

Training Simulator and Demo Software

MANUAL

Training Simulator and Demo Software

[TRACE32 Online Help](#)

[TRACE32 Directory](#)

[TRACE32 Index](#)

TRACE32 Training	
Training Simulator and Demo Software	1
About the Demo	3
Starting the TRACE32 Simulator	3
User Interface - TRACE32 PowerView	4
TRACE32 Command Line and Softkeys	6
Window Captions - What Makes Them Special in TRACE32?	7
Debugging the Program	8
Basic Debug Commands	8
Debug Modes	10
Displaying the Stack Frame	12
Breakpoints	13
Setting Breakpoints	13
Setting Read/Write Breakpoints	14
Listing all Breakpoints	15
Variables	16
Displaying Variables	16
Displaying Variables of the Current Program Context	18
Using the Symbol Browser	18
Formatting Variables	19
Modifying Variables	21
Memory	22
Displaying Memory	22
Modifying Memory	23
Peripheral View	24

About the Demo

What is this? This is a guided tour through TRACE32 - a tutorial. 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.

How long does this tutorial take? 0.5 to 1 hrs.

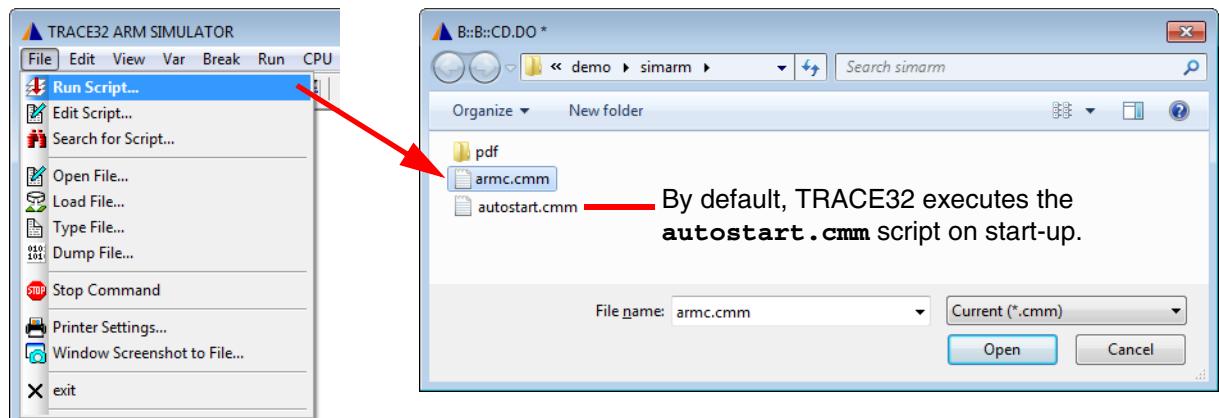
How can I learn most from this tutorial? Work completely through all chapters in sequence and then do the quiz at the end.

Where can I download the TRACE32 Simulator for the hands-on debug session? From: <https://www.lauterbach.com/download.html>. You do not need any hardware for this tutorial.

Starting the TRACE32 Simulator

1. Unzip the downloaded file. You do not need to install the TRACE32 Simulator.
2. Double-click the `t32m<architecture>.exe` file (e.g. `t32marm.exe`) to start the demo debug session. When the TRACE32 Instruction Set Simulator starts, a start-up PRACTICE script that sets up a debug session is automatically executed.

You can manually execute the same start-up PRACTICE script by choosing **File** menu > **Run Script**.



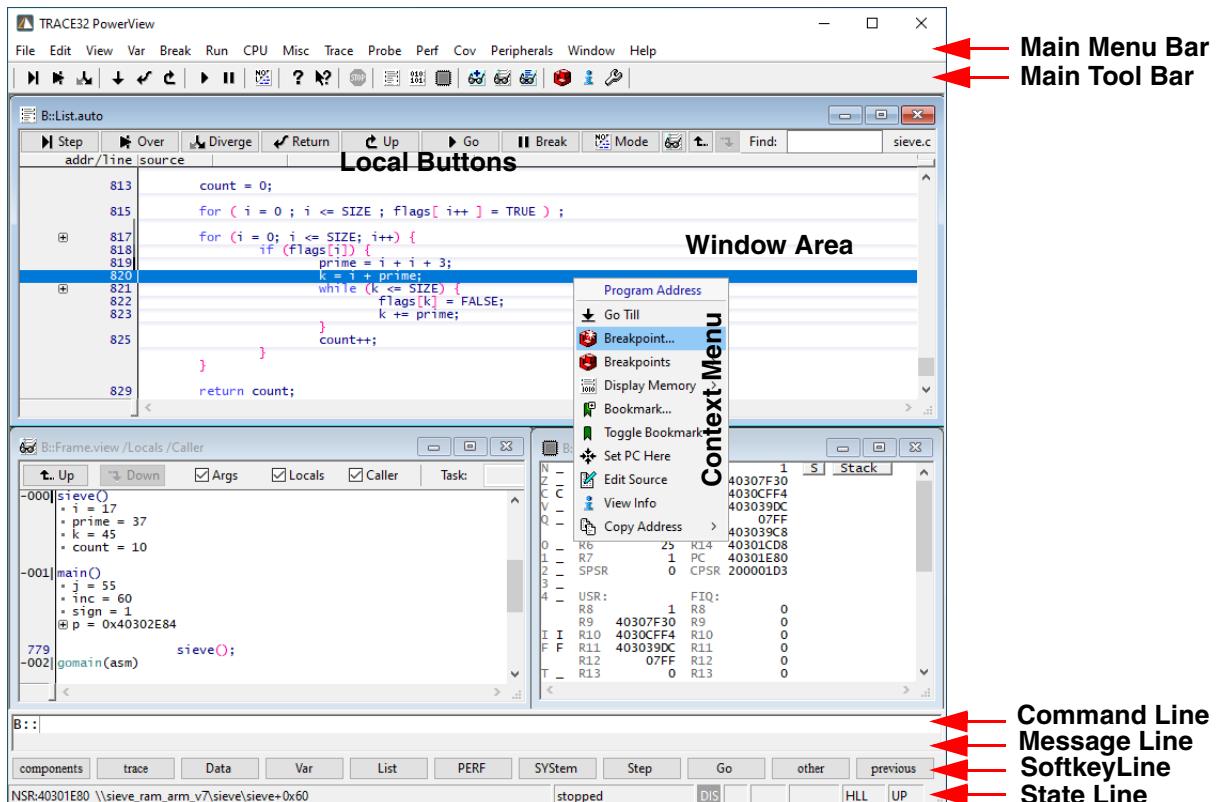
PRACTICE, the Lauterbach script language, is used for automating tests, configuring the TRACE32 PowerView GUI and your debug environment.

For our demo debug session, the PRACTICE start-up script `armc.cmm` loads the application program `armle.axf` and generates a TRACE32 internal symbol database out of the loaded information.

User Interface - TRACE32 PowerView

The graphical user interface (GUI) of TRACE32 is called TRACE32 PowerView.

The following screenshot presents the main components of this interface.

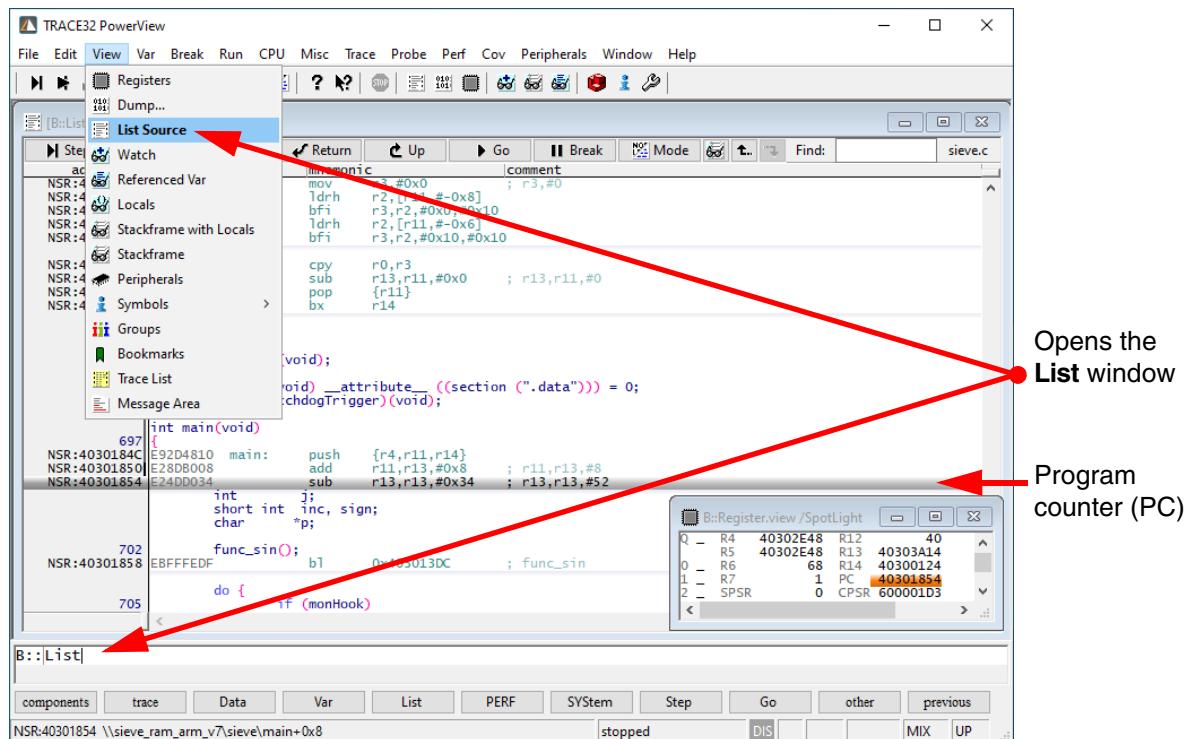


We'll briefly explain the GUI using the **List** command and **List** window as an example. For a more comprehensive introduction, a video tutorial about the TRACE32 PowerView GUI is available here: support.lauterbach.com/kb/articles/introduction-to-trace32-gui

To open the **List** window, do one of the following:

- Choose **View > List Source from the menu**
- At the TRACE32 command line, type: **List** (or **L**)

The **List** window displays the code in both assembler mnemonic and HLL (High-Level Language). HLL refers to the programming language of your source code, e.g. C or C++.



In the **List** window, the gray bar indicates the position of the program counter (PC). In the screenshot above, it is located at the symbolic address of the label **main**.

A video tutorial about the source code display in TRACE32 is available here:
support.lauterbach.com/kb/articles/displaying-the-source-code

To summarize, you can execute commands in TRACE32 PowerView using the following methods:

1. Menus on the menu bar
2. Buttons on the main toolbar and the buttons on the toolbars of TRACE32 windows
3. Context menus in TRACE32 windows
4. Using commands via the TRACE32 command line.

TRACE32 Command Line and Softkeys

TRACE32 commands are **not** case sensitive: `register.view` is the same as `Register.view`

- UPPER CASE letters indicate the short forms of commands and must not be omitted.
- All lower case letters can be omitted.

This makes short forms an efficient time saver when entering frequently-used commands in the command line.

Examples:

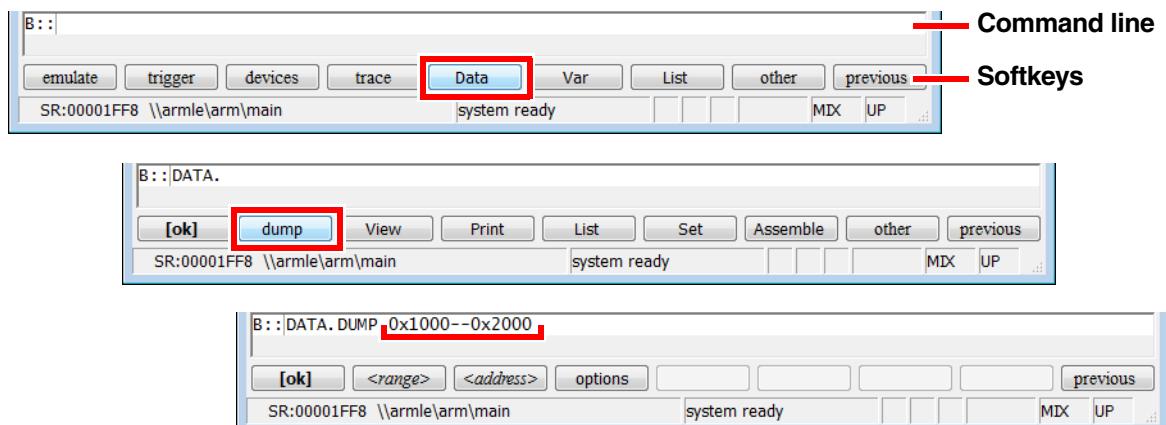
- Instead of the long form `Register.view` type just the short form `r` or `R`
- Instead of the long form `List` type just the short form `l` or `L`

The softkeys are located below the command line. The camel casing (i.e. upper and lower case letters) on any softkey indicates the long form of a command. The softkeys guide you through the command input, displaying all possible commands and parameters.

Example - To assemble the Data.dump command using the softkeys:

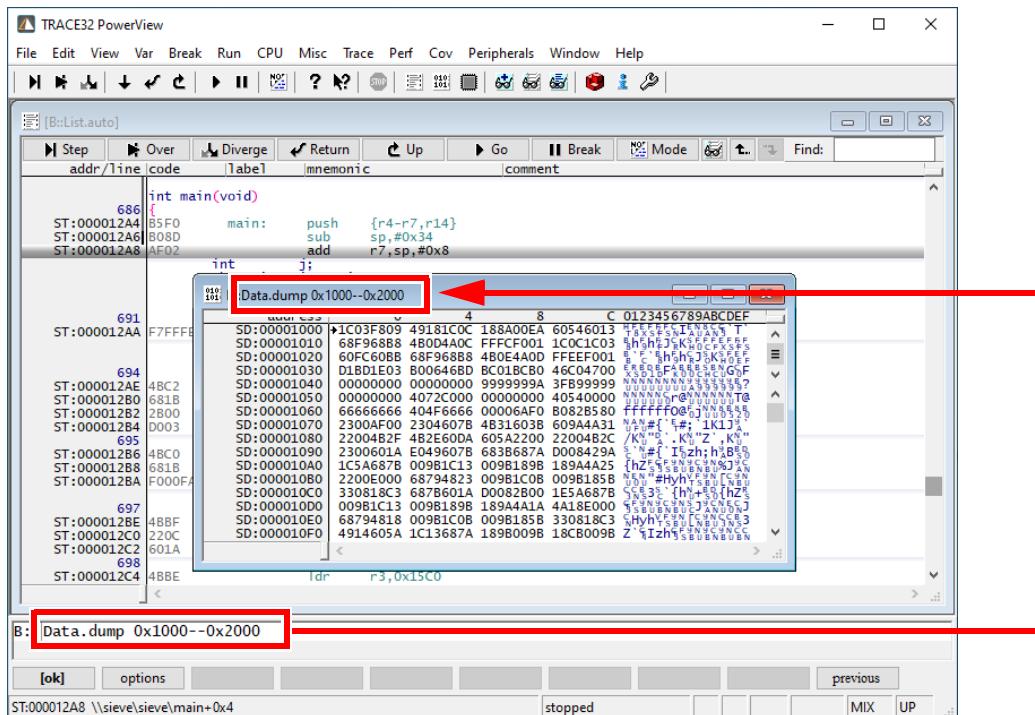
1. Click **Data**.
2. Click **dump**.
3. Type the `<range>` or `<address>` you want to dump. For example, `0x1000--0x2000`
4. Click **[ok]** to execute the command.

The **Data.dump** window will open.



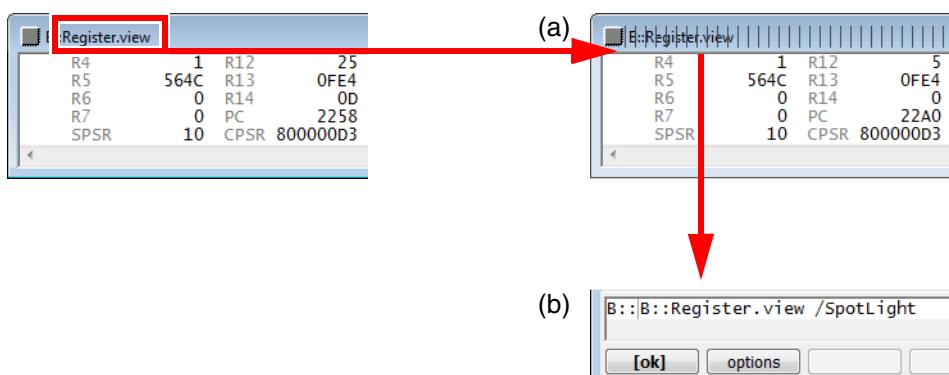
Window Captions - What Makes Them Special in TRACE32?

The command used to open a window is displayed in the window caption, along with any parameters and options used.



You can **re-insert** a command from a window caption (a) into the command line (b) in order to modify the command. Let's do this with the **Register** window.

1. Choose **View > Register** from the menu.
2. Right-click the window caption (a).
3. Modify the command, e.g. by adding the **/SpotLight** option: This will highlight changed registers.



4. Click **[ok]** to execute the modified command.
5. Click **Single Step** on the TRACE32 toolbar. Changed registers are highlighted immediately.

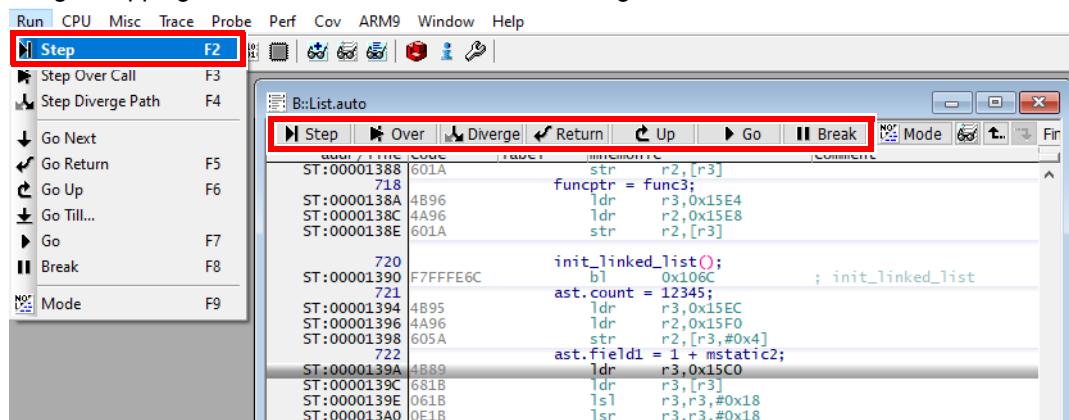
Debugging the Program

Basic Debug Commands

The basic debug commands are accessible from:

- the **Run** menu
- the toolbar of the **List** window
- the main toolbar
- the TRACE32 command line.

Single stepping  is one of these fundamental debug commands.



Single Step

Step over function calls or subroutines

Step till next unreached line

Go to the next code line written in the program listing

Useful e.g. to leave loops

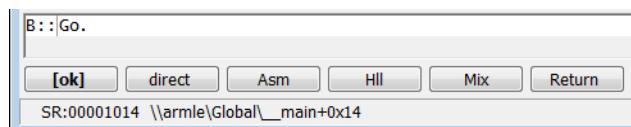
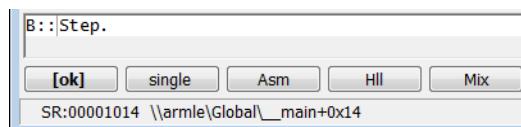


Stop the program execution

Go / Start program execution

Go Up / return to the caller function

Go Return / Go to the last instruction of the current function



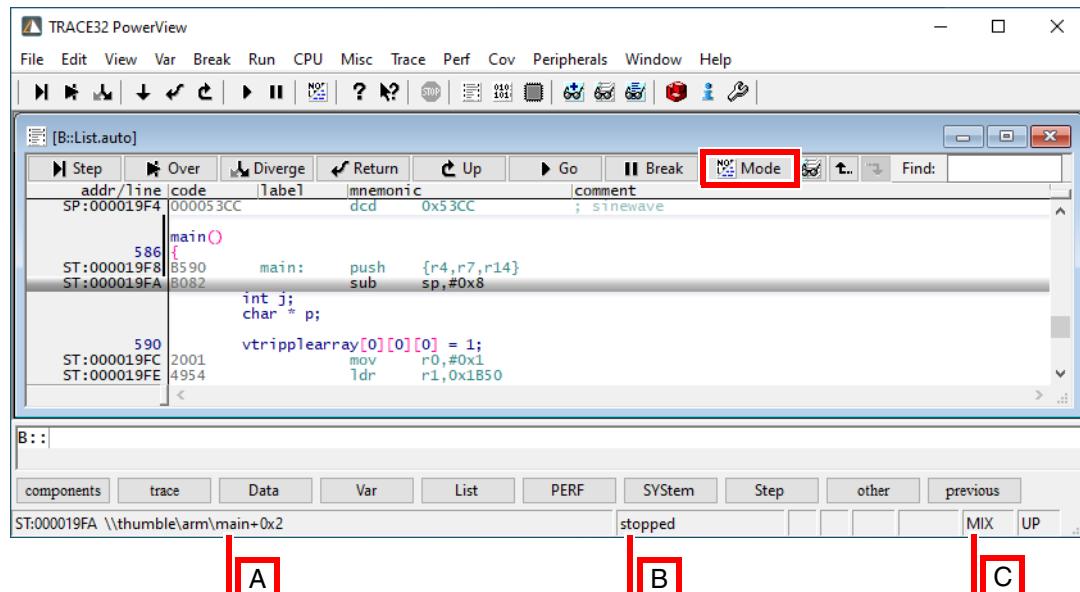
TRACE32 PowerView also offers more complex debug control commands. For example, you can step until an expression changes or becomes true.

Example:

```
Var.Step.Till i>11. ; single-steps the program until the
; variable i becomes greater than 11.
; Please note that TRACE32 uses a dot to
; denote decimal numbers.
```

Debug Modes

Take a look at the state line at the bottom of the TRACE32 PowerView main window:



The state line provides the following information

- A** The (symbolic) address of the current cursor position.
The program counter (PC) is highlighted in gray.
- B** The state of the debugger: **stopped** indicates that the program execution is stopped. At this point, you can inspect or modify memory, for example.
- C** The state line displays the currently selected debug mode, which can be:
 - **HLL** (High Level Language)
 - **ASM** (assembler)
 - **MIXed** mode showing both HLL and its corresponding assembler mnemonic.

1. On the toolbar of the **List** window, click **Mode** to toggle the debug mode to **HLL**.

Debug mode HLL

Debug mode MIX

2. Click **Step**.

In HLL mode, this action moves the program execution to the next source code line.

3. Click **Mode** again to toggle the debug mode to **MIX**.

4. Click **Step**.

This time, the step executes one assembler instruction.

5. Right-click a code line, then select **Go Till**.

Program execution starts, and stops when the program reaches the selected code line.

TRACE32 PowerView

File Edit View Var Break Run CPU Misc Trace Perf Cov Peripherals Window Help

[B::List.auto]

addr/line code label mnemonic comment

607	ast.count = 12345;	ldr	r0,0x1B5C
ST:00001A28	484C	ldr	r1,0x1B60
ST:00001A2A	494D	str	r0,[r1,#0x4]
ST:00001A2C	6048	ldr	r0,0x1B60
608	ast.left = *	ldr	r0,[r1,#0x8]
ST:00001A2E	6088	str	r0,[r1,#0x8]
609	ast.field1 = 1;	ldr	r0,[r0,#0x10]
ST:00001A32	6900	lsr	r0,r0,#0x2
ST:00001A34	0880	lsl	r0,r0,#0x2
ST:00001A36	0800	mov	r3,#0x1
ST:00001A38	2301	orr	r0,r3
ST:00001A3A	4318	str	r0,[r1,#0x10]
ST:00001A3C	6108	ldr	r0,0x1B60
610	ast.field2 = 2;	ldr	r0,[r0,#0x10]
ST:00001A3E	4848	ldr	r0,0x1B60
ST:00001A40	6900	ldr	r0,[r0,#0x10]
ST:00001A42	231C	mov	r3,#0x1C
ST:00001A44	4398	bic	r0,r3
ST:00001A46	2308	mov	r3,#0x8
ST:00001A48	4318	orr	r0,r3
ST:00001A4A	6108	str	r0,[r1,#0x10]
ST:00001A4C	06C0	lsl	r0,r0,#0x1B
612	ast = func4(ast);	mov	r0,r13
ST:00001A4E	4668	add	r1,#0x0C
ST:00001A50	310C	ldm	r1!,{r4,r7}
ST:00001A52	C990	stmia	r0!,{r4,r7}
ST:00001A54	0C90		

Program Address

- Go Till
- Breakpoint...
- Breakpoints >
- Display Memory >
- Bookmark...
- Toggle Bookmark
- Set PC Here
- Edit Source
- View Info
- Copy Address >
- Go Till There
- List There
- Assemble...
- Modify...
- Patch...

B::

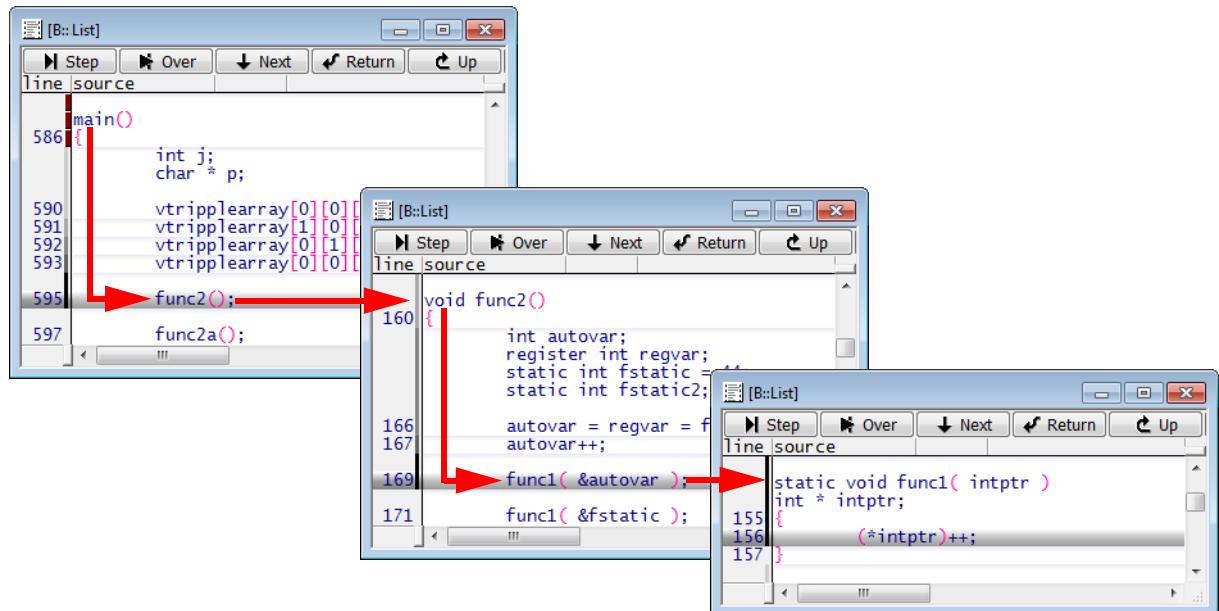
components trace Data Var List

ST:00001A30 \\thumble\\arm\\main+0x38 stopped

MIX UP

Displaying the Stack Frame

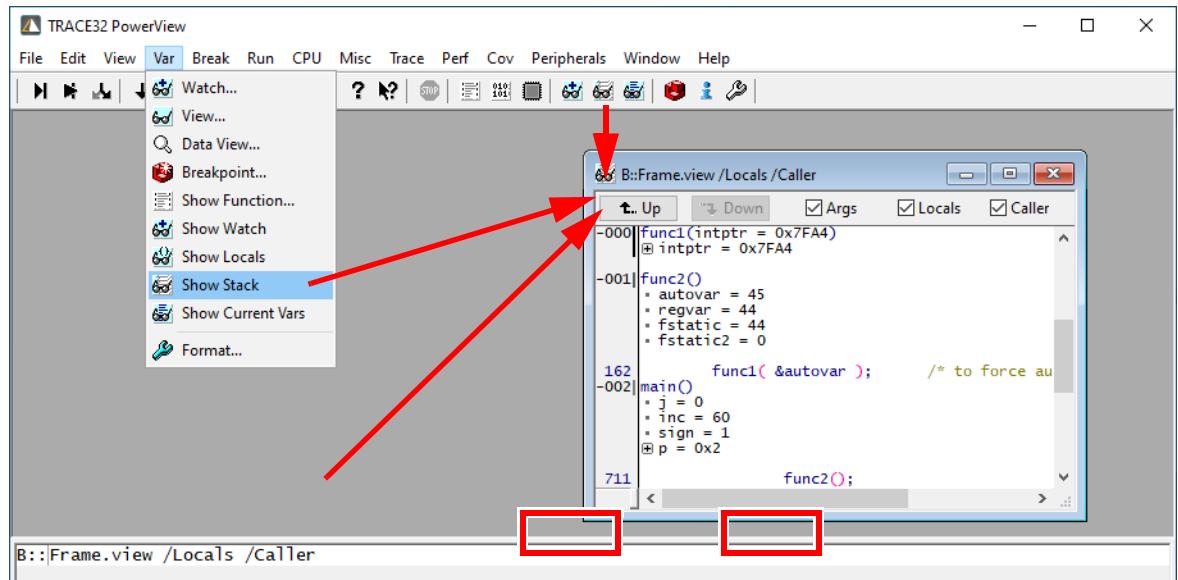
For the next example, we will assume that we have the following call hierarchy: `main()` calls `func2()` and `func2()` calls `func1()`:



Select **Show Stack** in the **Var** menu. This will open the **Frame.view** window, displaying the call hierarchy.

The **/Locals** option displays the local variables of each function, while the **/Caller** option shows a few source code lines to indicate where the function was called.

This screenshot corresponds to the calling hierarchy described above.



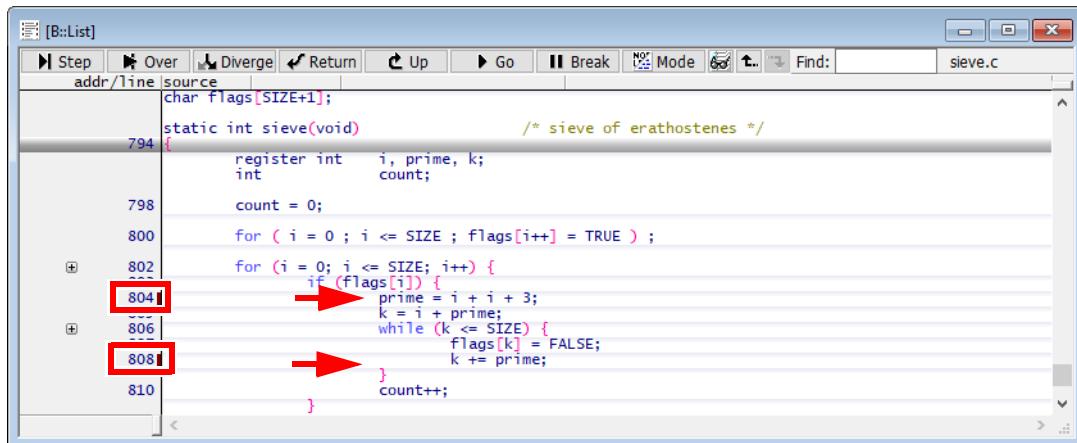
Breakpoints

Video tutorials about breakpoints in TRACE32 PowerView are available here:
support.lauterbach.com/kb/articles/using-breakpoints-in-trace32

Setting Breakpoints

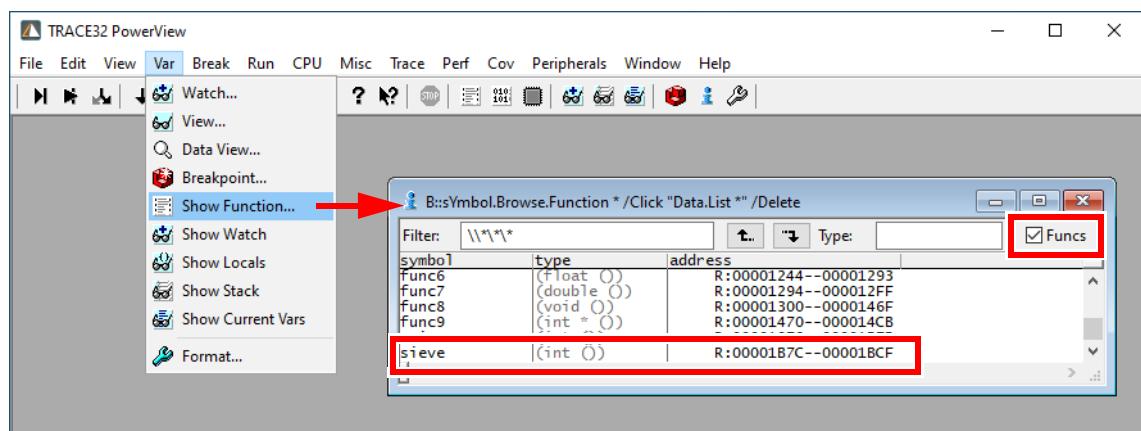
Let's set a breakpoint at the instruction `prime = i + i + 3` and the instruction `k += prime`

To set a program breakpoint, double-click a code line where you want to set the breakpoint. Ensure to click the white space in the code line, and not the code literal. All code lines with a program breakpoint are marked with a red vertical bar.



To set a breakpoint to an instruction that is not in the focus of the current source listing:

1. Choose **Var > Show Function** from the menu.
The **sYmbol.Browse.Function** window opens.

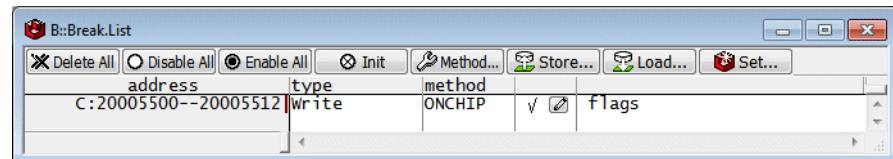
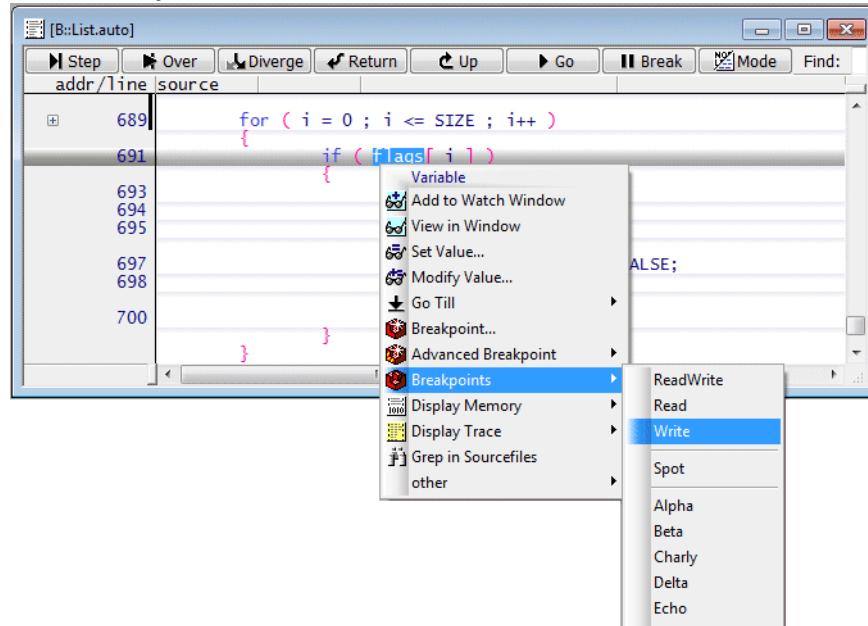


2. Select the function you are interested in, for example **sieve**.
The **List** window will open, displaying this function. This window is now fixed to the start address of the function **sieve** and does not move with the program counter cursor.

Setting Read/Write Breakpoints

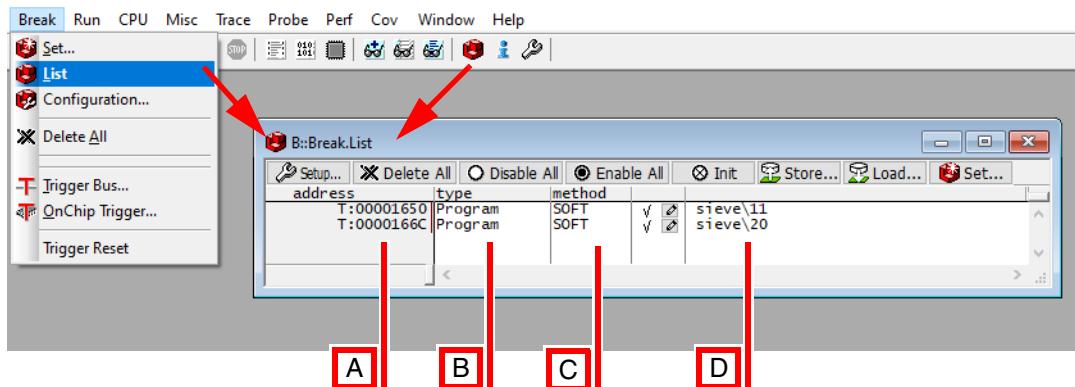
You can set a breakpoint that halts the program execution at a read or write access to a memory location, such as global variable.

To set a breakpoint on the array **flags**, for instance, right-click on the array name in the **List** window then select **Breakpoints > Write**.



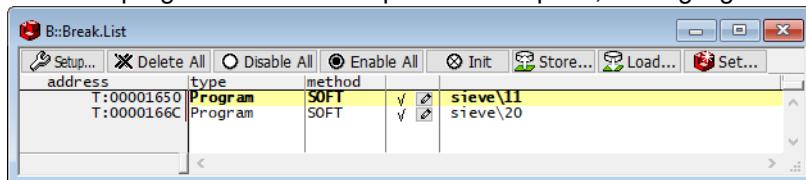
Listing all Breakpoints

1. Choose **Break > List** from the menu to list all breakpoints.
The **Break.List** window opens, providing an overview of the set breakpoints.



- A Address of the breakpoint.
- B Breakpoint type, for example Program, Read, Write
- C Breakpoint method: SOFTware, ONCHIP or DISABLED.
- D Symbolic address of the breakpoint. Example:
 - `sieve\11` means source code line 11 in function `sieve`.

2. On the toolbar, click **Go** to start the program execution.
3. When the program execution stops at a breakpoint, it is highlighted in the **Break.List** window.



Variables

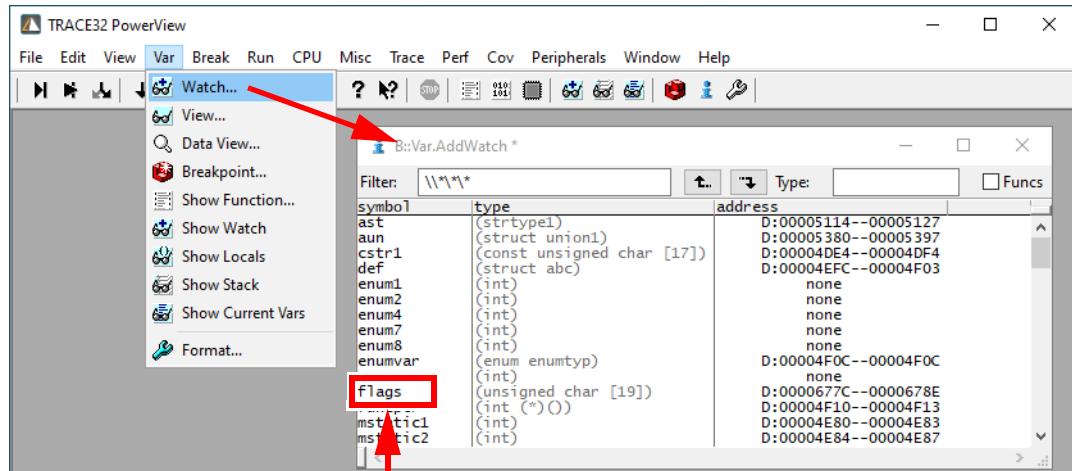
Video tutorials about variable display in TRACE32 are available here:
support.lauterbach.com/kb/articles/variable-logging-and-monitoring-in-trace32

Displaying Variables

Let's display the variables **flags**, **def**, and **ast**.

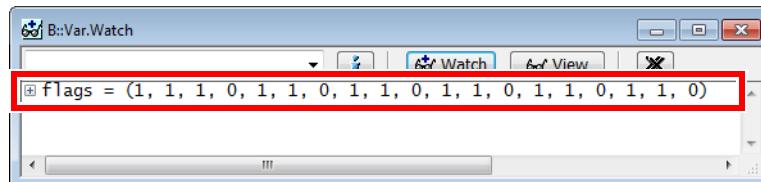
1. Choose **Var > Watch...** from the menu.

The **Var.AddWatch** window will open, displaying the variables known to the symbol database.



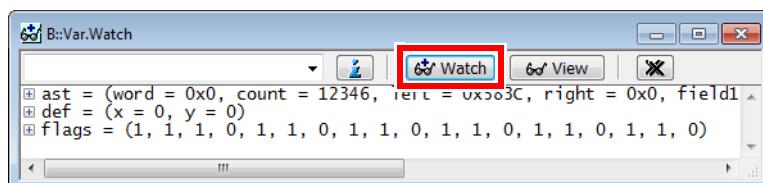
2. Double-click the variable **flags**.

The **Var.Watch** window will open, displaying the selected variable.



3. Alternative steps:

- In the **Var.Watch** window, click **Watch**, and then double-click the variables **def** and **ast** to add them to the **Var.Watch** window.



- From a **List** window, drag and drop any variable you want into the **Var.Watch** window.
- In a **List** window, right-click any variable, and then select **Add to Watch window** from the context menu.

- If you want to display a more complex structure or an array in a separate window, select the menu **Var > View**.

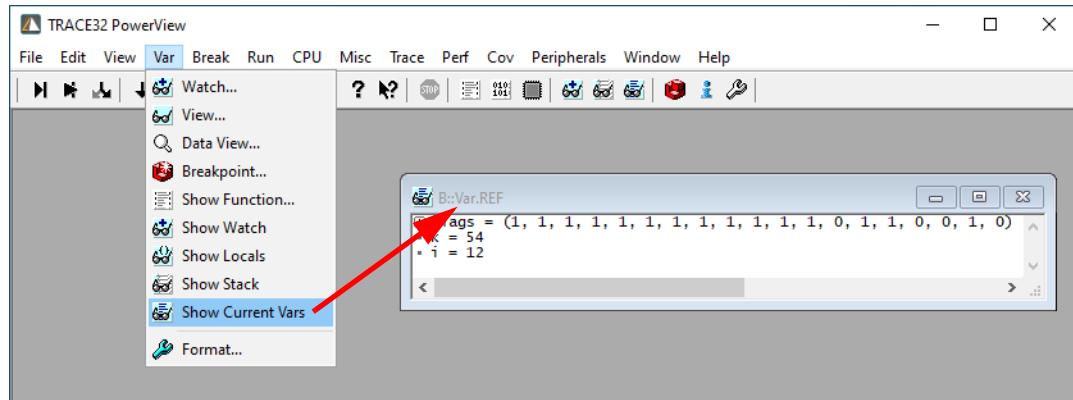
Displaying Variables of the Current Program Context

1. Set the program counter (PC) to the function **sieve()** by typing the following at the TRACE32 command line:

Register. Set PC sieve

2. Select the menu **Var > Show Current Vars.**

The **Var.REF** window opens, displaying all variables accessed by the current program context.



3. Click  **Step** on the TRACE32 PowerView toolbar to execute a few single steps. The **Var.REF** window is updated automatically.

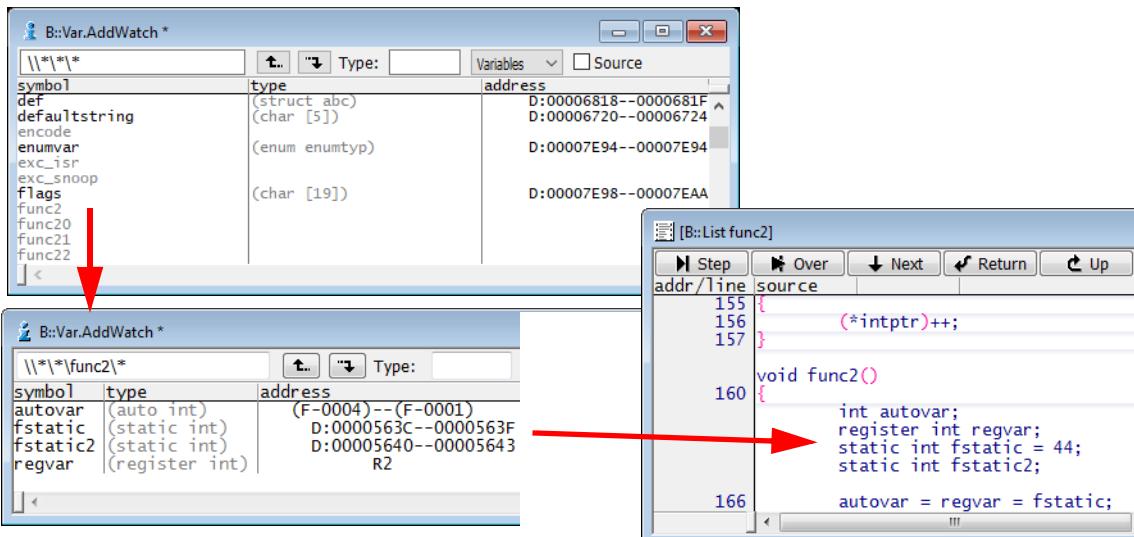
Using the Symbol Browser

The symbol browser offers an overview of the variables, functions, and modules currently stored in the symbol database.

1. Select **Var** from the menu, then choose **Watch...**

The **Var.AddWatch** window will open, allowing you to browse through the contents of the symbol database. Global variables are displayed in black and functions in gray. Double-clicking a function will display its local variables are displayed.

2. In the **Var.AddWatch** window, double-click **func2**.



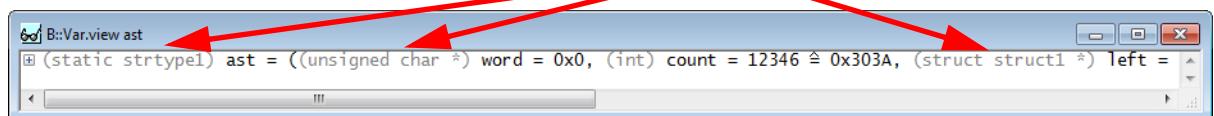
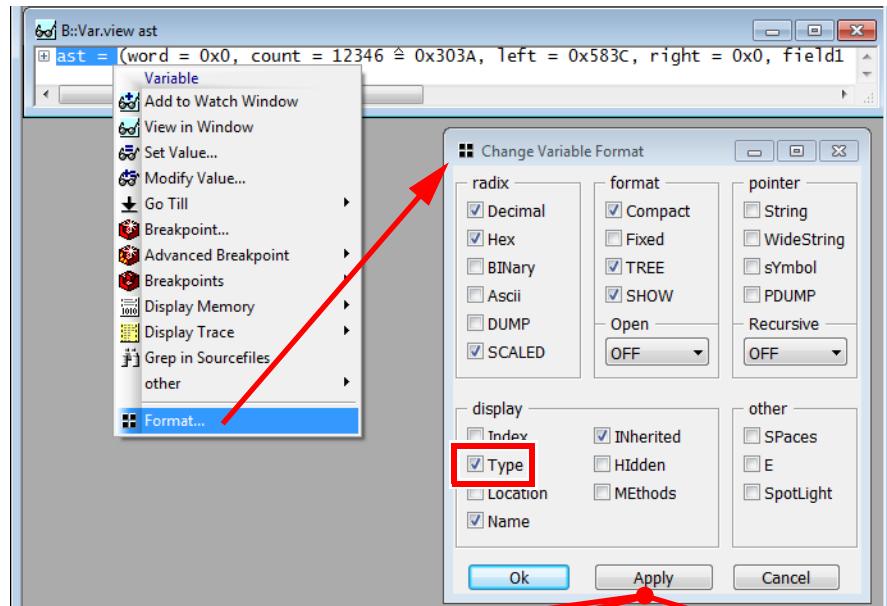
Formatting Variables

To format the display of variables with global settings:

1. Choose **Var** from the menu, then select **Format**.
2. In the **SETUP.Var** window, configure your settings. TRACE32 applies your settings to all **Var.view** windows that you open *afterwards*.

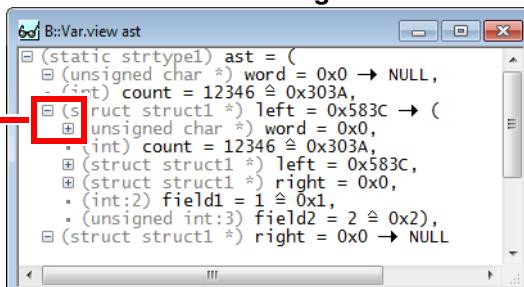
To format the display of an individual variable:

1. At the command line, type: **var.view ast** (The variable **ast** is included in this demo.)
2. In the **Var.view** window, right-click **ast**, and then click **Format**. The **Change Variable Format** dialog opens.
3. Select the **Type** check box to display the variable **ast** with the complete type information.
4. Click **Apply**. The format of **ast** in the **Var.view** window is updated immediately.



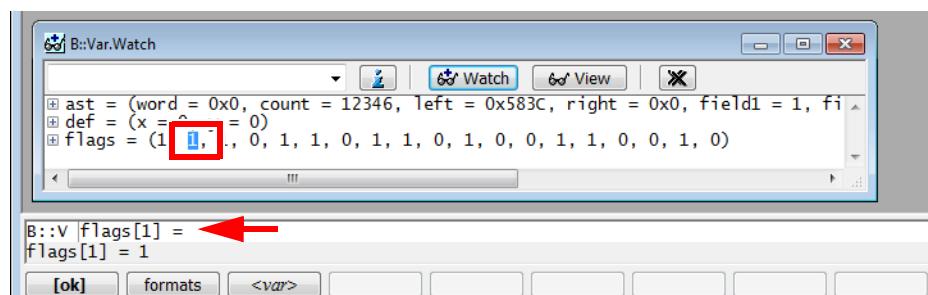
5. For more complex variable select **TREE** in the **Change Variable Format** dialog box.

Click + and - to expand and collapse the tree.



Modifying Variables

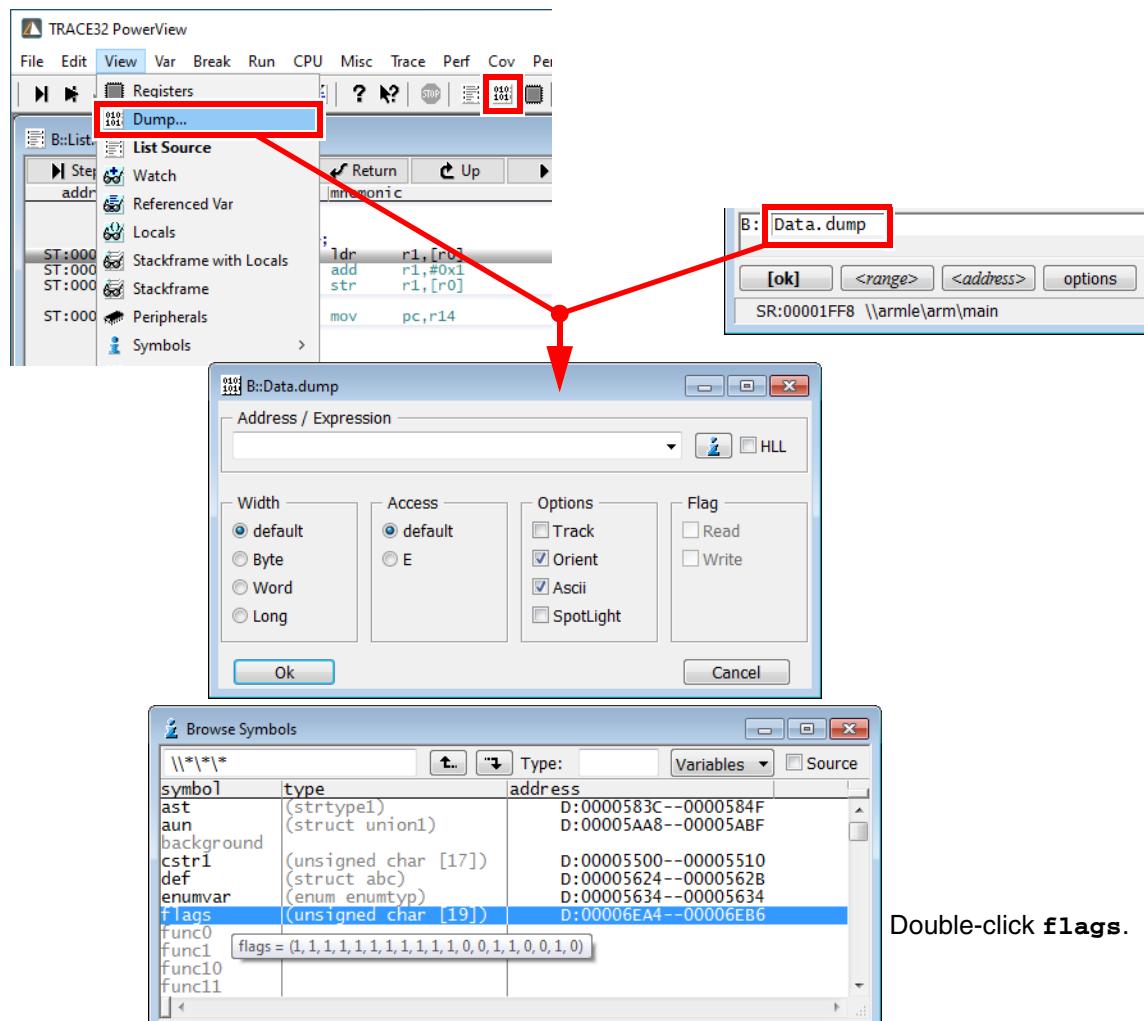
1. Double-click the variable value to modify the value. The **Var.set** command will be displayed in the command line. The short form of the command is **v** or **var**.



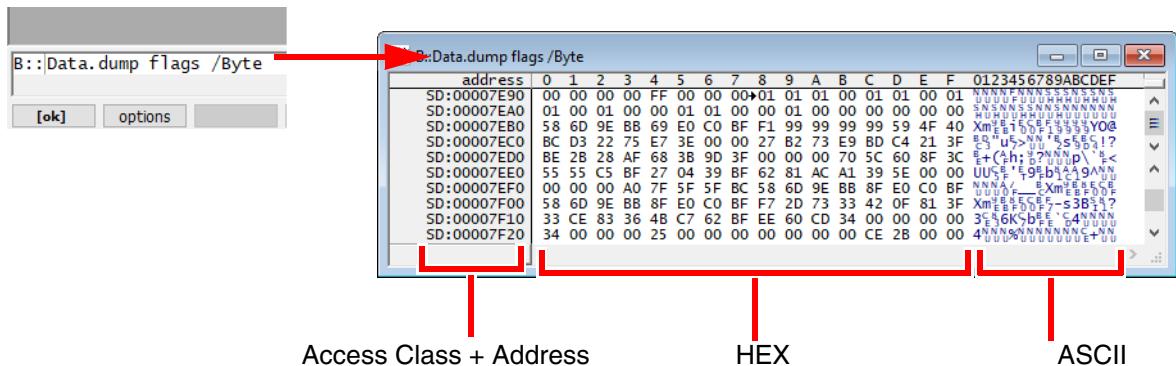
2. Enter the new value directly after the equal sign and confirm with **[ok]**.

Displaying Memory

1. To display a memory dump in a **Data.dump** window, do one of the following:
 - Choose **View** from the menu then select **Dump**,
 - Click  **Memory Dump** on the toolbar,
 - Type: **Data.dump** in the TRACE32 command line. You can also specify an address or symbol directly, e.g.: **Data.dump flags**
2. In the **Data.dump** dialog, enter the data item, e.g. **flags**
 - Alternatively click  to browse through the symbol database.
3. In the **Browse Symbols** window, double-click the symbol **flags** to select it, and then click **OK**.



In the following screenshot, the **Data.dump** window is called via the TRACE32 command line.



There are different ways to define an address range:

- $<\text{start_address}>\text{--}<\text{end_address}>$ (SD is an [access class](#))

Data.dump SD:0x5530--SD:0x554F

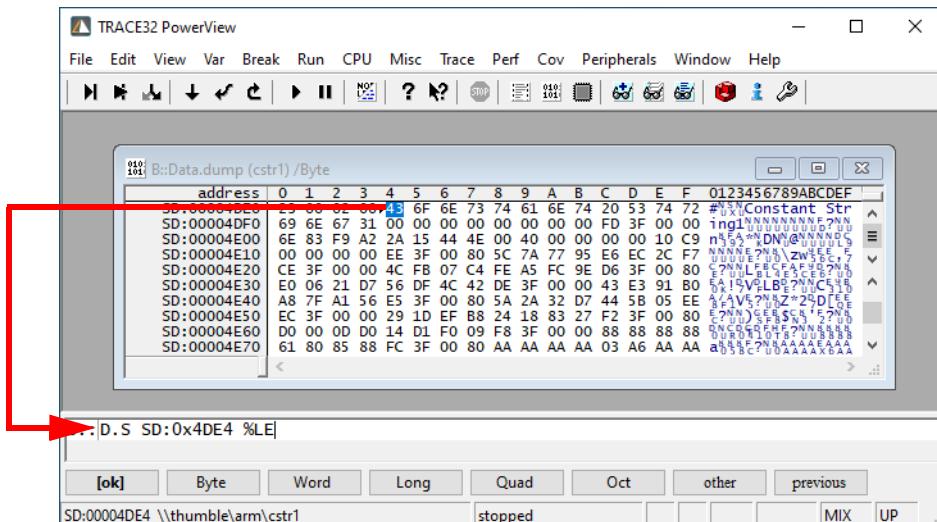
- $<start_address>++<offset>$

Data.dump cstr1++0x1f /Byte ;start at cstr1 plus the next 0x1f bytes

Modifying Memory

1. In a **Data.dump** window, double-click the value you want to modify.

A **Data.Set** command for the selected address is displayed in the command line. The short form of the command is **D.S** or **d.s**

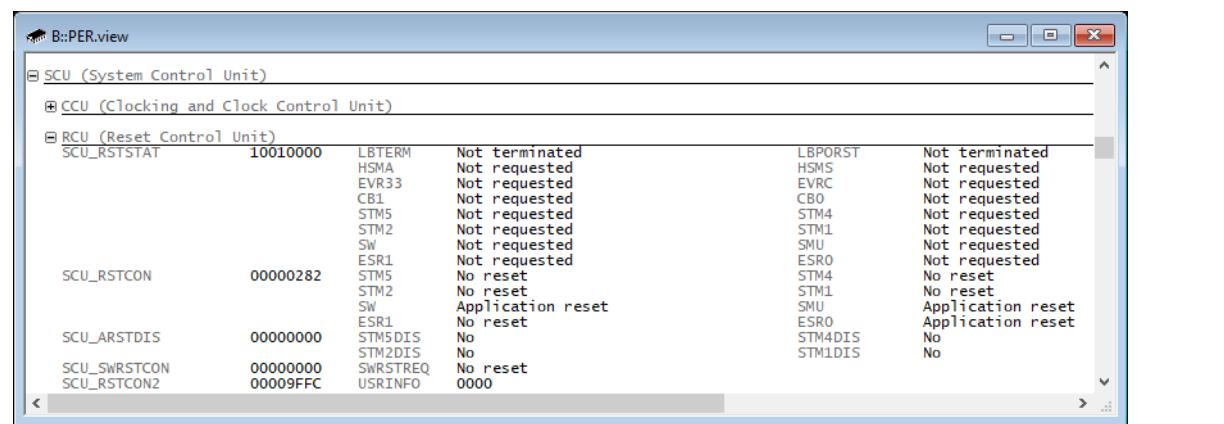


2. Enter the new value directly after **%LE**, and then confirm with **[ok]**. (**%LE** stands for Little Endian).

TRACE32 supports a freely configurable window for displaying and manipulating configuration registers and on-chip peripheral registers at a logical level. Predefined peripheral files are available for most standard processors/chips.

You can open the peripheral register view in the TRACE32 by selection the **CPU** menu, then **Peripherals**, or by using the command **PER.view** in the TRACE32 command line.

PER.view Display peripheral registers



The screenshot shows the TRACE32 PER.view window with the title 'B::PER.view'. The window displays the System Control Unit (SCU) peripheral registers. The tree view on the left shows nodes for SCU, CCU, and RCU. The RCU node is expanded, showing sub-registers SCU_RSTSTAT, SCU_RSTCON, SCU_ARSTDIS, SCU_SWRSTCON, and SCU_SWRSTCON2. The SCU_RSTSTAT register is selected, showing its bit fields: LBTERM (Not terminated), HSMA (Not requested), EVR33 (Not requested), CB1 (Not requested), STM5 (Not requested), STM2 (Not requested), SW (Not requested), ESR1 (Not requested), STM5 (No reset), STM2 (No reset), SW (Application reset), and ESR1 (No reset). The SCU_RSTCON register shows bit fields STM5DIS (No), STM2DIS (No), and SWRSTREQ (No reset). The SCU_SWRSTCON and SCU_SWRSTCON2 registers show bit field USRINFO with value 0000. The right side of the window shows the bit names and their current state for each register.

Register	Bit	Value	State
SCU_RSTSTAT	LBTERM	Not terminated	Not terminated
	HSMA	Not requested	Not requested
	EVR33	Not requested	Not requested
	CB1	Not requested	Not requested
	STM5	Not requested	Not requested
	STM2	Not requested	Not requested
	SW	Not requested	Not requested
	ESR1	Not requested	Not requested
SCU_RSTCON	STM5	No reset	STM4
	STM2	No reset	STM1
	SW	Application reset	SMU
	ESR1	No reset	ESRO
SCU_ARSTDIS	STM5DIS	No	STM4DIS
	STM2DIS	No	STM1DIS
SCU_SWRSTCON	SWRSTREQ	No reset	
SCU_SWRSTCON2	USRINFO	0000	