr>
<td>ITM.TraceID 16.</td>
<td>itm.traceid 16.</td>
<td>itm.tid 16.</td>
</tr>
<tr>
<td>ITM.TracePriority 2.</td>
<td>itm.tracepriority 2.</td>
<td>itm.tp 2.</td>
</tr>
</tbody>
</table>

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.
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:

```plaintext
; 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].

   ![Image of window caption with command inserted](image)

   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).

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 128.

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.

<table>
<thead>
<tr>
<th>Device</th>
<th>Command</th>
<th>Subcommand</th>
</tr>
</thead>
<tbody>
<tr>
<td>B:</td>
<td>Data.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>dump</td>
</tr>
<tr>
<td></td>
<td></td>
<td>List</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Print</td>
</tr>
<tr>
<td></td>
<td></td>
<td>View</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Set</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LOAD.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Break.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>direct</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Set</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.

::B::WinPrint.Data.dump 0x1000 /Byte

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:

<table>
<thead>
<tr>
<th>Long form</th>
<th>Short forms</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYStem.state</td>
<td>SYS or just sys</td>
</tr>
<tr>
<td></td>
<td>• You can use short forms in UPPER CASE or lower case.</td>
</tr>
<tr>
<td></td>
<td>• You can omit words in all lower-case letters, e.g. state</td>
</tr>
</tbody>
</table>

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:

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

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

  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.)

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.
**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**

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.

<table>
<thead>
<tr>
<th>OS level</th>
<th>devices</th>
<th>HELP</th>
<th>os</th>
<th>windows</th>
<th>practice</th>
<th>EDIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device level</td>
<td>::B::</td>
<td>emulate</td>
<td>Data</td>
<td>Var</td>
<td>trigger</td>
<td>devices</td>
</tr>
<tr>
<td>Command</td>
<td>B::Data.</td>
<td>[ok]</td>
<td>dump</td>
<td>View</td>
<td>Print</td>
<td>List</td>
</tr>
<tr>
<td>Sub-command</td>
<td>B::Data.List</td>
<td>[ok]</td>
<td>&lt;range&gt;</td>
<td>&lt;address&gt;</td>
<td>options</td>
<td></td>
</tr>
<tr>
<td>Parameter</td>
<td>B::Data.List</td>
<td>0x1000</td>
<td>[ok]</td>
<td>options</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>B::Data.List</td>
<td>/MarkPC</td>
<td>[ok]</td>
<td>Mark</td>
<td>Track</td>
<td>TOrder</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Device Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>::B::</td>
<td>select ICD Debugger</td>
</tr>
<tr>
<td>::E::</td>
<td>select In-Circuit Emulator TRACE32-ICE</td>
</tr>
<tr>
<td>::F::</td>
<td>select RISC Emulator TRACE32-FIRE</td>
</tr>
<tr>
<td>B::E::Data.dump 0x100</td>
<td>A TRACE32-ICE window is generated out of the ICD debugger environment</td>
</tr>
<tr>
<td>B::QUIT</td>
<td>The QUIT command is a part of the operating system and therefore, it is recognized for all devices</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CmdPOS</th>
<th>FramePOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controls the position of TRACE32 in MWI window mode</td>
<td></td>
</tr>
<tr>
<td>Controls the position of TRACE32 in MDI window mode</td>
<td></td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HISTORY.execute</td>
<td>Execute command history</td>
</tr>
<tr>
<td>HISTORY.save</td>
<td>Store command history log</td>
</tr>
<tr>
<td>HISTORY.set</td>
<td>History settings</td>
</tr>
<tr>
<td>HISTORY.size</td>
<td>Define command history log size</td>
</tr>
<tr>
<td>HISTORY.type</td>
<td>Display command history log</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classic</td>
<td>Number base is hex.</td>
</tr>
<tr>
<td>Decimal</td>
<td>Number base is decimal, C-like operators are used.</td>
</tr>
<tr>
<td>Hex (default)</td>
<td>Number base is hex, C-like operators are used.</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>Parameter Type</th>
<th>Examples</th>
<th>RADIX.Classic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td><code>&lt;address&gt; = [&lt;accessmode&gt;::] [[&lt;machineid&gt;::::] &lt;spaceid&gt;::] &lt;constant&gt;</code></td>
<td>UD:0x1000 D:0x1000 NSP:1::0x0000:0xffff00008080004 or UD:1000 or D:1000</td>
</tr>
<tr>
<td>Address Range</td>
<td>For details, see Address Range below.</td>
<td>D:0x4040--0x406F</td>
</tr>
<tr>
<td></td>
<td>D:0x4040..0x406F</td>
<td>NSP:1::0x2000:0x8080004++0xffff</td>
</tr>
<tr>
<td></td>
<td>func11--sYmbol.END(func11)</td>
<td></td>
</tr>
<tr>
<td>ASCII value</td>
<td>'A'</td>
<td></td>
</tr>
<tr>
<td>Binary mask or bit mask</td>
<td>0yX111XXX</td>
<td>or 0X111XXX!</td>
</tr>
<tr>
<td>Binary value</td>
<td>0y1, 0y0, 0y100010001</td>
<td>or 1!, or 0!, or 0100010001!</td>
</tr>
<tr>
<td>Boolean</td>
<td><code>&lt;operand1&gt;&lt;compare_operator&gt;&lt;operand2&gt;</code> or any function returning a boolean expression, such as the functions TRUE() and FOUND().</td>
<td></td>
</tr>
<tr>
<td>Decimal value</td>
<td>1.123445</td>
<td></td>
</tr>
<tr>
<td>Floating point number</td>
<td>1.3, 1.3e+34, 0.123</td>
<td></td>
</tr>
<tr>
<td>Hex mask</td>
<td>0xFX, 0xff1cxxxx</td>
<td>or 0FX or 0ff1cxxxx</td>
</tr>
<tr>
<td>Hex value</td>
<td>0x0, 0xd344, 0x1234, 0xEEEE</td>
<td>or 0 or 0d344 or 1234 or 0xEEEE</td>
</tr>
<tr>
<td>Range</td>
<td>0x10..0x20, 10.--30., 0x10--0xed 'A'--'Z'</td>
<td>or 10--0ed</td>
</tr>
</tbody>
</table>
### String (with quotation marks)
- "customer 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<br>TEST.CMM<br>A:\FOLDER\TEST.CMM<br>objs\abc.abs<br>NOTE: ‘C:TEST.C’ is not valid name!

**File name special (for 3rd-party tool integration e.g. Eclipse)** | `\program\"C:\folder\helloworld.c"`<br>`"/home/myuser/examples/demo1.cpp"

**File name UNIX/OS-9** | objs/abc.abs<br>../src/abc.def<br>~/proj/src/main.c<br>~/~~~/demo/analyzer/perf.ts
<table>
<thead>
<tr>
<th>Parameter Type</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>File name VMS</td>
<td>[.objs]abs.abs</td>
</tr>
<tr>
<td></td>
<td>[.src]abc.def</td>
</tr>
<tr>
<td></td>
<td>DISK$DISK2:[t32.font]abc.d;4</td>
</tr>
<tr>
<td>Line numbers</td>
<td>\100</td>
</tr>
<tr>
<td></td>
<td>\MCC\100</td>
</tr>
<tr>
<td></td>
<td>\module\100</td>
</tr>
<tr>
<td></td>
<td>\program&quot;C:\folder\helloworld.c&quot;\100</td>
</tr>
<tr>
<td>PRACTICE Function</td>
<td>Register(PC)</td>
</tr>
<tr>
<td></td>
<td>FOUND()</td>
</tr>
<tr>
<td></td>
<td>OS.ENV(HOME)</td>
</tr>
<tr>
<td></td>
<td>Data.Word.BigEndean(MX:0x1234)</td>
</tr>
<tr>
<td>Symbol</td>
<td>;&lt;symbolname&gt;</td>
</tr>
<tr>
<td></td>
<td>main</td>
</tr>
<tr>
<td></td>
<td>SIEVE</td>
</tr>
<tr>
<td></td>
<td>;&lt;modulename&gt;&lt;symbolname&gt;</td>
</tr>
<tr>
<td></td>
<td>MCCsieve</td>
</tr>
<tr>
<td></td>
<td>;&lt;machinename&gt;&lt;programname&gt;&lt;modulename&gt;&lt;symbolname&gt;</td>
</tr>
<tr>
<td></td>
<td>\1\linux\do_mounts\load_ramdisk</td>
</tr>
<tr>
<td></td>
<td>\Dom0\linux\do_mounts\load_ramdisk</td>
</tr>
<tr>
<td></td>
<td>;&lt;programname&gt;&lt;empty&gt;&lt;symbolname&gt; results in</td>
</tr>
<tr>
<td></td>
<td>\linux\load_ramdisk ;2 backslashes before the &lt;symbolname&gt;</td>
</tr>
<tr>
<td></td>
<td>;&lt;machinename&gt;&lt;empty&gt;&lt;empty&gt;&lt;symbolname&gt; results in</td>
</tr>
<tr>
<td></td>
<td>\Dom0\\load_ramdisk ;4 backslashes before the &lt;symbolname&gt;</td>
</tr>
<tr>
<td>Symbol Syntax:</td>
<td>&lt;symbol&gt; =  \&lt;machinename&gt;</td>
</tr>
<tr>
<td></td>
<td>\[&lt;programname&gt;]</td>
</tr>
<tr>
<td></td>
<td>[&lt;modulename&gt;]</td>
</tr>
<tr>
<td></td>
<td>&lt;symbolname&gt;</td>
</tr>
<tr>
<td></td>
<td>[&lt;symbolname&gt;]...</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[ [\&lt;programname&gt;]</td>
</tr>
<tr>
<td></td>
<td>[&lt;modulename&gt;]]</td>
</tr>
<tr>
<td></td>
<td>&lt;symbolname&gt;</td>
</tr>
<tr>
<td></td>
<td>[&lt;symbolname&gt;]...</td>
</tr>
<tr>
<td>NOTE: The use of \&lt;machinename&gt; requires that the machine spaces are enabled with the command SYStem.Option MACHINESPACES ON.</td>
<td></td>
</tr>
<tr>
<td>Symbol with special chars</td>
<td><code>nestxf1::~nestxf1</code></td>
</tr>
<tr>
<td></td>
<td><code>ops::operator&lt;&lt;=</code></td>
</tr>
</tbody>
</table>
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 \texttt{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:

\begin{verbatim}
;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--0xFFF
\end{verbatim}

Example 2: All four \texttt{Data.SAVE.Binary} commands save 0x30 bytes beginning from D:0x4040

\begin{verbatim}
;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)
\end{verbatim}
White spaces before or after operators are interpreted as separators of consecutive expressions. Values can be linked by operators.

<table>
<thead>
<tr>
<th>Type</th>
<th>Example</th>
<th>RADIX.Classic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brackets</td>
<td>(main+1)*20</td>
<td></td>
</tr>
</tbody>
</table>
| 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--0x1FF0  
"abc"<<3.  
result: "abccce"  
".-"<<10.  
result: "---------" | or 10<<2  
or 1000--1ff<<4 |
<table>
<thead>
<tr>
<th>Type</th>
<th>Example</th>
<th>RADIX.Classic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shift right</td>
<td>&quot;abc&quot;&gt;&gt;3.</td>
<td>or 10&gt;&gt;2.</td>
</tr>
<tr>
<td></td>
<td>result: &quot;aaaabc&quot;</td>
<td>or 1000--1ffff&gt;&gt;2.</td>
</tr>
<tr>
<td></td>
<td>0x10&gt;&gt;2.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>result: 0x04</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x1000--(0x1fff&gt;&gt;0x2) result: 0x1000--0x7fff</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x1000--0x1fff&gt;&gt;0x10 result: 0xff0--0x11ef</td>
<td></td>
</tr>
<tr>
<td>Multiplication</td>
<td>1000.*0x2e</td>
<td>or 1000.*2e</td>
</tr>
<tr>
<td></td>
<td>1000.*0y10</td>
<td>or 1000.*10!</td>
</tr>
<tr>
<td>Division</td>
<td>1000./0x2e</td>
<td>or 1000./2e</td>
</tr>
<tr>
<td></td>
<td>1000./0y10</td>
<td>or 1000./10!</td>
</tr>
<tr>
<td>Addition</td>
<td>0x1000+0x03</td>
<td>or 1000+3</td>
</tr>
<tr>
<td></td>
<td>sieve+0x33</td>
<td>or sieve+33</td>
</tr>
<tr>
<td>Concatenation</td>
<td>&quot;abc&quot;+&quot;def&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>or &quot;abc&quot; &quot;def&quot; result: &quot;abcdef&quot;</td>
<td></td>
</tr>
<tr>
<td>Subtraction</td>
<td>0x1000-0x34</td>
<td>or 1000-34</td>
</tr>
<tr>
<td></td>
<td>1000.-0x34</td>
<td>or 1000.-34</td>
</tr>
<tr>
<td>Comparisons</td>
<td>sieve&gt;0x1000</td>
<td>or sieve&lt;&gt;1000</td>
</tr>
<tr>
<td></td>
<td>sieve&lt;0x1000</td>
<td>or sieve&lt;=1000</td>
</tr>
<tr>
<td></td>
<td>sieve==0x1000</td>
<td>or sieve==1000</td>
</tr>
<tr>
<td></td>
<td>sieve!=0x1000</td>
<td>or sieve!=&lt;1000</td>
</tr>
<tr>
<td></td>
<td>sieve&gt;&lt;=0x1000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sieve&lt;=0x1000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Data.Byte(my_char)==('a'--'z'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Register(PC)!=P:0x1000</td>
<td></td>
</tr>
<tr>
<td>Binary AND</td>
<td>mask&amp;0x1000</td>
<td>or mask#A#1000</td>
</tr>
<tr>
<td>Binary XOR</td>
<td>mask^0x1000</td>
<td>or mask#X#1000</td>
</tr>
</tbody>
</table>
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.

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

```
E::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.

E::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
```
**Path Prefixes**

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

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

**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  r(PC)! =0x1000
 ( 

PRINT "Program stopped at address: " r(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--y.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:

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

<table>
<thead>
<tr>
<th>Address</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>C:00008233</td>
<td>W</td>
</tr>
<tr>
<td>\taskc\Global\FLAGS</td>
<td></td>
</tr>
</tbody>
</table>
```

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

<table>
<thead>
<tr>
<th>Address</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>C:00008233--00008245</td>
<td>W</td>
</tr>
<tr>
<td>\taskc\Global\FLAGS</td>
<td></td>
</tr>
</tbody>
</table>
```

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`)

©1989-2020 Lauterbach GmbH
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**

<table>
<thead>
<tr>
<th>RADIX</th>
<th>Radix mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classic</td>
<td>number base is hex - all input formats for operands and operators permitted</td>
<td></td>
</tr>
<tr>
<td>Decimal</td>
<td>number base is decimal - <em>old-fashioned operators</em> and <em>operands</em> are locked</td>
<td></td>
</tr>
<tr>
<td>Hex</td>
<td>number base is hex - old-fashioned operators and operands are locked - default</td>
<td></td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Written Value</th>
<th>Interpreted Value in Radix Mode</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td><strong>1000h==4096d</strong></td>
<td><strong>1000d==1000d</strong></td>
</tr>
<tr>
<td>P:1234</td>
<td><strong>P:1234h==P:4660d</strong></td>
<td><strong>P:1234d==P:1234d</strong></td>
</tr>
<tr>
<td>1000.</td>
<td><strong>1000d</strong></td>
<td><strong>1000d</strong></td>
</tr>
<tr>
<td>1234.</td>
<td><strong>1234d</strong></td>
<td><strong>1234d</strong></td>
</tr>
</tbody>
</table>

'd': decimal value - 'h': hexadecimal value.
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.

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

<table>
<thead>
<tr>
<th>Written Value</th>
<th>Interpreted Value</th>
<th>New Syntax for V2.00 and later</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;=V1.90</td>
<td>&gt;=V2.00</td>
<td></td>
</tr>
<tr>
<td>\texttt{0xx13} (see 1.)</td>
<td>\texttt{xx13}</td>
<td>\texttt{x13}</td>
</tr>
<tr>
<td>\texttt{0xxxx1100!} (see 1.)</td>
<td>\texttt{xxxx1100!}</td>
<td>syntax error</td>
</tr>
<tr>
<td>\texttt{0x23af} (see 1.)</td>
<td>\texttt{x23af}</td>
<td>\texttt{23af !!}</td>
</tr>
<tr>
<td>\texttt{0x23x56x} (see 1.)</td>
<td>\texttt{x23x56x}</td>
<td>\texttt{23x56x !!}</td>
</tr>
<tr>
<td>.symbolname (symbol names with . as prefix, see 2.))</td>
<td>.symbolname</td>
<td>syntax error</td>
</tr>
</tbody>
</table>

1. Now bit- and hex masks which start with 0x (e.g. \texttt{0xx10}) will always be interpreted as a hex constant (only one 'x' in value - e.g. \texttt{0x23}) or hex mask (more than one 'x' - e.g. \texttt{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 \texttt{.start'})
   - the automatic symbol prefix is set to '.' with the command \texttt{sYmbol.PREFIX} . Then the symbol name has to be entered without prefix (e.g. \texttt{start}).
Operands

Examples for operands:

<table>
<thead>
<tr>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Break.Set sieve /Alpha</td>
<td>sets alpha breakpoint at function begin of sieve</td>
</tr>
<tr>
<td>Data.List P:0x1AF</td>
<td>opens source list window at program address 0x1AF</td>
</tr>
<tr>
<td>Data.dump P:0x10--0x1ff</td>
<td>opens a data dump window from address 0x10 to 0x1ff</td>
</tr>
</tbody>
</table>

Restriction:

Not all operand formats could be used in all radix modes. Please refer to the Operand Format Table.
##Operand Format Examples (Literals)

<table>
<thead>
<tr>
<th>Operand</th>
<th>Meaning</th>
<th>Classic</th>
<th>Decimal</th>
<th>Hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>0y1010</td>
<td>binary constant</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>1010!</td>
<td>binary constant</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x12af</td>
<td>hex constant</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>1234</td>
<td>hex constant</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1234</td>
<td>decimal constant</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>1000.</td>
<td>decimal constant</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>1.2</td>
<td>float constant</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>0y10xx10</td>
<td>bitmask constant</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>10xx10!</td>
<td>bitmask constant</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>0x12axfx</td>
<td>hexmask constant</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>12axfx</td>
<td>hexmask constant</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>'a'</td>
<td>ASCII constant</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;abcdef&quot;</td>
<td>string constant</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;abc&quot;&quot;def&quot;</td>
<td>string constant with escape</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sequence for using string delimiter inside string literals string value: abc&quot;def</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>main</code></td>
<td>backticks for HLL symbols</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000--2000</td>
<td>numeric range constant</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000..2000</td>
<td>numeric range constant</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P:0x1af</td>
<td>address constant (hex)</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P:1234</td>
<td>address constant (hex)</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>P:1234</td>
<td>address constant (decimal)</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>P:1234.</td>
<td>address constant (decimal)</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P:0x1000--0x1fff</td>
<td>address range constant</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P:0x1000..0x1fff</td>
<td>address range constant</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>123ms</td>
<td>time constant</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>123ns--4.25s</td>
<td>time range constant</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>123ns..4.25s</td>
<td>time range constant</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Operand Transition Table

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

<table>
<thead>
<tr>
<th>Old-fashioned</th>
<th>New-fashioned</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;= V1.90</td>
<td>&gt;= V2.00</td>
<td></td>
</tr>
<tr>
<td>1010!</td>
<td>0y1010</td>
<td>bit constant</td>
</tr>
<tr>
<td>12af</td>
<td>0x12af</td>
<td>hex constant</td>
</tr>
<tr>
<td>10xx10!</td>
<td>0y10xx10</td>
<td>bitmask constant</td>
</tr>
<tr>
<td>12axfx</td>
<td>0x12axfx</td>
<td>hexmask constant</td>
</tr>
</tbody>
</table>
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--y.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 Formats

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
<th>Classic</th>
<th>Decimal</th>
<th>Hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>logical NOT</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>logical AND</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>^^</td>
<td>logical XOR</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>II</td>
<td>logical OR</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>N:</td>
<td>logical NOT</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>:A:</td>
<td>logical AND</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>:X:</td>
<td>logical XOR</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>:O:</td>
<td>logical OR</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>~</td>
<td>binary NOT</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>&amp;</td>
<td>binary AND</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>^</td>
<td>binary XOR</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>l</td>
<td>binary OR</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>N#</td>
<td>binary NOT</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#A#</td>
<td>binary AND</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#X#</td>
<td>binary XOR</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#O#</td>
<td>binary OR</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>negation or minus</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>+</td>
<td>plus</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>multiplication</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>/</td>
<td>division</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>%</td>
<td>modulo (reminder)</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>&lt;&lt;</td>
<td>shift left</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>&gt;&gt;</td>
<td>shift right</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Operator</td>
<td>Meaning</td>
<td>Classic</td>
<td>Decimal</td>
<td>Hex</td>
</tr>
<tr>
<td>----------</td>
<td>--------------------------</td>
<td>---------</td>
<td>---------</td>
<td>-----</td>
</tr>
<tr>
<td>&lt;</td>
<td>smaller than</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>&lt;=</td>
<td>smaller or equal than</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>==</td>
<td>smaller or equal than</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&gt;=</td>
<td>bigger or equal than</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>=&gt;</td>
<td>bigger or equal than</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>==</td>
<td>equal to</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>not equal</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;=&lt;</td>
<td>not equal</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>!=</td>
<td>not equal</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>() {}</td>
<td>parenthesis</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>--</td>
<td>range with borders</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>..</td>
<td>range with borders</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>++</td>
<td>range with offset</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
Please use the following table to convert old-fashioned operators into the new format.

<table>
<thead>
<tr>
<th>Old-fashioned</th>
<th>New-fashioned</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>N#</td>
<td>~</td>
<td>bitwise negation</td>
</tr>
<tr>
<td>#A#</td>
<td>&amp;</td>
<td>bitwise and</td>
</tr>
<tr>
<td>#X#</td>
<td>^</td>
<td>bitwise xor</td>
</tr>
<tr>
<td>#O#</td>
<td></td>
<td></td>
</tr>
<tr>
<td>N:</td>
<td>!</td>
<td>logical negation</td>
</tr>
<tr>
<td>:A:</td>
<td>&amp;&amp;</td>
<td>logical and</td>
</tr>
<tr>
<td>:X:</td>
<td>^^</td>
<td>logical xor</td>
</tr>
<tr>
<td>:O:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>!=</td>
<td>not equal comparator</td>
</tr>
<tr>
<td>&gt;&lt;</td>
<td>!=</td>
<td>not equal comparator</td>
</tr>
<tr>
<td>&lt;=</td>
<td>&lt;=</td>
<td>smaller or equal comparator</td>
</tr>
<tr>
<td>=&gt;</td>
<td>&gt;=</td>
<td>bigger or equal comparator</td>
</tr>
<tr>
<td>not available</td>
<td>%</td>
<td>modulo (remainder)</td>
</tr>
</tbody>
</table>
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 \texttt{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:

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

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.

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:

```plaintext
;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

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

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

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:

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 35.

- 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.
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”.
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.

<table>
<thead>
<tr>
<th>Visible bookmarks</th>
<th>View menu &gt; 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.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hidden bookmark</td>
<td>Recall Position returns to the position you have previously saved with Store Position. The steps below describe how to place a hidden bookmark.</td>
</tr>
<tr>
<td>Current Selection</td>
<td>Goto Selection returns you to the currently selected position or last active view (in case the selection is no longer active).</td>
</tr>
</tbody>
</table>

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.

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:

```plaintext
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.
   - 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.

```
E::Var.View \ iar196\iar196\flags
\ iar196\iar196\flags = (1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1,
E:: \ iar196\iar196\flags[3]=0
```

**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».

```
E::Data.dump

<table>
<thead>
<tr>
<th>address</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>0123</th>
</tr>
</thead>
<tbody>
<tr>
<td>SD:00002764</td>
<td>0000</td>
<td>0000</td>
<td>....</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SD:00002768</td>
<td>0000</td>
<td>0000</td>
<td>....</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SD:0000276C</td>
<td>0000</td>
<td>0000</td>
<td>....</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SD:00002770</td>
<td>0000</td>
<td>0000</td>
<td>....</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SD:00002774</td>
<td>0101</td>
<td>0101</td>
<td>....</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SD:00002778</td>
<td>0101</td>
<td>0001</td>
<td>....</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SD:0000277C</td>
<td>0100</td>
<td>0100</td>
<td>....</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SD:00002780</td>
<td>0001</td>
<td>0100</td>
<td>....</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SD:00002784</td>
<td>0001</td>
<td>0000</td>
<td>....</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

E::Data.Set SD:0x2774 %WORD %UP --- Command generated by mouse click
```
The windows in TRACE32 provide a window manager menu with special commands. For a short description of these commands, see below.

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

![Windows GUI Window Manager Menu](image)

- **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:

![Motif GUI Window Manager Menu](image)
### Short Descriptions of the Special Commands in the Window Manager Menu

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Next</strong></td>
<td>Jump to next window.</td>
</tr>
</tbody>
</table>
| **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.                                                                                       |
| **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**        | 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**. |
<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>To Clipboard</td>
<td><strong>To Clipboard</strong> copies the visible window contents as text to the clipboard. See also PRinTer.</td>
</tr>
<tr>
<td>To Clipboard all</td>
<td><strong>To Clipboard All</strong> behaves within a TRACE32 window as if you scroll to the top of the terminal buffer and choose <strong>To Clipboard</strong>, then scroll down one visible terminal page and do the next <strong>To Clipboard</strong>, and so on. <strong>NOTE</strong>: 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.</td>
</tr>
<tr>
<td>Window Screen-shot to File</td>
<td>Captures a screenshot of the active window and opens the <strong>Save Window Screenshot</strong> dialog. Enter file name and select file type (PNG, GIF etc.) See also SCreenShot.</td>
</tr>
<tr>
<td>Window Screen-shot to Clipboard</td>
<td>Copies a screenshot of the visible part of the window to the clipboard.</td>
</tr>
<tr>
<td>Store Command</td>
<td>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.</td>
</tr>
</tbody>
</table>
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.
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**.

<table>
<thead>
<tr>
<th><strong>WinPrint.&lt;command&gt;</strong></th>
<th>Print one window (full printer size) to file</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WinPRT</strong></td>
<td>Make hardcopy of existing window</td>
</tr>
<tr>
<td><strong>PRinTer.HardCopy</strong></td>
<td>Print all windows on screen</td>
</tr>
<tr>
<td><strong>PRinTer.select</strong></td>
<td>Select type of printer</td>
</tr>
<tr>
<td><strong>PRinTer.ChipBoard</strong></td>
<td>Re-route printer output to clipboard in specified format</td>
</tr>
<tr>
<td><strong>PRinTer.Area</strong></td>
<td>Re-route printer output to <strong>AREA</strong> window in specified format</td>
</tr>
</tbody>
</table>
Saving Window Contents

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.

**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
```

---

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.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRinTer.OPEN</td>
<td>Open file and re-route multiple printer outputs to this file</td>
</tr>
<tr>
<td>PRinTer.FILE</td>
<td>Define file for single printer output and select output format for file</td>
</tr>
<tr>
<td>PRinTer.CLOSE</td>
<td>Close file after multiple printer outputs</td>
</tr>
<tr>
<td>WinPrint</td>
<td>Print one window (full size) to file</td>
</tr>
<tr>
<td>PRinTer.export</td>
<td>Export CSV-formatted printer output to file</td>
</tr>
</tbody>
</table>
Windows with some special behavior can be created by the following commands:

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

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.

<table>
<thead>
<tr>
<th>WinFIND</th>
<th>Search for a text string in a window</th>
</tr>
</thead>
<tbody>
<tr>
<td>WinPAN</td>
<td>Scroll window</td>
</tr>
</tbody>
</table>

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).

**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].

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

©1989-2020 Lauterbach GmbH
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.

Example 2 - Tracking by going to an address:

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.

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 At 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.
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.

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

For information about wildcard characters and path prefixes supported with the file and folder handling commands, see “File Names”, page 49 and “Path Prefixes”, page 50.
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:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLOSE</td>
<td>Close file</td>
</tr>
<tr>
<td>Data.WRITESTRING</td>
<td>Write string from target memory to PRACTICE file</td>
</tr>
<tr>
<td>OPEN</td>
<td>Open data file</td>
</tr>
<tr>
<td>READ</td>
<td>Read data from file</td>
</tr>
<tr>
<td>Var.EXPORT</td>
<td>Export variables in CSV format to file</td>
</tr>
<tr>
<td>Var.WRITE</td>
<td>Write variables to file</td>
</tr>
<tr>
<td>WinPrint</td>
<td>Print window</td>
</tr>
<tr>
<td>WRITE</td>
<td>Write data to file</td>
</tr>
<tr>
<td>WRITEB</td>
<td>Write binary data to file</td>
</tr>
<tr>
<td>APPEND</td>
<td>Append data to file</td>
</tr>
</tbody>
</table>
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.

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

```plaintext
; 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
```

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

```plaintext
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.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRinTer.select</td>
<td>Select printer type and output style</td>
</tr>
<tr>
<td>PRinTer.FILE</td>
<td>Define file for single printer output and select output format for file (ASCII, CSC, XML, etc.)</td>
</tr>
<tr>
<td>PRinTer.OPEN</td>
<td>Open file and re-route multiple printer outputs to this file</td>
</tr>
<tr>
<td>PRinTer.CLOSE</td>
<td>Close file after multiple printer outputs</td>
</tr>
</tbody>
</table>

©1989-2020 Lauterbach GmbH
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--0xffff ; print window

; example for copy to file operation
PRinTer.FILE test.lst ; open file for printing
WinPrint.Data.dump 0x0--0xffff ; 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--0xffff ; print Data.dump window to file
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 (Emulator, Analyzer).

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SETUP.ASCIITEXT</td>
<td>Configure ASCII text display</td>
</tr>
<tr>
<td>SETUP.BAKfile</td>
<td>Set backup file mode</td>
</tr>
<tr>
<td>SETUP.CClear</td>
<td>Erase caches</td>
</tr>
<tr>
<td>SETUP.COLOR</td>
<td>Configure colors</td>
</tr>
<tr>
<td>SETUP.DEVNAME</td>
<td>Set logical device name</td>
</tr>
<tr>
<td>SETUP.EDITOR</td>
<td>Configure the TRACE32 editors</td>
</tr>
<tr>
<td>SETUP.EDITEXT</td>
<td>Define an external editor</td>
</tr>
<tr>
<td>SETUP.EXTension</td>
<td>Set default file name extensions</td>
</tr>
<tr>
<td>SETUP.HOLDDIR</td>
<td>Configure working directory</td>
</tr>
<tr>
<td>SETUP.ICONS</td>
<td>Display icons in popup menus</td>
</tr>
<tr>
<td>SETUP.QUITDO</td>
<td>Define a PRACTICE file that is executed automatically when you quit a TRACE32 session</td>
</tr>
<tr>
<td>SETUP.ReDraw</td>
<td>Update whole screen</td>
</tr>
<tr>
<td>SETUP.RESTARTDO</td>
<td>Define restart PRACTICE file</td>
</tr>
<tr>
<td>SETUP.SOUND</td>
<td>Set sound generator mode</td>
</tr>
<tr>
<td>SETUP.STATistic</td>
<td>Cache and memory usage</td>
</tr>
<tr>
<td>SETUP.TabSize</td>
<td>Configure tab width</td>
</tr>
<tr>
<td>SETUP.TIMEFORM</td>
<td>Select scientific time format</td>
</tr>
<tr>
<td>SETUP.URATE</td>
<td>Limit window update rate</td>
</tr>
<tr>
<td>SETUP.WARNSTOP</td>
<td>Configure PRACTICE stops</td>
</tr>
<tr>
<td>ZERO</td>
<td>Set time reference</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOG.OPEN</td>
<td>Generate command log file and start logging</td>
</tr>
<tr>
<td>LOG.CLOSE</td>
<td>Terminate logging and close command log file</td>
</tr>
<tr>
<td>LOG.type</td>
<td>Show contents of the command log file</td>
</tr>
<tr>
<td>LOG.ON</td>
<td>Resume logging</td>
</tr>
<tr>
<td>LOG.OFF</td>
<td>Pause logging</td>
</tr>
<tr>
<td>LOG.toAREA</td>
<td>Log commands by writing them to an AREA window</td>
</tr>
<tr>
<td>LOG.DO</td>
<td>Log the call hierarchy of PRACTICE scripts (*.cmm)</td>
</tr>
</tbody>
</table>

For more information, see “Logging the Call Hierarchy of PRACTICE Scripts” (practice_user.pdf).

Example:

```plaintext
LOG.OPEN ; opens file 't32.log'
...
LOG.CLOSE ; commands are logged
LOG.OFF ; switch off log function
...
LOG.ON ; commands are not logged
...
LOG.ON ; switch on log function
...
LOG.CLOSE ; close file and terminate log function
```
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 123.

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: BUTTON "Click Me" "Data.List"

Multi-line script:

```
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:

```c
LOCAL &file                  ;declare PRACTICE macro
DIALOG.view                  ;start the dialog definition
(POS 1. 1. 15.
  myLabel: EDIT "" ""
  POSX 1. 6.
  BUTTON "[:edit]File"  "DIALOG.SetFileW 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:

```c
LOCAL &file                  ;declare PRACTICE macro
DIALOG.view dialog.dlg

STOP
&file=DIALOG.STRing(LAB)    ;get return value of EDIT box by label
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.SetFileW 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

```plaintext
DIALOG.view
{
;your comment
ICON ":objects"
;your comment
TEXT "Hello World!"
}
```

Wrong comment position

```plaintext
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 can 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 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:

<table>
<thead>
<tr>
<th>Dialog Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAR</td>
<td>Define a progress bar</td>
</tr>
<tr>
<td>BOX &quot;&lt;text&gt;&quot;</td>
<td>Define a decorative border</td>
</tr>
<tr>
<td>BUTTON &quot;&lt;text&gt;&quot; [&lt;command&gt;]</td>
<td>Define a button</td>
</tr>
<tr>
<td>CHECKBOX &quot;&lt;text&gt;&quot; [&lt;command&gt;]</td>
<td>Define a check box</td>
</tr>
<tr>
<td>CHOOSEBOX &quot;&lt;text&gt;&quot; [&lt;command&gt;]</td>
<td>Define a choose box</td>
</tr>
<tr>
<td>CLOSE [&lt;command&gt;]</td>
<td>Catch window close</td>
</tr>
<tr>
<td>COMBOBOX &quot;&lt;list_items&gt;&quot; [&lt;command&gt;]</td>
<td>Define a combo box</td>
</tr>
<tr>
<td>DEFBUTTON &quot;&lt;text&gt;&quot; [&lt;command&gt;]</td>
<td>Define the default button</td>
</tr>
<tr>
<td>DEFCOMBOBOX &quot;&lt;list_items&gt;&quot; [&lt;command&gt;]</td>
<td>Define a combo box</td>
</tr>
<tr>
<td>DEFEDIT &quot;&lt;initial_text&gt;&quot; [&lt;command&gt;]</td>
<td>Define an edit control</td>
</tr>
<tr>
<td>DEFHOTCOMBOBOX &quot;&lt;list_items&gt;&quot; [&lt;command&gt;]</td>
<td>Define a default hot combo box</td>
</tr>
<tr>
<td>DEFHOTEDIT &quot;&lt;initial_text&gt;&quot; [&lt;command&gt;]</td>
<td>Define a hot edit control</td>
</tr>
<tr>
<td>DEFMDEDIT &quot;&lt;initial_text&gt;&quot; [&lt;command&gt;]</td>
<td>Define a default multiline edit control</td>
</tr>
<tr>
<td>DLISTBOX &quot;&lt;list_items&gt;&quot; [&lt;command&gt;]</td>
<td>Define a default list box</td>
</tr>
<tr>
<td>DYNAMIC &quot;&lt;initial_text&gt;&quot; [&lt;command&gt;]</td>
<td>Define a dynamic, single-line area</td>
</tr>
<tr>
<td>DYNCOMBOBOX &quot;&lt;list_items&gt;&quot; [&lt;command&gt;]</td>
<td>Define a dynamic combo box</td>
</tr>
<tr>
<td>DYNDEFCOMBOBOX &quot;&lt;list_items&gt;&quot; [&lt;command&gt;]</td>
<td>Define a default dynamic combo box</td>
</tr>
<tr>
<td>DYNDEFHOTCOMBOBOX &quot;&lt;list_items&gt;&quot; [&lt;command&gt;]</td>
<td>Define a dynamic default hot combo box</td>
</tr>
<tr>
<td>DYNHOTCOMBOBOX &quot;&lt;list_items&gt;&quot; [&lt;command&gt;]</td>
<td>Define a dynamic hot combo box</td>
</tr>
<tr>
<td>DYNLTEXT &quot;&lt;initial_text&gt;&quot; [&lt;command&gt;]</td>
<td>Define a dynamic single-line text area in bold and large font size</td>
</tr>
<tr>
<td>DYNPULLDOWN &quot;&lt;list_items&gt;&quot; [&lt;command&gt;]</td>
<td>Define a dynamic pull-down list</td>
</tr>
<tr>
<td>DYNTEXT &quot;&lt;initial_text&gt;&quot; [&lt;command&gt;]</td>
<td>Define a dynamic, single-line text area in regular font size</td>
</tr>
<tr>
<td>EDIT &quot;&lt;initial_text&gt;&quot; [&lt;command&gt;]</td>
<td>Define an edit control</td>
</tr>
<tr>
<td>HEADER &quot;&lt;text&gt;&quot;</td>
<td>Define window header</td>
</tr>
<tr>
<td>HELP</td>
<td>Define a help icon</td>
</tr>
<tr>
<td>HOTEDIT &quot;&lt;initial_text&gt;&quot; [&lt;command&gt;]</td>
<td>Define a hot edit control</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>HOTCOMBOBOX</td>
<td>Define a hot combo box</td>
</tr>
<tr>
<td>ICON</td>
<td>New icon in top left corner of dialog</td>
</tr>
<tr>
<td>INFOTEXT</td>
<td>Define a multiline info text box on a dialog</td>
</tr>
<tr>
<td>LINE</td>
<td>Define a decorative line</td>
</tr>
<tr>
<td>LISTBOX</td>
<td>Define a list box</td>
</tr>
<tr>
<td>LTEXT</td>
<td>Static, single-line text area in bold and large font size</td>
</tr>
<tr>
<td>MEDIT</td>
<td>Define a multiline edit control</td>
</tr>
<tr>
<td>MLISTBOX</td>
<td>Define a multiline list box</td>
</tr>
<tr>
<td>NAME</td>
<td>Define an internal dialog name</td>
</tr>
<tr>
<td>POS</td>
<td>Define position and size</td>
</tr>
<tr>
<td>POSX</td>
<td>Define position and size on the x-axis</td>
</tr>
<tr>
<td>POSY</td>
<td>Define position and size on the y-axis</td>
</tr>
<tr>
<td>PULLDOWN</td>
<td>Define a static pull-down list</td>
</tr>
<tr>
<td>SPACE</td>
<td>Define space</td>
</tr>
<tr>
<td>STATIC</td>
<td>Place an icon in a dialog</td>
</tr>
<tr>
<td>TEXT</td>
<td>Define a text item</td>
</tr>
<tr>
<td>TEXTBUTTON</td>
<td>Define a flat button with text only</td>
</tr>
<tr>
<td>TREEBUTTON</td>
<td>Define a +/- toggle button</td>
</tr>
<tr>
<td>UPDATE</td>
<td>Executes commands periodically</td>
</tr>
<tr>
<td>VLINE</td>
<td>Define a decorative vertical line</td>
</tr>
</tbody>
</table>
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.

\[
\&retVal = \text{DIALOG.STRing(<label>)}
\]

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:

```
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... )
```

```
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 = \text{DIALOG.STRing(myLAB)} ; get return value by label
DIALOG.END
DIALOG.MESSAGE "The EDIT box contains the address &retVal"
```
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.

```pascal
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
- Context-Sensitive Help
- Structure of the Help System
- Configure the Help System
- Recommendations for Choosing a PDF Viewer
- Bookmarks for Help Topics
- Troubleshooting the Help System
- Change the Installation Path of the PDF Files

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

The following **HELP** commands are available:

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

©1989-2020 Lauterbach GmbH
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.
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

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 99.

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.
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.

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 use 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 102.

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.
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)
Troubleshooting the Help System

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.


Situation: The TRACE32 message line displays this warning message:

```
B:|
| components | trace | Data | Var | List | PERF | SYStem | Step | Go | Break
```

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)
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/
   ```

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:

     ![Environment Variables Dialog]

     - 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:

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

SumatraPDF:

```bash
@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:

```bash
@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:

```bash
@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):

```bash
#!/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:

```bash
#!/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).
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**

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 105.

Adobe Reader:

```bash
@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.

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.
The **InterCom** system allows the exchange of data between different TRACE32 systems. The exchange is based on TCP/IP. 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.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InterCom.Evaluate</td>
<td>Evaluates InterCom</td>
</tr>
<tr>
<td>InterCom.execute</td>
<td>Remote execute command line</td>
</tr>
<tr>
<td>InterCom.PING</td>
<td>Test the InterCom system</td>
</tr>
<tr>
<td>InterCom.PipeCLOSE</td>
<td>Close named pipe</td>
</tr>
<tr>
<td>InterCom.PipeOPEN</td>
<td>Open named pipe</td>
</tr>
<tr>
<td>InterCom.PipeREAD</td>
<td>Read from named pipe</td>
</tr>
<tr>
<td>InterCom.PipeWRITE</td>
<td>Write to named pipe</td>
</tr>
<tr>
<td>TargetSystem.state</td>
<td>Show overview of multicore system</td>
</tr>
<tr>
<td>TargetSystem.NewInstance</td>
<td>Start new TRACE32 PowerView instance</td>
</tr>
</tbody>
</table>
Example: The TRACE32 PowerView instance named **firstInst** starts another instance named **secondInst** for the purpose of debugging two cores of an AMP system.

```plaintext
; 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
```
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.

For more information on finding serial numbers, see “Serial Numbers” (updates.pdf).

For more information on using license numbers, see “The LICENSE Command” (updates.pdf).

The following commands are described in the “PowerView Command Reference” (ide_ref.pdf).

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VERSION.HARDWARE</td>
<td>Displays the version of the used debug hardware</td>
</tr>
<tr>
<td>VERSION.SOFTWARE</td>
<td>Displays detailed information about the used TRACE32 software</td>
</tr>
<tr>
<td>VERSION.view</td>
<td>Displays window with version info</td>
</tr>
<tr>
<td>LICENSE.state</td>
<td>Displays the currently used maintenance contract</td>
</tr>
<tr>
<td>LICENSE.List</td>
<td>Displays all license information</td>
</tr>
<tr>
<td>LICENSE.UPDATE</td>
<td>Updates the maintenance contract inside a debug cable</td>
</tr>
</tbody>
</table>
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 set in the SETUP.EDITOR.state window:

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).
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)
- Automatic script formatting and command expansion.

For a detailed description of available configuration options see the \texttt{SETUP.EDITOR} command group.

**Keyboard Shortcuts**

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

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTRL+SPACE</td>
<td>Open Auto Completion box. See also \texttt{SETUP.EDITOR.AutoSuggest}.</td>
</tr>
<tr>
<td>CTRL+A</td>
<td>Select all text</td>
</tr>
<tr>
<td>CTRL+C</td>
<td>Copy selected text to clipboard</td>
</tr>
<tr>
<td>CTRL+D</td>
<td>Comment line or selection (D = Disable)</td>
</tr>
<tr>
<td>CTRL+E</td>
<td>Uncomment line or selection (E = Enable)</td>
</tr>
<tr>
<td>CTRL+F</td>
<td>Open \texttt{Find} dialog</td>
</tr>
<tr>
<td>CTRL+G</td>
<td>Open \texttt{Goto Line} dialog</td>
</tr>
<tr>
<td>CTRL+H</td>
<td>Open \texttt{Replace} dialog</td>
</tr>
<tr>
<td>CTRL+S</td>
<td>Save document</td>
</tr>
<tr>
<td>CTRL+V</td>
<td>Paste from clipboard</td>
</tr>
<tr>
<td>CTRL+X</td>
<td>Cut (copy to clipboard and delete)</td>
</tr>
<tr>
<td>CTRL+Y</td>
<td>Redo</td>
</tr>
<tr>
<td>CTRL+Z</td>
<td>Undo</td>
</tr>
</tbody>
</table>
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].

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 119.
Overview of the Various TRACE32 Editors

General-purpose editor:

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

Special-purpose editors:

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

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). |
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
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:

<table>
<thead>
<tr>
<th>Icon Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUTTON</td>
<td>Define a button</td>
</tr>
<tr>
<td>DEFINE</td>
<td>Define the default button</td>
</tr>
<tr>
<td>DYNAMIC</td>
<td>Define a dynamic, single-line area</td>
</tr>
<tr>
<td>ICON</td>
<td>Define an icon for the top left corner of a dialog</td>
</tr>
<tr>
<td>MENUITEM</td>
<td>Define an item in a menu, popup menu or a local button</td>
</tr>
<tr>
<td>STATIC</td>
<td>Define a non-dynamic area in a dialog</td>
</tr>
<tr>
<td>TOOLITEM</td>
<td>Define a button in the toolbar</td>
</tr>
</tbody>
</table>

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 123.

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 124.

<table>
<thead>
<tr>
<th>Command</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>BITMAPEDIT</td>
<td>Open the bitmap editor</td>
</tr>
</tbody>
</table>
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"

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...”).
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.:

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

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

   **Examples:**

   ```plaintext
   ; 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.

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

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
   ```

   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.
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).
Shortcuts

**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).

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.

<table>
<thead>
<tr>
<th>F2</th>
<th>Single step</th>
</tr>
</thead>
<tbody>
<tr>
<td>F3</td>
<td>Step over function call or subroutines</td>
</tr>
<tr>
<td>F4</td>
<td>Step Diverge Path: Leave loops, go till something new happens.</td>
</tr>
<tr>
<td>F5</td>
<td>Go return / Go to the last instruction</td>
</tr>
<tr>
<td>F6</td>
<td>Go up / return to the caller function</td>
</tr>
<tr>
<td>Shift+Tab</td>
<td>Go / Start real-time execution</td>
</tr>
<tr>
<td>F7</td>
<td>Break / Stop real-time execution</td>
</tr>
<tr>
<td>F8</td>
<td>Toggle between the debug modes HLL and MIX</td>
</tr>
<tr>
<td>Up and down arrow keys</td>
<td>Go up / down in the command line history.</td>
</tr>
</tbody>
</table>

**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 .)

**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.

(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.
   - 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:

<table>
<thead>
<tr>
<th>Format:</th>
<th>(&lt;type&gt;).&lt;year&gt;.&lt;month&gt;.&lt;build_number&gt;)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt;type&gt;):</td>
<td>(R</td>
</tr>
</tbody>
</table>

- **<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