

# Debugger Tutorial

Release 02.2025


# Debugger Tutorial

---

TRACE32 Online Help

TRACE32 Directory

TRACE32 Index

TRACE32 Debugger Getting Started .....	
<b>Debugger Tutorial</b> .....	<b>1</b>
<b>History</b> .....	<b>4</b>
<b>About the Tutorial</b> .....	<b>4</b>
<b>Step 1: Connect the TRACE32 Hardware</b> .....	<b>5</b>
<b>Step 2: Start TRACE32 PowerView Software</b> .....	<b>5</b>
<b>Step 3: Explore the TRACE32 PowerView User Interface</b> .....	<b>8</b>
<b>Step 4: Set Up Your Debug Environment</b> .....	<b>10</b>
Start-Up Scripts	10
Typical Setup Procedure	11
Selecting the CPU	11
Establish Debug Communication	12
<b>Step 5: Loading Your Application</b> .....	<b>13</b>
Loading an Application Located in RAM	13
Loading an Application Located in Flash	13
<b>Step 6: Debugging the Program</b> .....	<b>15</b>
Basic Debug Commands	16
Displaying the Stack Frame	17
<b>Breakpoints</b> .....	<b>18</b>
Setting Program Breakpoints	18
Setting Read/Write Breakpoints	20
Listing all Breakpoints	21
<b>Variables</b> .....	<b>22</b>
Displaying Variables	22
Displaying Variables Referenced by the Current Source Line	23
Using the Symbol Browser	24
Formatting Variables	25
Modifying Variables	27
<b>Memory</b> .....	<b>28</b>
Displaying Memory	28
Modifying Memory	30
Peripheral View	31

<b>Store Window Configuration</b> .....	<b>32</b>
<b>Getting Online Help</b> .....	<b>33</b>
Contacting the Lauterbach Support	35

## History

---

05-Jun-2024 Revised manual.

## About the Tutorial

---

Welcome to this tutorial, designed to help you quickly get up and running with the TRACE32 hardware-assisted debugger. We'll walk you through the essential steps to configure and start a debug session, using the TRACE32 PowerView interface. By the end of this guide, you'll be familiar with the core features of TRACE32, empowering you to confidently navigate your debugging tasks.

This guide is perfect for beginners, especially those who are new to TRACE32 tools. If you're looking for a more in-depth exploration of all the debugging capabilities, check out our detailed resource, [“Training Basic SMP Debugging”](#) (training\_debugger\_smp.pdf).

To make the most of this tutorial, having a basic understanding of software debugging and C programming will be beneficial. This knowledge will help you follow along with the example code. Additionally, familiarity with your target processor in use is important for getting your debug environment up and running smoothly.

Before diving in, please ensure that the TRACE32 debugger software is installed on your system. You can install it from the DVD included with your tools or download the installation package from the [Lauterbach website](#). For step-by-step installation instructions, refer to [“Software Installation”](#) in TRACE32 Installation Guide, page 20 (installation.pdf).

# Step 1: Connect the TRACE32 Hardware

---

To begin this tutorial, you'll need a functioning target platform, such as a development board. Follow these steps to ensure a smooth setup:

## 1. Disconnect Existing Connections:

If the Debug Cable, CombiProbe, or MicroTrace is already connected to the target, disconnect it while the target power is off.

## 2. Connect the TRACE32 Hardware:

Follow the connection instructions detailed in [“Tool Configuration”](#) in TRACE32 Installation Guide, page 9 (installation.pdf).

## 3. Power On:

Connect the POWER DEBUG module or MicroTrace to the host computer over USB or Ethernet then power it on.

# Step 2: Start TRACE32 PowerView Software

---

Starting the TRACE32 PowerView software varies depending on your host operating system and the target processor architecture you selected during installation. Here's how to proceed:

## For Windows Users:

After installation, you can launch the TRACE32 PowerView software directly from the Windows Start Menu. Simply navigate to the menu entry created during installation and click to start the software.

## For Linux Users:

On Linux, the software must be started from the command line. The name of the executable depends on the target processor architecture you selected during installation. For example:

If you selected an Arm processor, the executable will be:

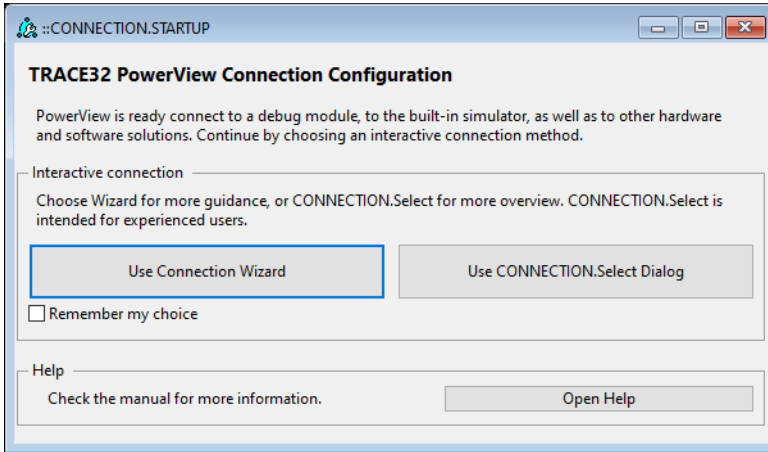
```
/opt/t32/bin/pc_linux64/t32marm
```

If you selected a TriCore processor, the executable will be:

```
/opt/t32/bin/pc_linux64/t32mtc
```

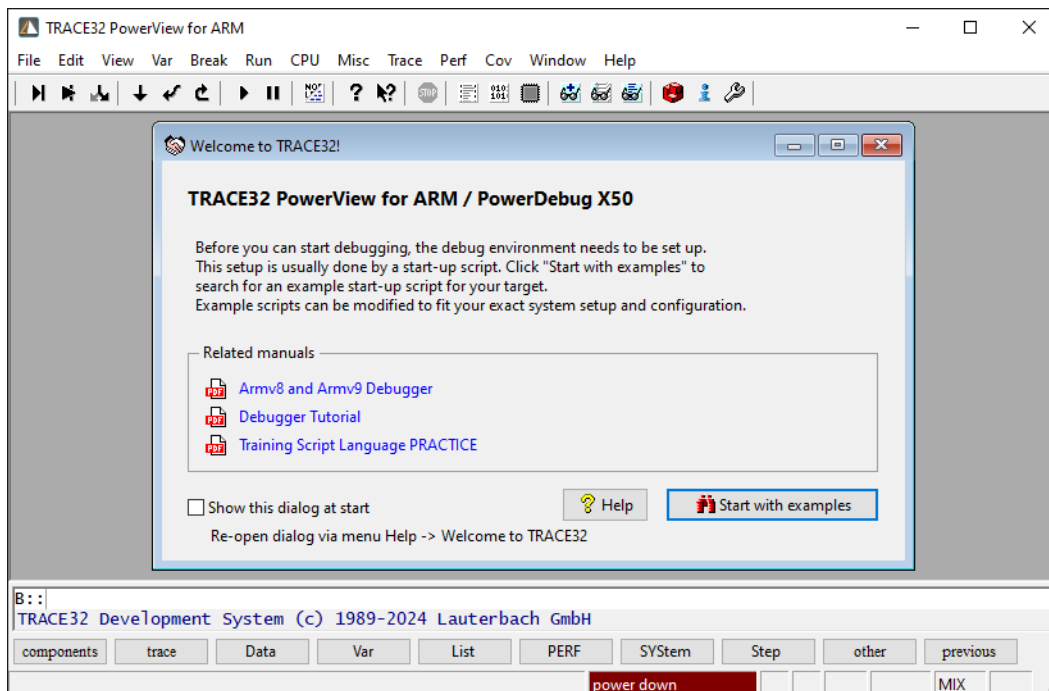
These examples assume the software is installed under `/opt/t32`. Adjust the path if your installation is in a different location.

If you've started TRACE32 PowerView in **“Interactive Connection Mode”**, the TRACE32 PowerView **Connection Configuration Wizard** will automatically open. This wizard will guide you step-by-step through the process of connecting to the POWER DEBUG module:

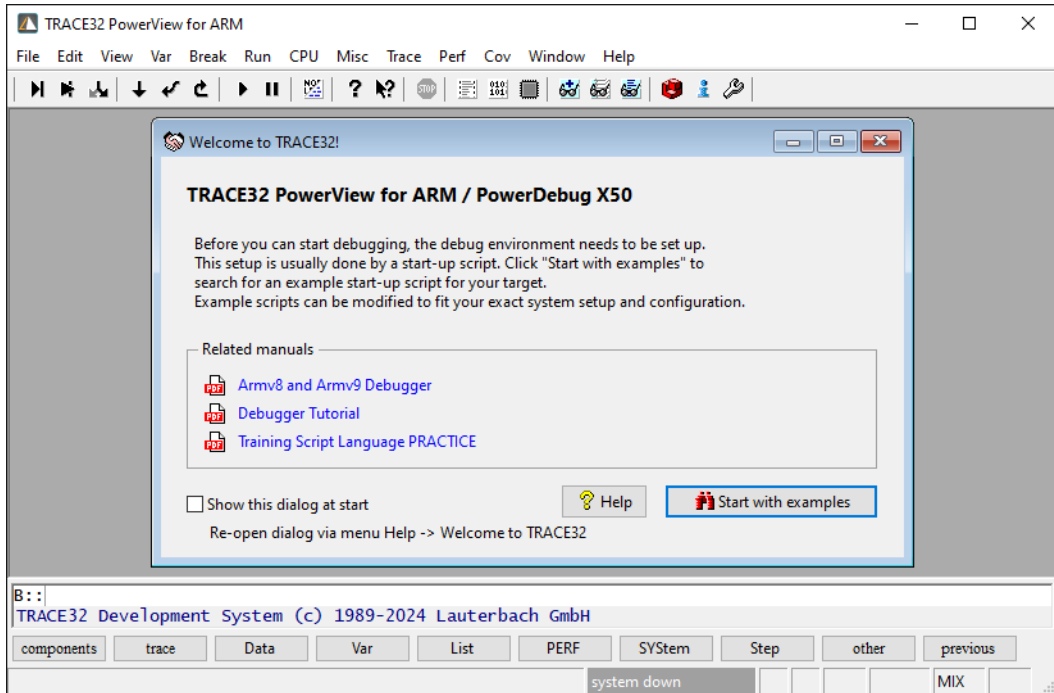


Otherwise, if you start TRACE32 PowerView without selecting **Interactive Connection Mode**, the software will launch using the hardware configuration specified during the **“Classic Mode”** installation.

Once the connection to the TRACE32 debugger hardware is successfully established, the TRACE32 PowerView interface will launch. By default, a **“Welcome to TRACE32!”** dialog will appear. This dialog provides important information, including the target architecture, the exact name of your POWER DEBUG module or MicroTrace, and links to essential manuals. A red **“power down”** status in the status bar indicates that the TRACE32 debugger is not connected to the target hardware, as outlined in the previous chapter **“Step 1: Connect the TRACE32 Hardware”**, page 5.



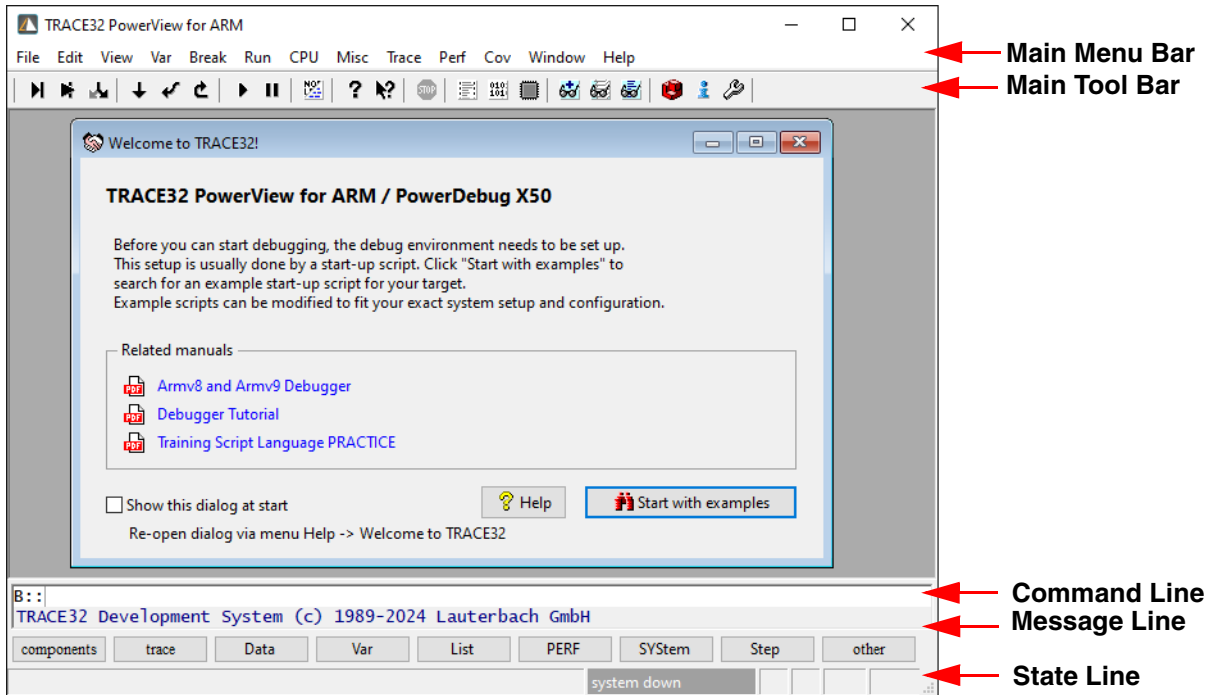
Connect your Debug Cable, CombiProbe, or MicroTrace to the target platform, then power on the target. The status line should change to a grey “**system down**” status:



# Step 3: Explore the TRACE32 PowerView User Interface

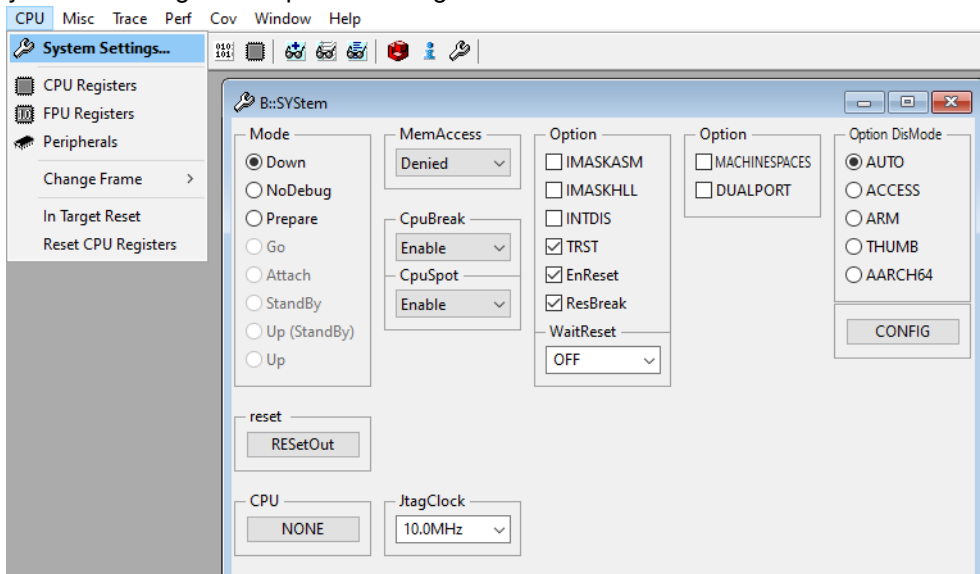
Now that we've started the TRACE32 PowerView user interface, let's take a closer look at its main components. This chapter will help you become familiar with the key elements of the interface, so you can navigate and utilize TRACE32 PowerView effectively.

The following screenshot presents the main components of TRACE32 PowerView:



The TRACE32 PowerView user interface consists of several key components designed to help you efficiently navigate and use the tool:

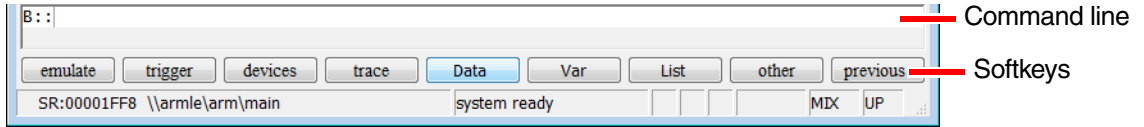
- **Main Menu Bar:** This bar provides access to all important commands for each functional unit of TRACE32. For example, selecting **CPU > System Settings...** opens the **SYSTEM** window, where you can manage core-specific settings.



- **Main Tool Bar:** This toolbar features buttons for frequently used TRACE32 commands. For instance, you can access the **SYSTEM** window using the corresponding icon on this toolbar.



- In addition to these, TRACE32 PowerView includes a **Command Line** where you can execute TRACE32 commands directly. Any action you can perform using the graphical user interface can also be accomplished through command line inputs. The command line is complemented by **Softkeys**, which guide you through command entry.



**NOTE:**

In this document, we'll focus on using the **Main Menu Bar** and **Main Tool Bar** for performing actions in TRACE32 PowerView. For comprehensive details about using the **Command Line**, please refer to "[PowerView Command Reference](#)" (ide\_ref.pdf).

- **Message Line:** This area displays error notifications as well as general messages, keeping you informed about the status of your operations.
- **State Line:** This line provides various details about the current states of both the debugger and the target, offering a quick overview of critical information.

## Step 4: Set Up Your Debug Environment

Now that we're ready to dive in, let's get your debug connection up and running with the target processor. First, we'll guide you on how to search for a ready-to-use start-up script, which simplifies the setup process. Then, we'll walk you through the typical start-up procedure step-by-step, as generally included in these scripts.

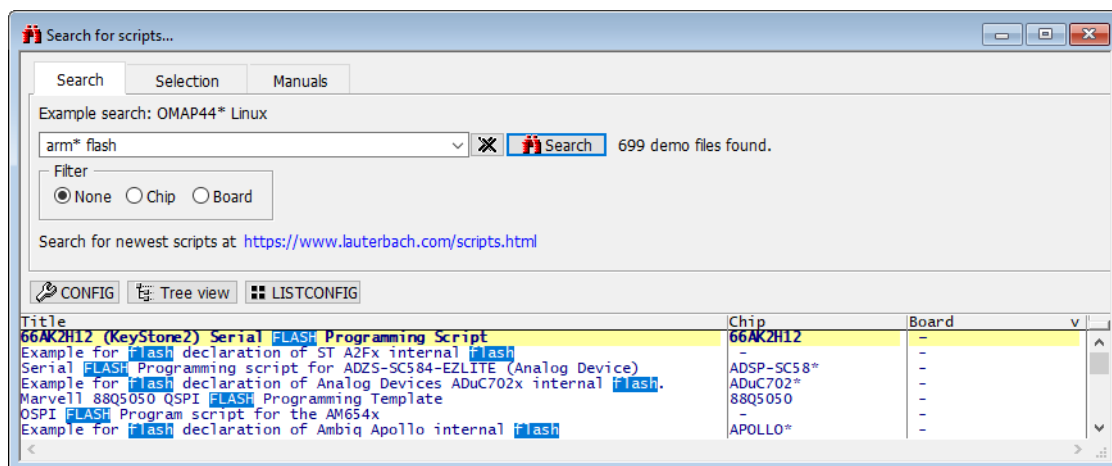
**NOTE:** For detailed instructions on CPU-specific settings, check out the [Processor Architecture Manual](#). You can find this manual by selecting from the TRACE32 PowerView menu **Help** then **Processor Architecture Manual**.

### Start-Up Scripts

To make things easier, TRACE32 provides a collection of ready-to-use PRACTICE scripts located in the demo directory within the TRACE32 installation folder. These scripts are designed to help you quickly set up a debug connection with a wide range of commonly used target chips and boards.

**NOTE:** *PRACTICE* is a scripting language integrated into TRACE32 tools. It allows you to automate various tasks, including setting up proper start-up sequences for development tools, programming FLASH memory, customizing the user interface, saving and reactivating specific TRACE32 settings, and running automated tests.

To find the right script for your target platform, use the “**Search scripts...**” feature. You can access this view by clicking the “**Start with examples**” button in the “**Welcome to TRACE32!**” dialog, or by navigating to **File > Search for Script** from the TRACE32 PowerView menu.



# Typical Setup Procedure

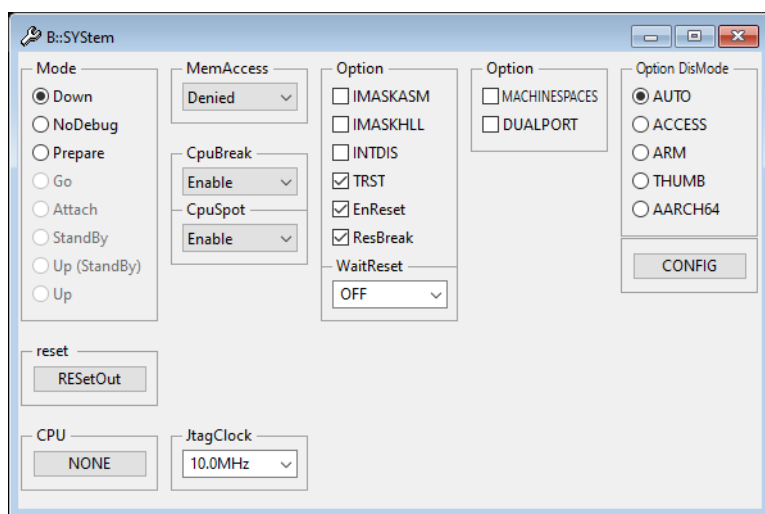
In this chapter, we'll walk you through a typical setup procedure for the debugger. For simplicity, we'll use a single-core system as our example.

## Selecting the CPU

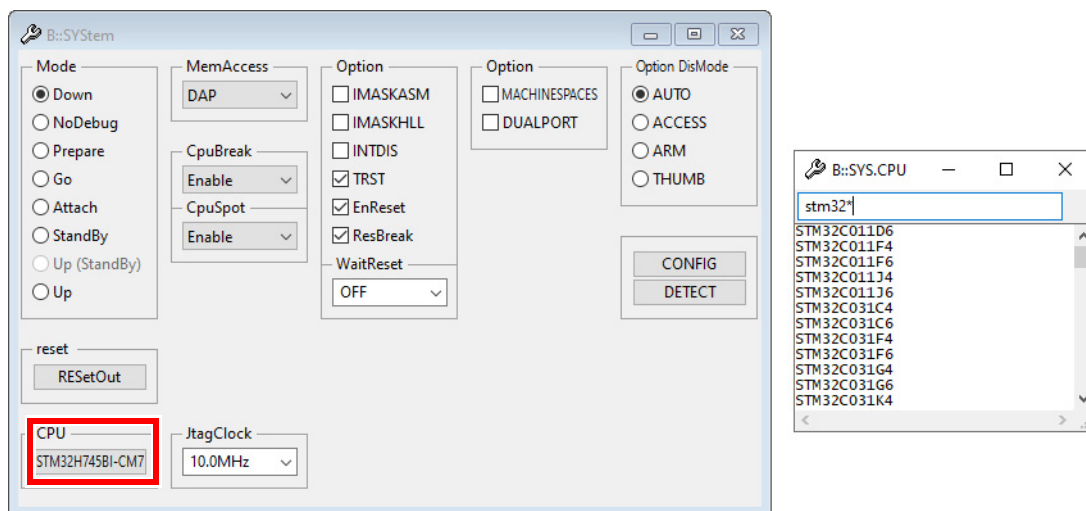
The first step in configuring your debugger is selecting the correct CPU core. This is crucial because it ensures that all subsequent settings are tailored to the specific processor you're working with.

To manage core-specific settings, you can use the **SYSTEM** window. You can access this window by navigating through the PowerView menu: click on **CPU**, then select **SYSTEM Settings**.

Below is an example of the **SYSTEM** window in TRACE32 PowerView for Arm processors. Keep in mind that the layout may differ slightly depending on the target processor architecture you are using, but the general procedure remains consistent.

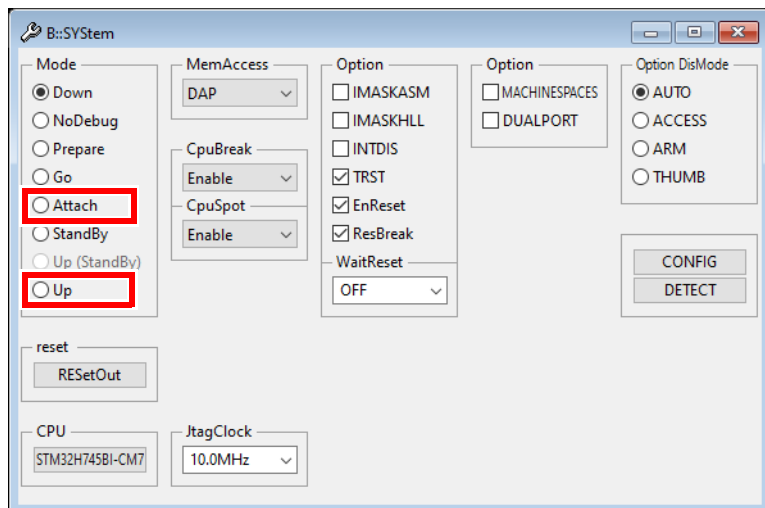


Now, let's select the CPU core you want to debug. In the **SYSTEM** window, look for the button labeled **CPU**. Clicking this button opens the **SYSTEM.CPU** window, where you can search for and select your target **CPU**.



## Establish Debug Communication

The most common method to establish the debug communication is to select the **Up** radio button under **Mode** from the **SYSTEM** window.



When you choose **Up**, the following actions typically occur by default:

- The core is reset.
- The debugger initializes communication with the core.
- If supported by your core, the debugger will stop the core at the reset vector.

Another useful method to establish the communication between the debugger and the core is **Attach**. **Attach** allows the debugger to connect to an already running core. The **Break** button from the **Main Tool Bar** can then be used to stop the running core.

If you get an error after selecting **Up** or **Attach**, refer to the [Processor Architecture Manual](#).

### NOTE:

Some cores require additional settings, for example related to reset handling, before communication can be established. Most relevant settings can be configured under **Option** from the **SYSTEM** window. The available settings depend on the target processor architecture. A detailed description can be found in the [Processor Architecture Manual](#).

In case you need assistance, please refer to the chapter "[Contacting the Lauterbach Support](#)", page 35.

## Step 5: Loading Your Application

---

Setting up your debugging environment involves loading both the code you want to debug and the associated debug symbols. The process varies slightly depending on whether your application is designed to run from RAM or flash memory.

### Loading an Application Located in RAM

---

If your application is meant to run from RAM, the process is straightforward:

1. Navigate to the **File** menu in TRACE32 PowerView.
2. Select **Load File**.
3. Choose the application you want to load.

This action will load your application directly into the target processor's memory. Additionally, it will automatically load the corresponding debug symbols into TRACE32 PowerView, ensuring that you can debug your application with full visibility into the code and variables.

**NOTE:**

Make sure that the program execution is stopped before loading your application. If you see a green “**running**” indication in the TRACE32 status line, stop the program execution using the **Break** button from the **Main Tool Bar** or by selecting the menu **Run > Break**.

### Loading an Application Located in Flash

---

TRACE32 supports programming on-chip and off-chip NOR flash memories, as well as serial flash memories, such as NAND, SPI and eMMC. However, For simplicity, this tutorial will focus on on-chip flash programming.

Ready-to-run scripts for most on-chip flash memories can be found in the TRACE32 installation under `~/demo/<architecture>/flash/<cpu>.cmm`

**Examples:**

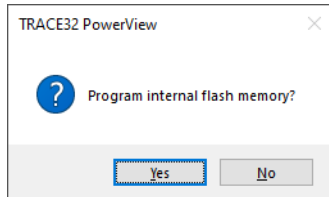
```
~/demo/arm/flash/stm32h7.cmm
```

```
~/demo/tricore/flash/tc37x.cmm
```

To program your application to the on-chip flash memory of your processor/chip, follow these steps:

1. Call the flash programming script appropriate to your processor/chip by choosing the menu **File > Run Script** then selecting the flash programming script for you chip from the `demo/<arch>/flash` directory.

2. The script will perform all necessary preparations then displays a pop-up asking the user to confirm proceeding with the flash programming.



3. A file dialog is then opened, allowing you to browse the target application you want to program.

A video tutorial on programming the processor's internal flash memory using TRACE32 is available here:

[support.lauterbach.com/kb/articles/flash-programming](https://support.lauterbach.com/kb/articles/flash-programming)

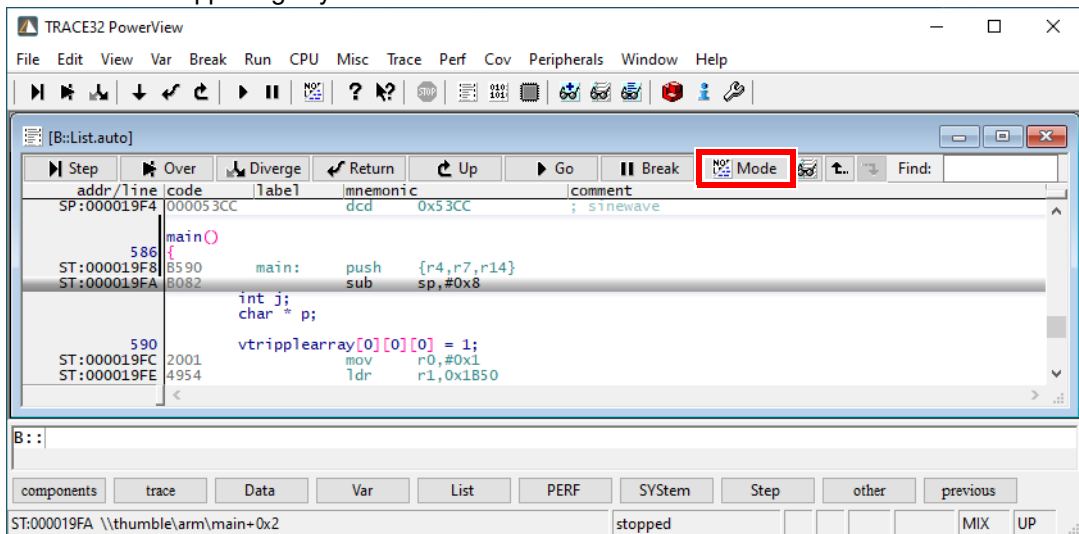
# Step 6: Debugging the Program

Once your target application and the corresponding debug symbols are loaded, it's time to start debugging. The first step in this process is to display the source code you'll be working with.

To do this:

- Go to the **View** menu in TRACE32 PowerView.
- Select **List Source** to open the **List** window.

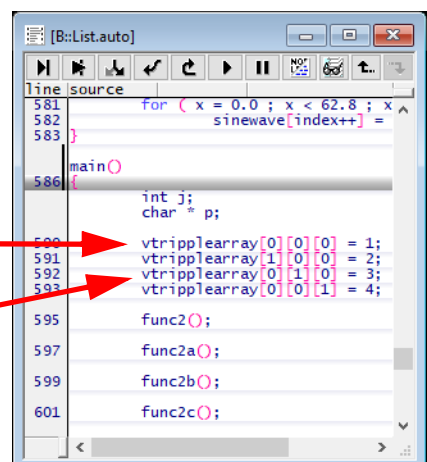
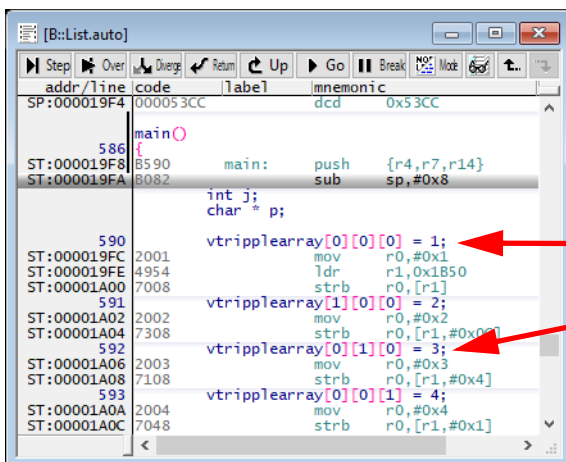
By default, the source code is displayed in **Mixed Mode**, which shows both the high-level language source code (HLL) and the corresponding assembly mnemonics side by side. This view provides a comprehensive look at what's happening in your code at both levels.



If you prefer to focus solely on the high-level source code, you can easily switch modes. Just click the **Mode** button in the **List** window, and it will toggle the view to display only the high-level source code.

Debug mode MIX

Debug mode HLL




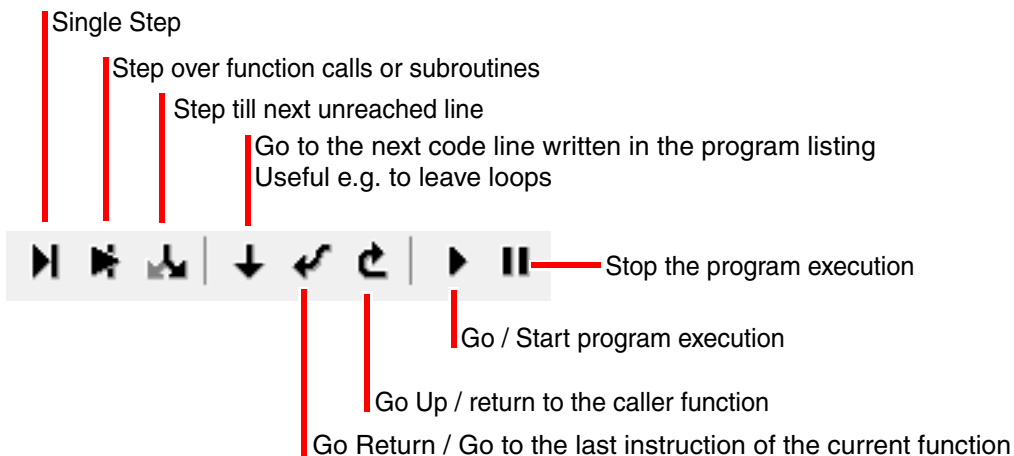
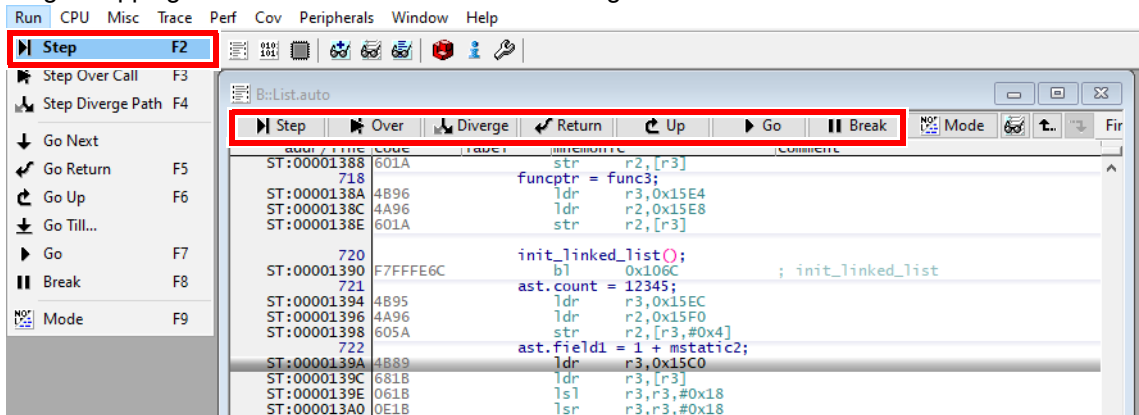
**NOTE:**

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


## Basic Debug Commands



The basic debug commands are accessible from, the **Run** menu, the **Main Tool Bar** or the toolbar of the **List** window.

Single stepping  is one of these fundamental debug commands.



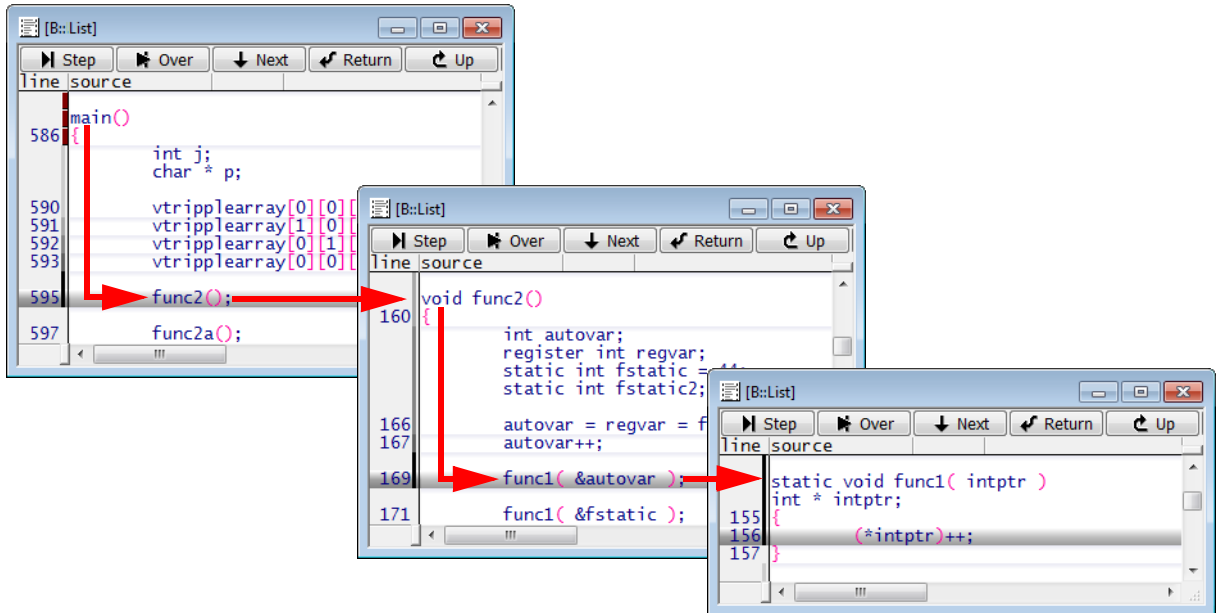
The behavior of the **Step** buttons depends on the selected debug mode, **Mixed Mode** or **HLL**. We can showcase this based on the Single Step button as an example:

- Toggle to **HLL** mode using the **Mode** button
- Click  **Step**.  
In HLL mode, this action moves the program execution to the next source code line.

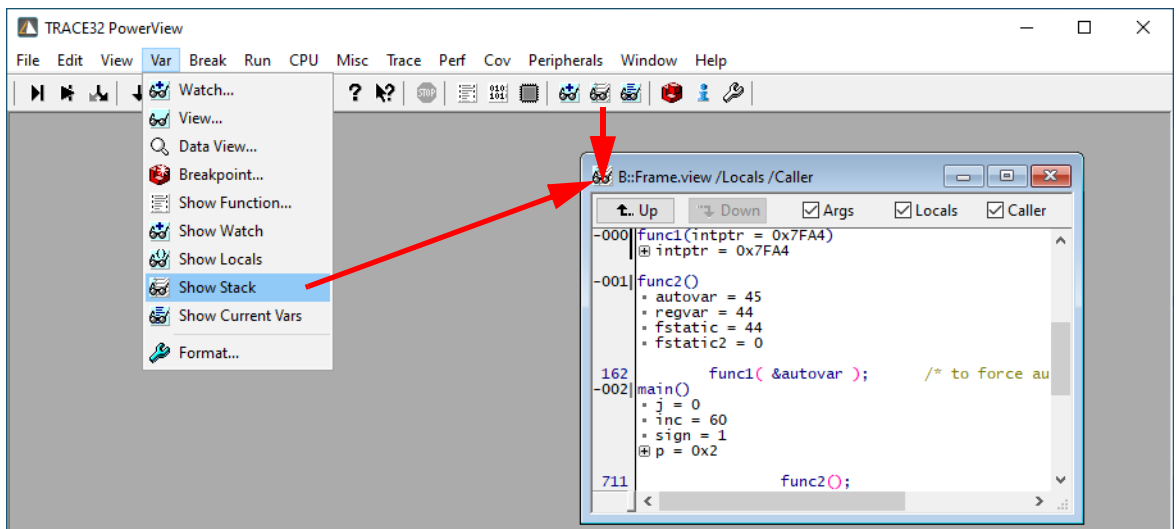
- Click  **Mode** again to toggle the debug mode to **MIX**.
- Click  **Step**.  
This time, the step executes one assembler instruction.

## 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 window also shows per default the local variables of each function as well as the high level language block, from which the function was called. The following screenshot corresponds to the calling hierarchy described above.



# Breakpoints

We can mainly distinguish between two types of breakpoints:

- **Program Breakpoints:** These stop program execution at a specific line of code.
- **Read/Write Breakpoints:** These halt execution when a memory location, such as a variable, is read from or written to by the target processor.

This tutorial will cover the basics of setting and listing breakpoints in TRACE32 PowerView. For more advanced breakpoint features, refer to the [“Training Basic SMP Debugging”](#) (training\_debugger\_smp.pdf).

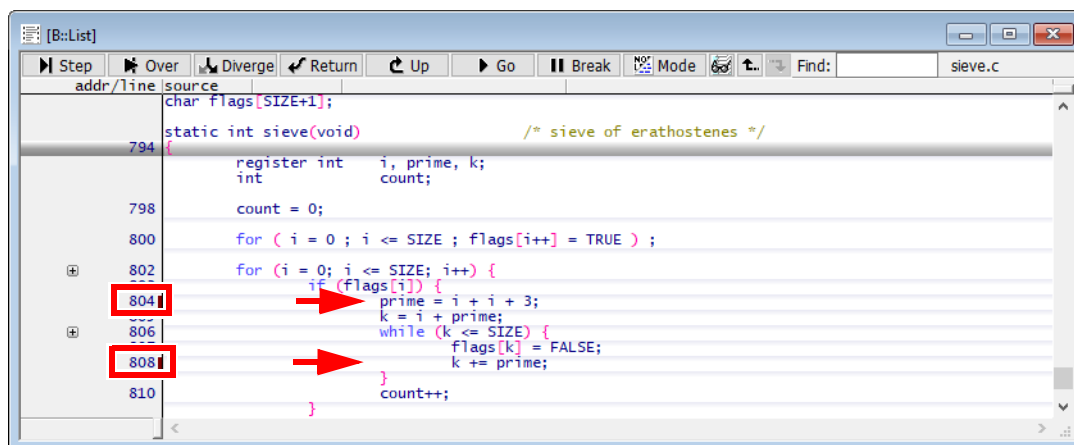
Additionally, video tutorials on using breakpoints in TRACE32 PowerView are available here: [support.lauterbach.com/kb/articles/using-breakpoints-in-trace32](http://support.lauterbach.com/kb/articles/using-breakpoints-in-trace32)

## Setting Program Breakpoints

To set a program breakpoint in the source code:

- Navigate in the **List** window to the line where you want to set a breakpoint.
- Double-click on the white space to the left of the code (not on the code itself). A red vertical bar will appear next to the line, indicating that a breakpoint has been successfully set.

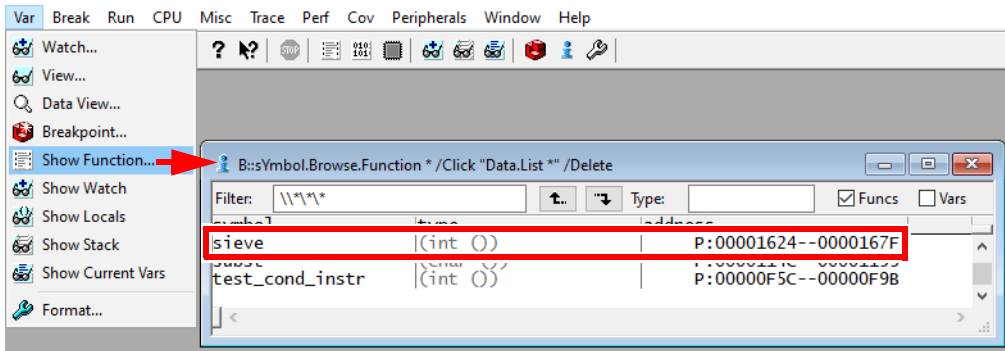
The screenshot below shows for instance two program breakpoints: one at the instructions `prime = i + i + 3` located at line 804, and another at `k += prime` on line 808.



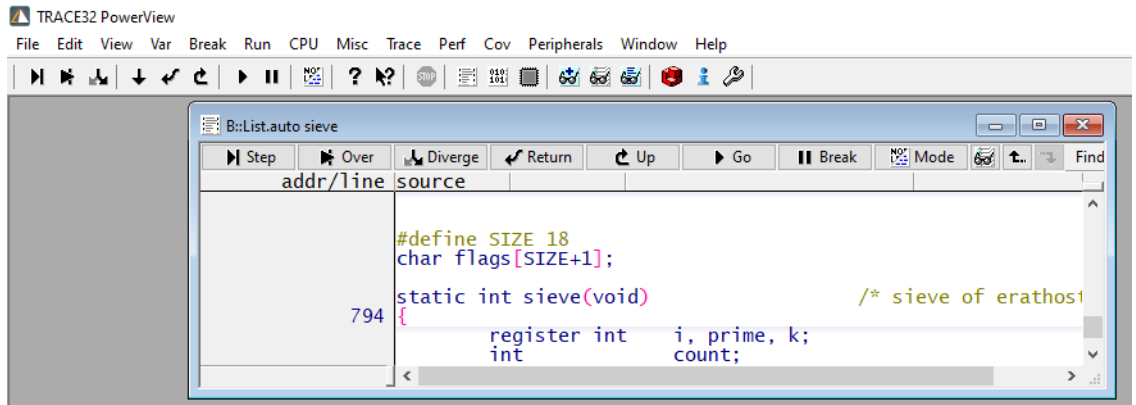
To remove a breakpoint, simply double-click the same line again.

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

- Choose **Var > Show Function** from the menu. The **sYMBOL.Browse.Function** window opens.



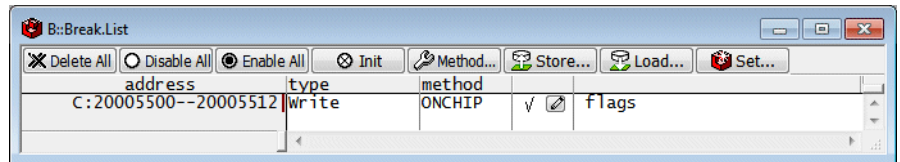
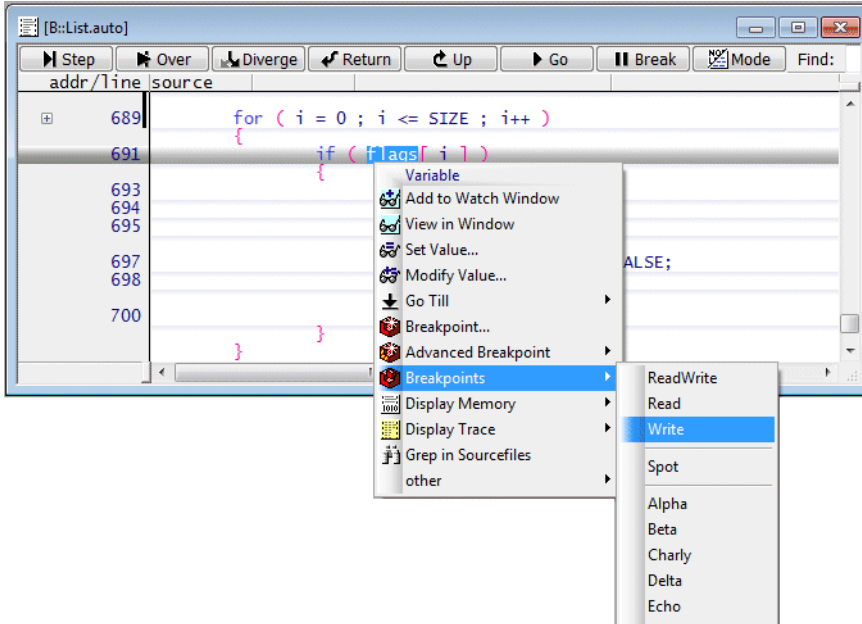
- Select the function you are interested in, for example **sieve**. The **List** window will open, displaying this function. This window is now fixed to the start address of the function **sieve** and does not move with the program counter cursor.



# Setting Read/Write Breakpoints

You can additionally set a breakpoint to halt program execution whenever there is a read or write access to a specific memory location, such as a global variable.

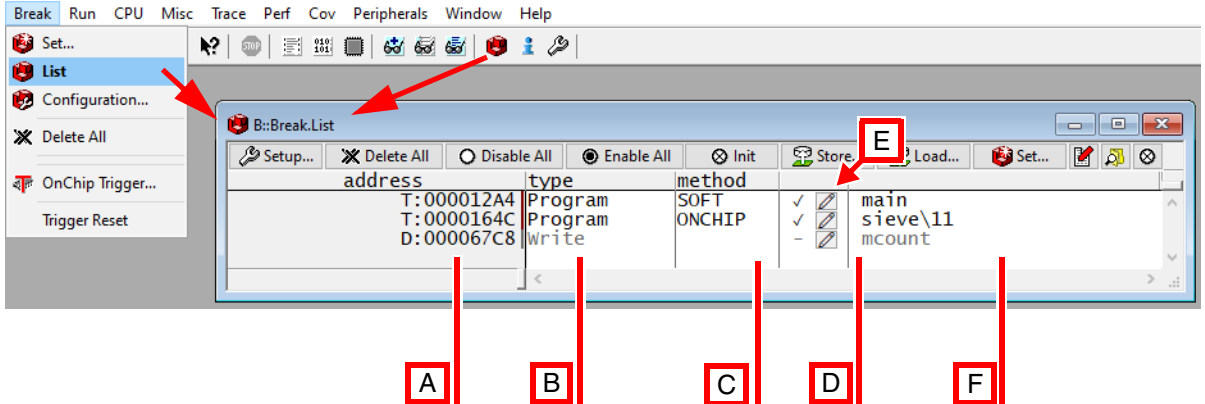
For example, to set a breakpoint on the array `flags`, right-click on the array name in the **List** window and then select **Breakpoints > Write**.



A **Write** breakpoint will be applied to the entire address range of the array. This means that any write access by the process to any element of the array will trigger the breakpoint and halt program execution.

# Listing all Breakpoints

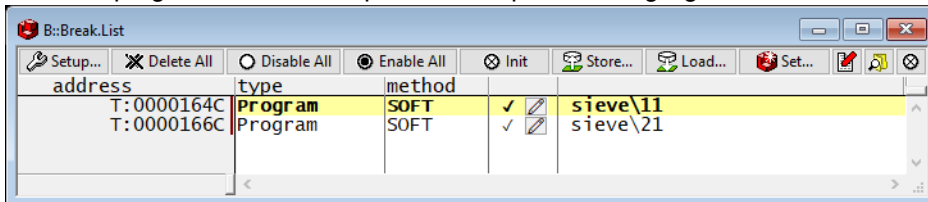
To view a list of all breakpoints set during your debug session, open the **Break.List** window by selecting **Break** then **List** from the menu.



The **Break.List** window shows the following information, from left to right:

- The address of the breakpoint [A]
- The breakpoint type [B], for example “Program”, “Read” or “Write”
- The breakpoint method [C], which can be **SOFT** for software breakpoints, or **ONCHIP** for onchip breakpoints.
- Breakpoint status [D]: a check mark indicates that the breakpoint is enabled, and a dash indicates that it is disabled. Additionally, disabled breakpoints are displayed in gray. Within the same column, a button [E] in front of the mark can be used to open the **Break.Set** window in order to edit the breakpoint.
- The symbolic address of the breakpoint [F]. **sieve\11** means for example the source code line 11 in function **sieve**.

When the program execution stops at a breakpoint, it is highlighted in the **Break.List** window.



# Variables

Understanding how to display and monitor variables is crucial when debugging with TRACE32 PowerView. Variables represent data storage locations within your program, and being able to observe their values while debugging your program can help you identify and fix issues quickly. For a more in-depth visual guide, you can explore our video tutorial

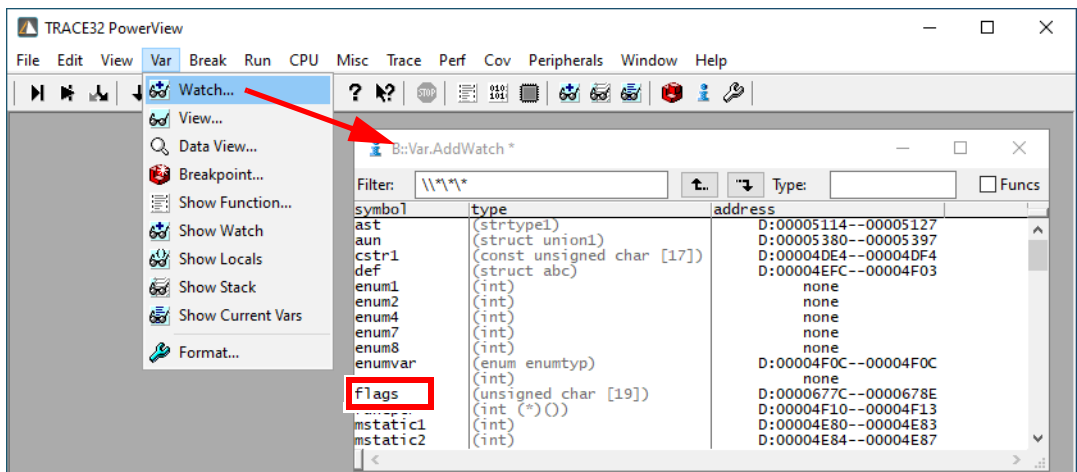
[support.lauterbach.com/kb/articles/variable-logging-and-monitoring-in-trace32](http://support.lauterbach.com/kb/articles/variable-logging-and-monitoring-in-trace32)

## Displaying Variables

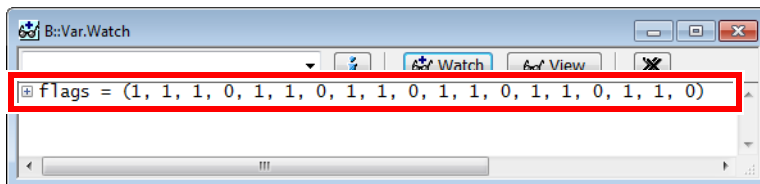
In this section, we'll walk through the process of displaying variables in TRACE32. Specifically, we'll display the variables **flags**, **def**, and **ast**.

Follow the steps below:

1. Open the Watch window:
  - Navigate to the menu and select **Var > Watch....**
  - This action opens the **Var.AddWatch** window, which lists all variables currently recognized by the symbol database.



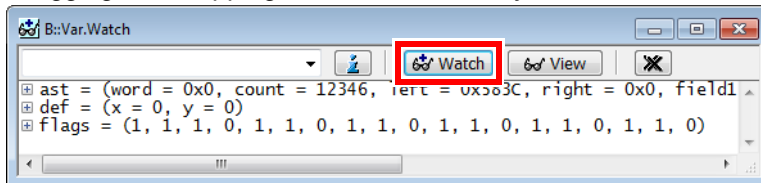
2. Add Variables to Watch:
  - Locate the variable **flags** in the **Var.AddWatch** window and double-click it.
  - This will open the **Var.Watch** window, displaying the current value of the **flags** variable.



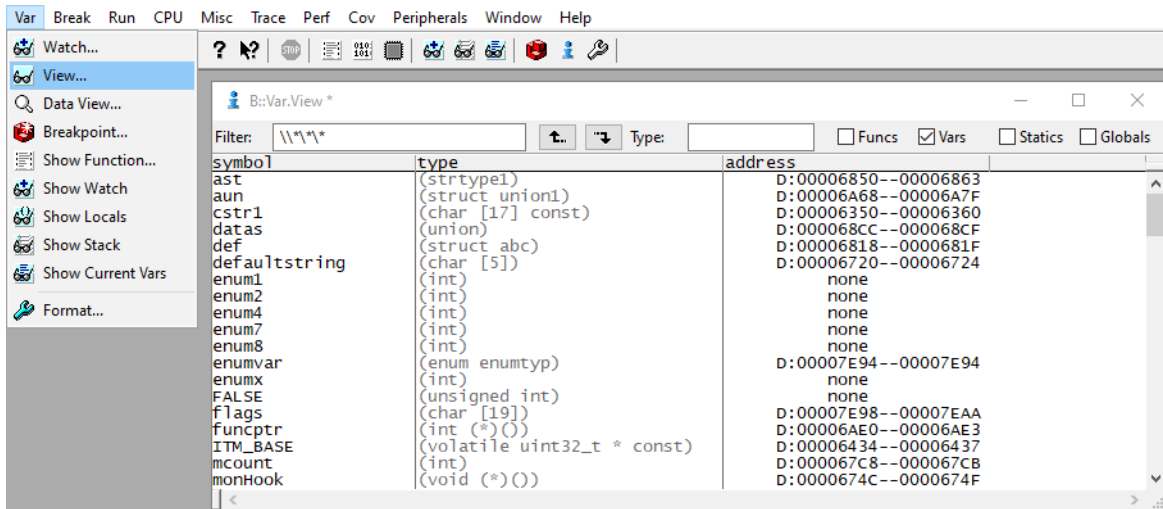
- Repeat the process for the **def** and **ast** variables by either:
  - Clicking **Watch** within the **Var.Watch** window and then double-clicking the desired variables,

or

- Dragging and dropping the variables directly from a **List** window into the **Var.Watch** window.



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



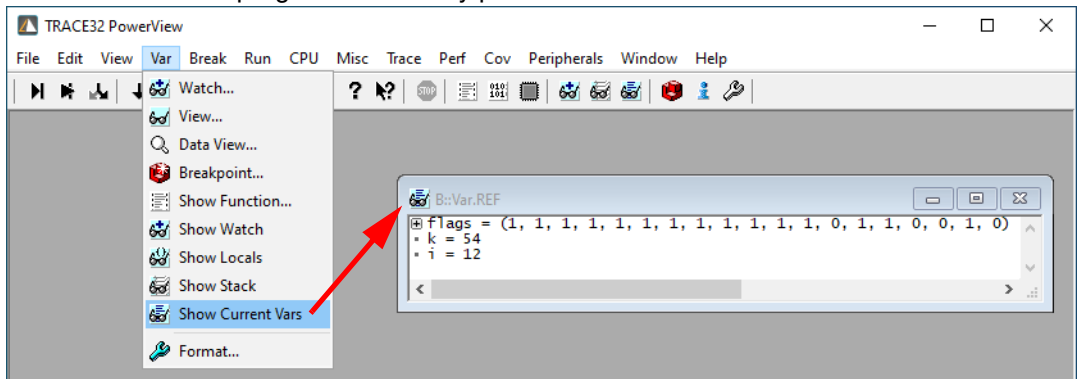
## Displaying Variables Referenced by the Current Source Line

When debugging, it's essential to focus on the variables relevant to the current context of your program. TRACE32 provides tools to easily display and monitor these variables, allowing you to observe how their values change as the program executes. This chapter will guide you through displaying variables specific to the current program context using TRACE32.


### Steps to display context-specific variables:

1. Run the program to the desired function:
  - Begin by running your program until it reaches the function you want to debug. Go to the menu **Run > Go Till...** then select the function with a double click
  - This action instructs TRACE32 to continue program execution until it hits the selected function, halting execution at its entry.
2. Display variables in the current context:
  - Select the menu **Var > Show Current Vars**.
  - The **Var.REF** window will open, displaying all variables accessed by the current program context. This window provides a dynamic view of the variables relevant to the function or code

block where the program is currently paused.



### 3. Monitor variable changes:

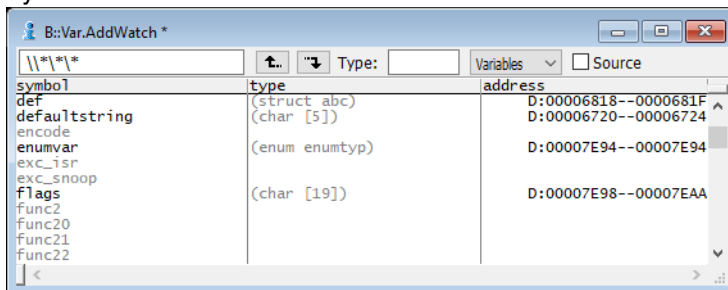
- To see how the variables change with each step of execution, use the **Step**  button on the TRACE32 PowerView toolbar.
- As you execute single steps, the **Var.REF** window will automatically update to reflect any changes in the variable values.

## Using the Symbol Browser

The symbol browser provides an organized overview of all variables, functions, and modules currently stored in the symbol database. This feature allows you to easily navigate through the program's symbols, making it simpler to inspect and monitor the components relevant to your debugging tasks.

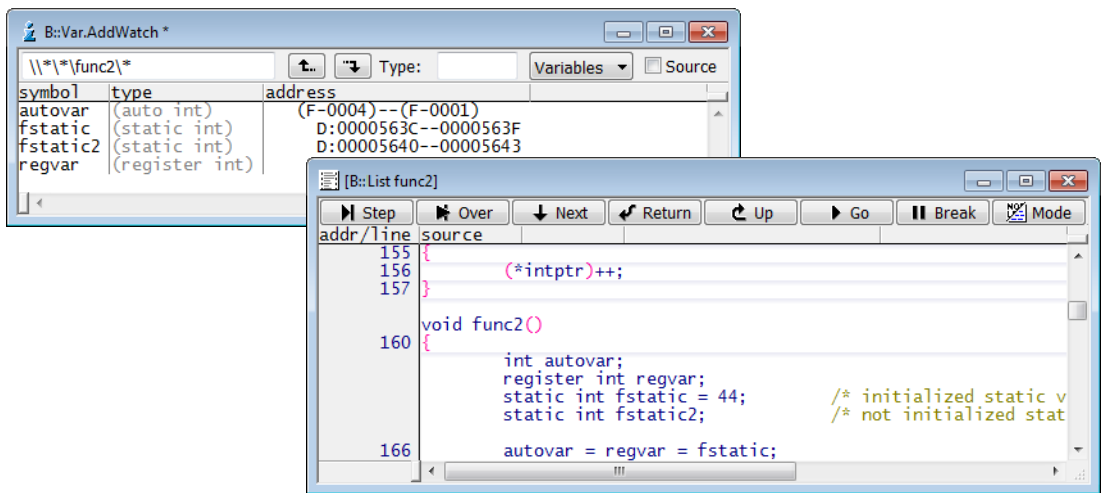
### 1. Access the symbol browser:

- To start, select **Var** from the main menu, and then choose **Watch....**
- This action opens the **Var.AddWatch** window, which allows you to browse through to the symbol database.



### 2. Navigating the symbol database:

- Within the **Var.AddWatch** window, you'll find a list of all the symbols stored in the database. Global variables are displayed in black while functions are displayed in gray. Double-clicking a function will display its local variables. For example, if you double-click on the function **func2**, TRACE32 PowerView will display all the local variables associated with this function. This allows you to monitor these variables specifically when the function is executed.



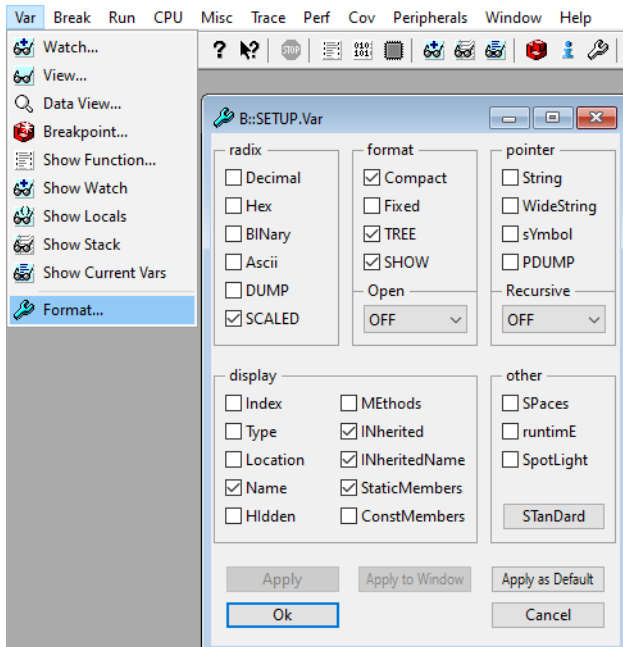
## Formatting Variables

Properly formatting the display of variables in TRACE32 PowerView can significantly enhance your debugging experience. By customizing how variables are presented, you can quickly grasp their values and types, making it easier to spot anomalies. TRACE32 allows you to format variables globally, affecting all displayed variables, or individually, tailoring the format for specific variables.

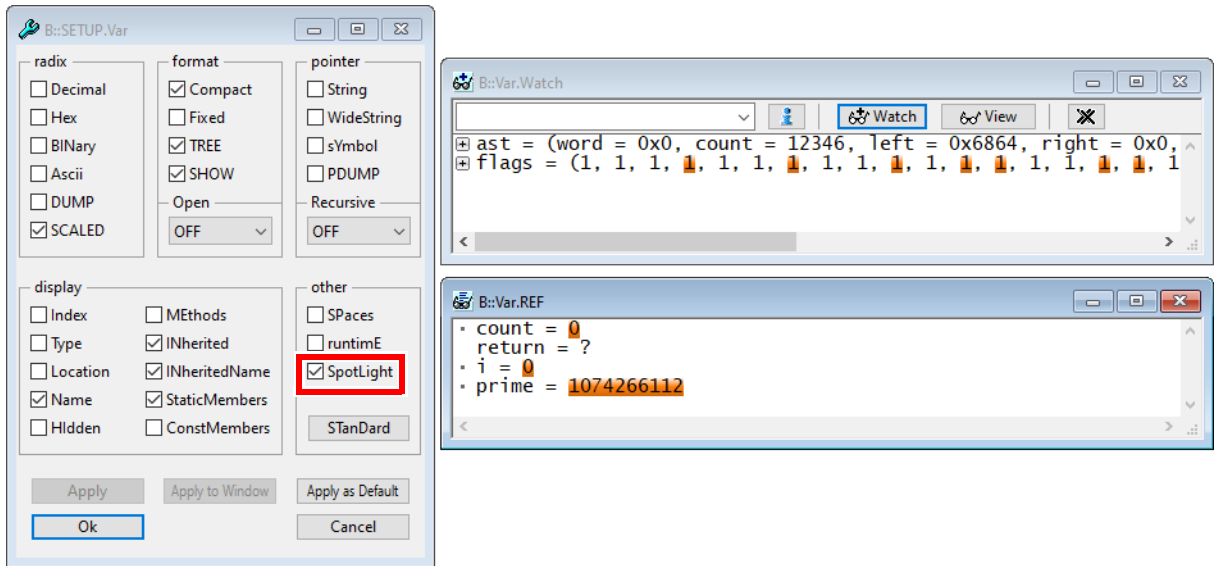
## Global Variable Formatting

Apply global formatting settings:

- Begin by selecting **Var** from the main menu, and then choose **Format**.
- This action opens the **SETUP.Var** window, where you can configure global formatting settings for all variables

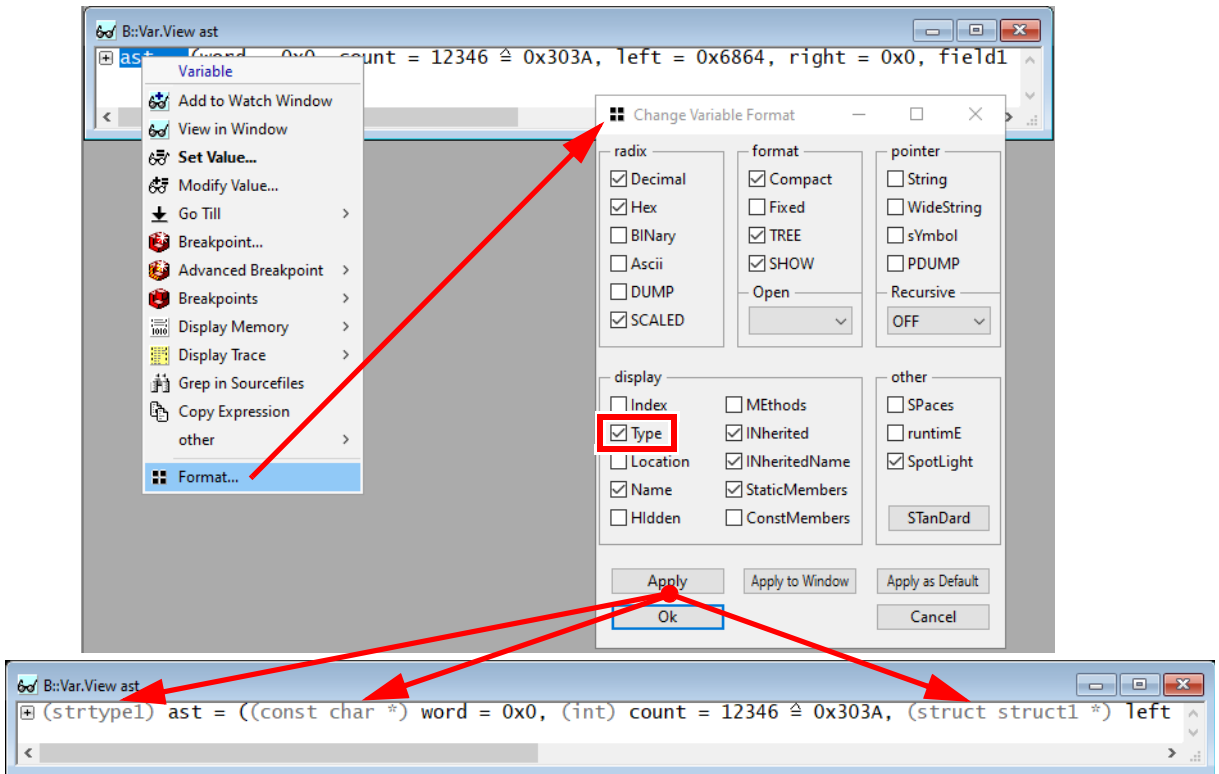


- Any changes you make here will apply to all **Var** windows that you open subsequently, ensuring a consistent display across your debugging environment. For example, by selecting **SpotLight**, all variable changes while debugging the program are highlighted.



## Individual Variable Formatting

- If you want to format a specific variable individually, start by displaying the variable in a **Var.View** window. For example go to the menu **Var > View** then select your variable from the symbol browser.
- In the **Var.View** window, right-click on the variable name and select **Format** from the context menu.
- The **Change Variable Format** dialog will open, providing various options to customize how the variable is displayed.
- To display the variable for instance with complete type information, check the **Type** box in the dialog.
- After making your selections, click **Apply**. The format of your variable in the **Var.View** window will be updated immediately, reflecting the new settings.

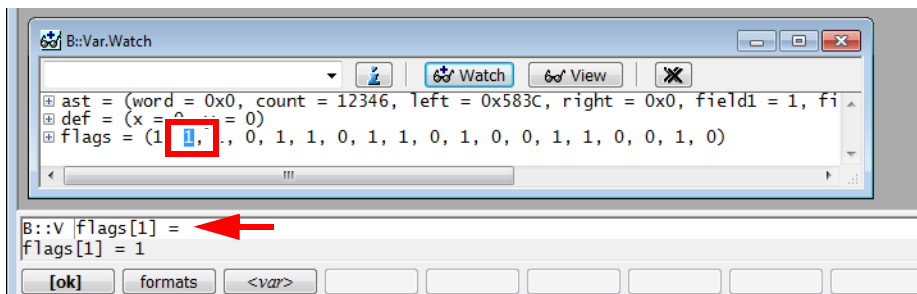


## Modifying Variables

During debugging, it's often necessary to modify variable values, e.g. to test different scenarios. TRACE32 PowerView makes it easy to change the value of a variable directly from the interface.

### Steps to modify a variable:

- To modify the value of a variable, simply double-click its current value in the **Var.View** window or any other window where the variable is displayed.
- The **Var.set** command will appear in the command line, followed by the (variable name and an equal sign (=), ready for you to input the new value.




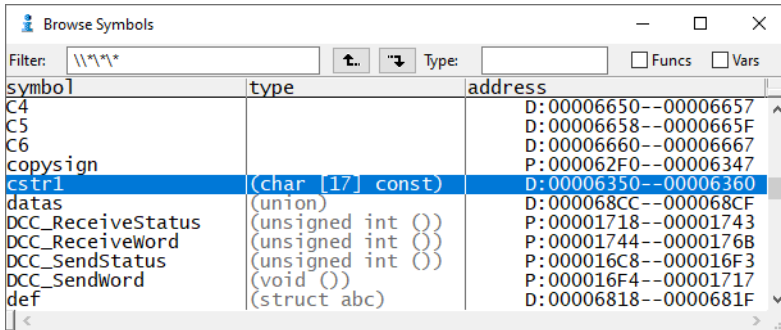
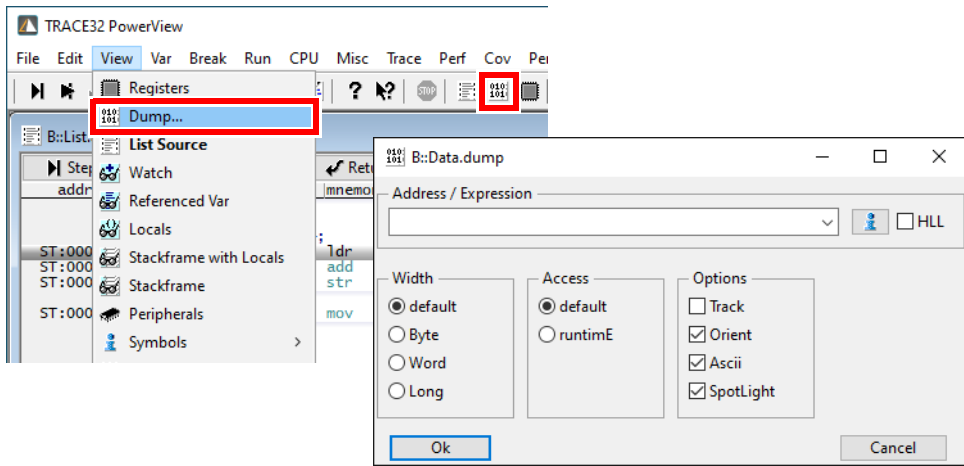
Enter the new value directly after the equal sign and press **[ok]** to confirm the change. The variable will immediately be updated with the new value.

## Displaying Memory

---

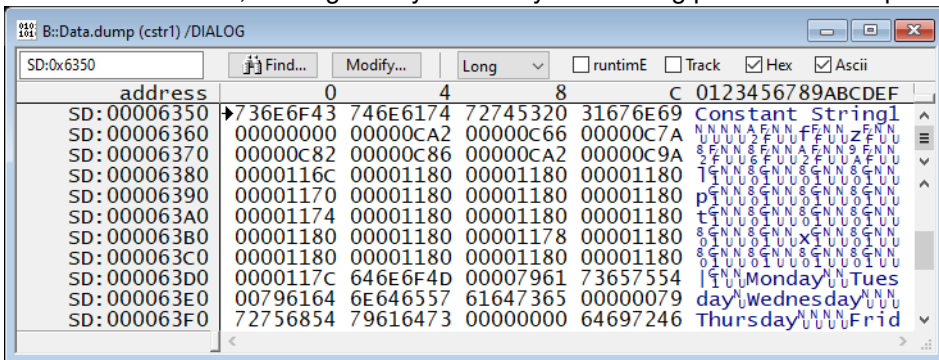
### Steps to display memory:

- Begin by selecting **View** from the main menu, then choose **Dump**.
- This action opens the **Data.dump** dialog, where you can specify the memory address or symbol you want to inspect.
- For instance, if you want to dump the memory associated with the array `cstr1`, you can easily select this symbol by:
  - Clicking on the  button to open the **Browse Symbols** window.
  - Double-clicking the symbol `cstr1` to select it.
  - After selecting the symbol, click **OK** to proceed.

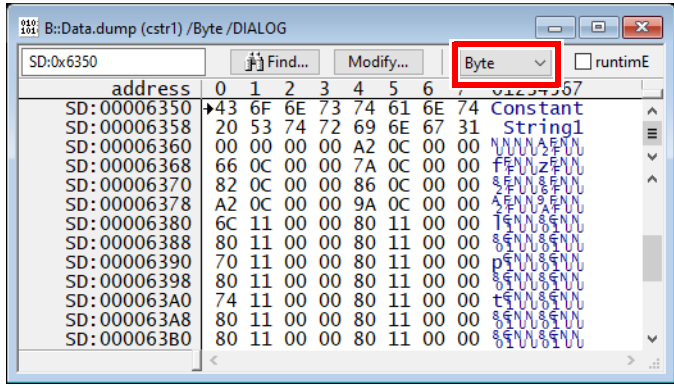


Double-click **cstr1**.

- The **Data.dump** window will open, displaying the memory content starting from the location specified by **cstr1**. By default, the memory is displayed in 32-bit (Long) hexadecimal values alongside their ASCII representation. This dual display allows you to see both the numeric data and its potential text representation. The current memory location being viewed is indicated by an arrow in the window, making it easy to identify the starting point of the dump.

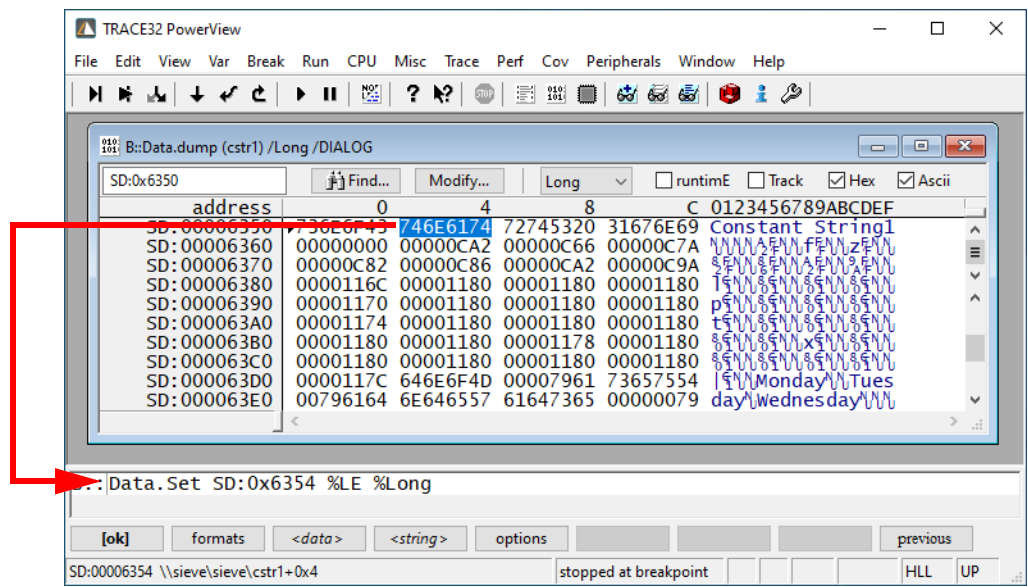


- If you wish to view the memory in a different format, you can select another display width directly from the **Data.dump** window, e.g. by selecting Byte in order to display memory as 8-bit values.



## Modifying Memory

- In the **Data.dump** window, locate the memory value you wish to modify.
- Simply double-click the value. This action prompts TRACE32 to generate a **Data.Set** command in the command line, corresponding to the selected memory address.
- After the **Data.Set** command appears in the command line, enter the new value you want to assign to the memory address then press **[ok]**.

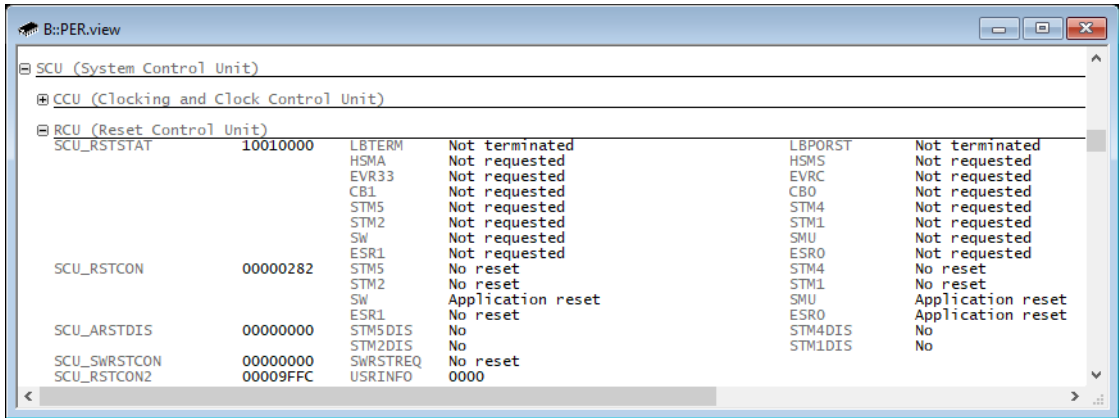


**NOTE:** %LE stands for Little Endian

# Peripheral View

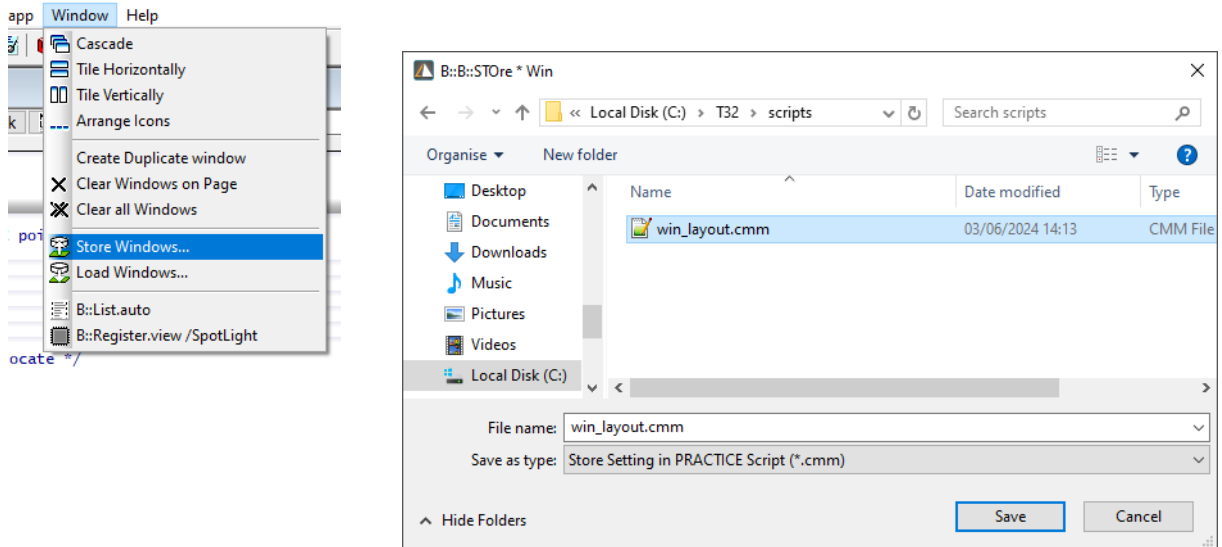
TRACE32 supports a freely configurable window for displaying and manipulating configuration registers and on-chip peripheral registers at a logical level. Predefined peripheral files are available for most standard processors/chips.

You can open the peripheral register view in the TRACE32 by selection the **CPU** menu, then **Peripherals**.



# Store Window Configuration

To save the window configuration for you next debug session use **Store Windows...** from the **Window** menu. This action generates a PRACTICE file that includes all commands to reactivate your complete window configuration automatically.



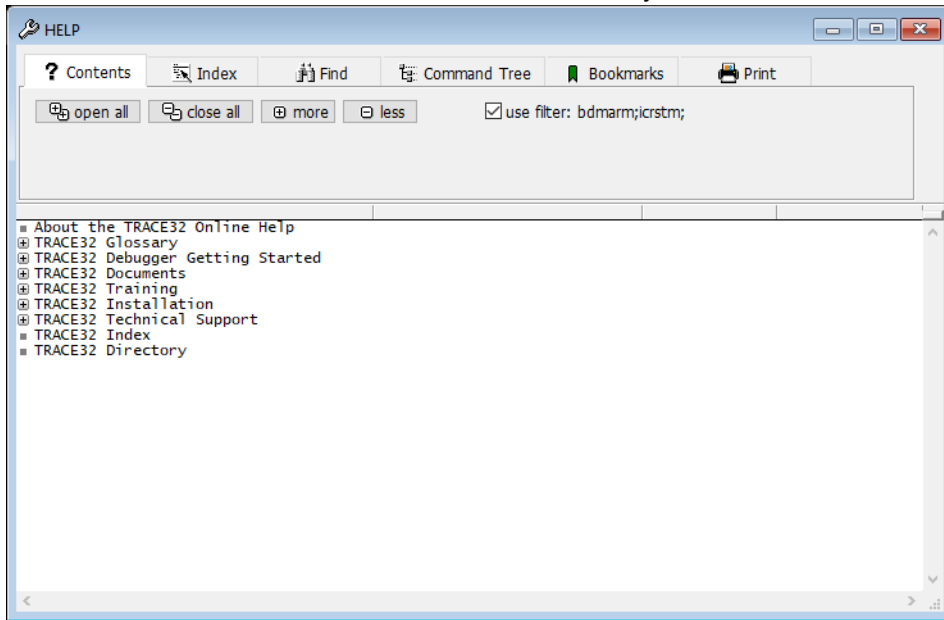
The saved window layout can be loaded again for the next debug session with the **Load Windows...** from the **Window** menu.

You can also add a call into your start-up file:

```
DO win_layout.cmm
```

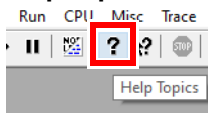
# Getting Online Help

The online help system consists of several documents. They are accessible as PDF-files directly from the TRACE32 software and can be found in the **pdf** directory.

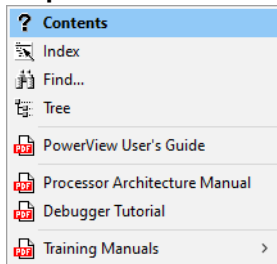


There are different ways to open the TRACE32 online help:

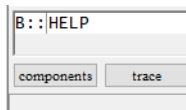
- **Help Topics** button on the toolbar



- **Help menu > Contents**

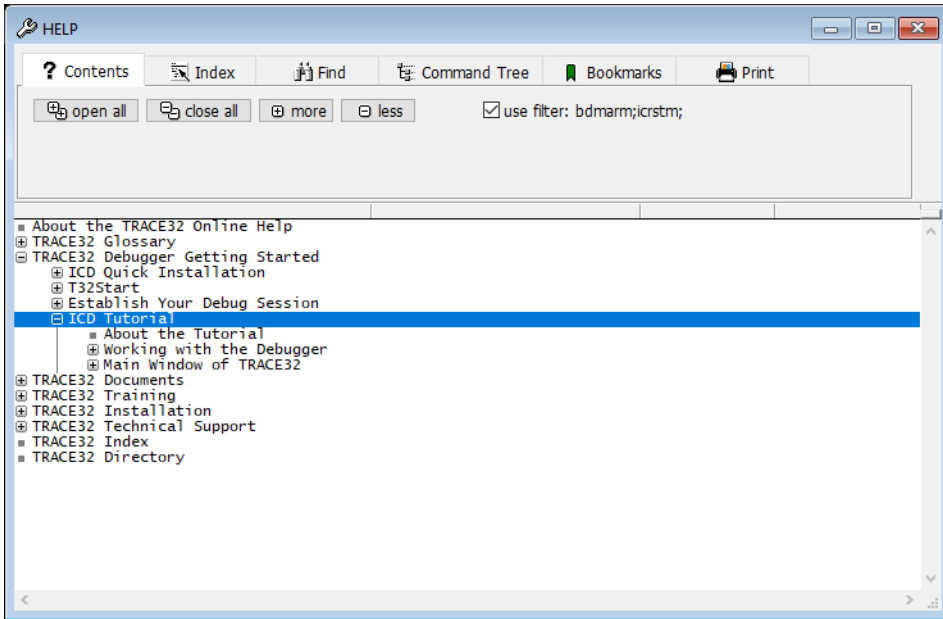


- **HELP** command in the command line



- **Help** button in the **Welcome to TRACE32!** dialog.

The help system is organized in a multilevel structured way. The screen below shows how to find this tutorial.




It is also possible to help for a single command. Enter the command into the command line, add a space and push F1.

# Contacting the Lauterbach Support

If you need assistance in setting up the debugging environment, be sure to include detailed system information.

1. To generate a system information report, choose **Help > Support > Systeminformation ...**

Generate TRACE32 Support Information

Press the following button to get help on how to generate Support Information: 

Company:  Department:

Prefix:

Firstname:

Surname:

Street:  P.O. Box:

City:  ZIP Code:

Country:

Telephone:

eMail:

Product:

Target CPU:

Hostsystem:

Compiler:

RealtimeOS:

Safe Mode:

Generate Support Information:

2. Fill in the form then click **Save to File**,
3. Send the system information as an attachment to [support@lauterbach.com](mailto:support@lauterbach.com) together with the problem description.