LAUTERBACH
DEVELOPMENT TOOLS

T32Start

Release 09.2023

MANUAL

T32Start

TRACE32 Online Help

TRACE32 Directory

TRACE32 Index

TRACE32 Debugger Getting Started ..... 🗁

T32Start **1**

History **4**

Introduction **5**

Features 5

Quick Start **7**

T32Start User Interface **8**

Buttons 8

Context Sensitive Menu 9

Mouse Actions 11

Configuration Tree: Settings **12**

Global Settings 12

Default Advanced Settings 13

Paths 13

License 14

Interfaces 15

Display Settings 18

Startup Script 19

Configuration Container and Configuration **21**

Podbus Device Chain 21

Connection Type 24

Target Option 27

MicroTrace 28

Software-only Debugging (Host MCI) 29

Debug Environment for Setup 1 (Single Instance) 31

Debug Environment for Setup 2 (Integrated Server) 32

Debug Environment for Setup 3 (Dedicated Server) 33

References to Tree Items **36**

Configuration Examples **37**

Hardware-based TRACE32 Tools 37

Single Core Debugging and Tracing (MicroTrace) 37

Multicore Debugging and Tracing (MicroTrace) 39

Single Core Debugging (USB 3) 41

Single Core Debugging and Tracing 44
Multicore Debugging (heterogenous AMP) 47
Multiprocessor Debugging 51
TRACE32 Software-only Tools 54
Instruction Set Simulator 54
Command Line Arguments **57**
Error Messages **58**

Version 09-Oct-2023

History

22-Dec-20	Revised manual.

Introduction

T32Start is only supported for Windows.

It can run on Linux and MacOSx using “WINE”.

The main objective of T32Start is to ease the setup of TRACE32 debug environments.

To start a TRACE32 instance a configuration file is required. The configuration file defines primarily:

- The host interface that is used to connect the TRACE32 debug hardware to the PC. Or alternatively the interface that is used by the TRACE32 Software-only tool.
- The environment variables that are required by the TRACE32 instance.

Please refer to **“Configuration File”** in TRACE32 Installation Guide, page 35 (installation.pdf) for more information about the syntax of the configuration file.

T32Start allows the user to do this setup in a graphical user interface instead of writing a configuration file manually. T32Start will generate a temporary configuration file based on the selected options and will set up the corresponding TRACE32 start parameters. The created configuration file as well as the command line options can also be viewed by the user.

Features

T32Start offers the following features for newcomers:

- Create a basic TRACE32 configuration for hardware-based TRACE32 tools (please refer to the configuration examples of the chapter **“Hardware-based TRACE32 Tools”** (app_t32start.pdf)).
- Create a basic TRACE32 configuration for TRACE32 Software only tools.
- Define a startup script that is executed when the TRACE32 instance is started.
- Specify the device names of the PowerDebug module.
- Specify the IP address of the TRACE32 PowerDebug module.

T32Start offers the following features for power users:

- Manage different TRACE32 configurations.
- Manage different TRACE32 configurations for different TRACE32 software versions.
- Configure the socket interface for the remote API in C (**"API for Remote Control and JTAG Access in C"** (api_remote_c.pdf)) or Python module PYRCL (**"Controlling TRACE32 via Python 3"** (app_python.pdf)).
- Configure the socket interface for controlling TRACE32 using the GDB protocol (**"TRACE32 as GDB Back-End"** (backend_gdb.pdf)).
- Configure Target Communication Framework (TCF) to control TRACE32 from within Eclipse (**"TRACE32 as TCF Agent"** (app_tcf_setup.pdf)).
- Configure the remote control for POWER DEBUG INTERFACE / USB (TCPUSB) (**"TRACE32 Installation Guide"** (installation.pdf)).
- Configure a License Pool Server for TRACE32 Software-Only tools that use AMP debugging (**"Floating Licenses"** (floatinglicenses.pdf)).
- Configure the TRACE32 instance for debugging and tracing a dedicated core via a PODBUS device chain.

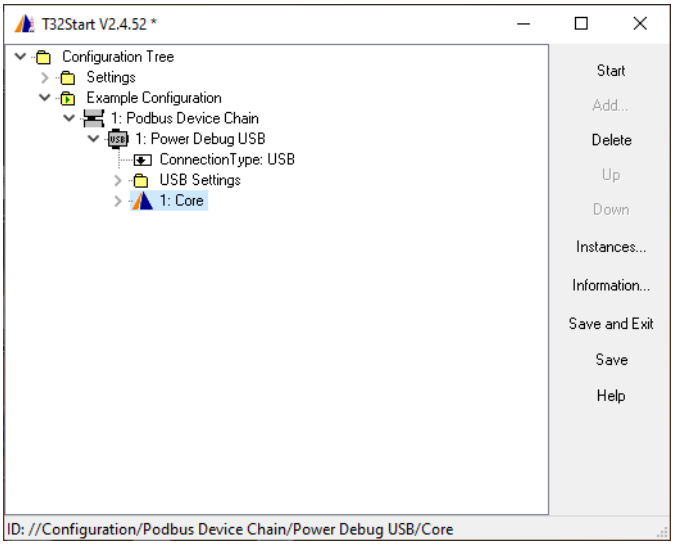
Quick Start

The executable **t32start.exe** is copied during the installation to the TRACE32 system directory. To start T32Start, do one of the following:

- Click the Windows **Start** button, and then select **T32Start**.
- Navigate to the TRACE32 system directory (by default **C:\T32**), sub-folders **bin\windows** or **bin\windows64**, and then double-click the **t32start.exe**.

  Example: `C:\T32\bin\windows64\t32start.exe`

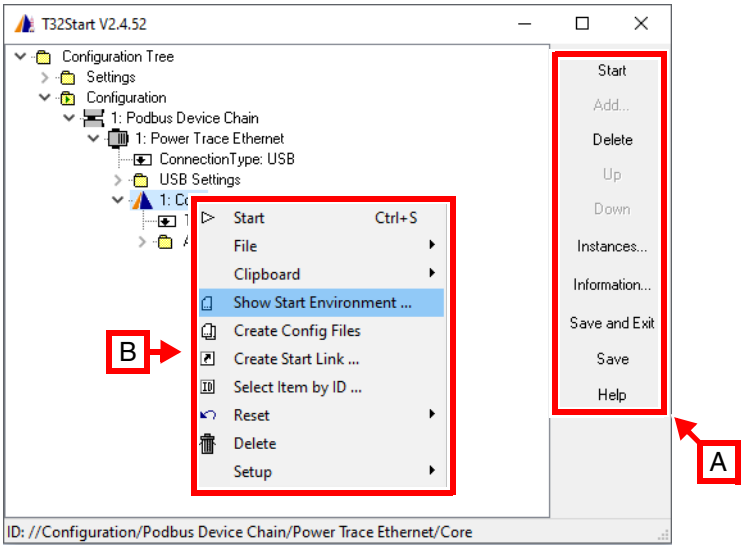
When T32Start is started the first time, the following basic settings are displayed:



T32Start User Interface

A set of often used functions is provided by buttons at the right side of the T32Start window **[A]**. Their action is executed in the context of the selected tree item except the global functions **Instances…**, **Save and Exit** and **Save**. If a function cannot be executed for the selected item the button is disabled.

Every tree item has additionally a context menu **[B]** with functions that can be performed on it and its sub-items.



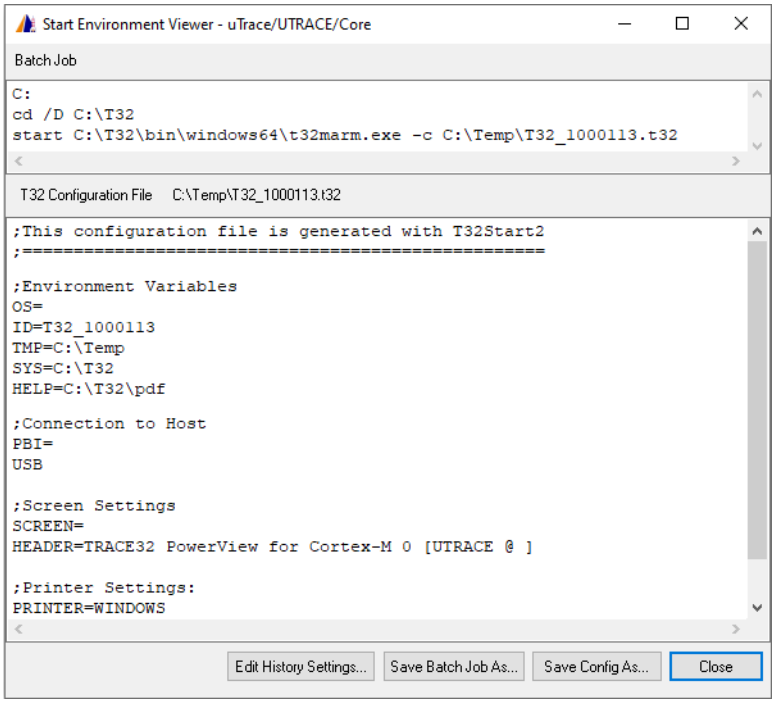
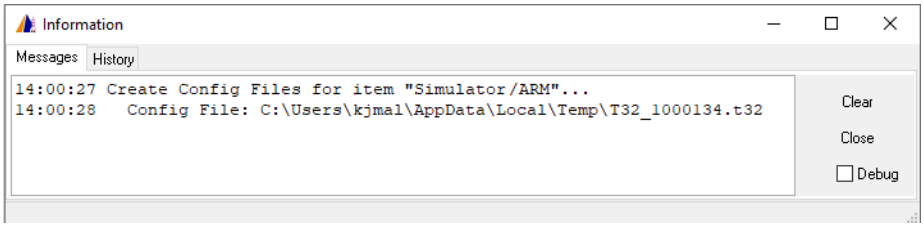
Buttons

Start	Starts the selected configuration.
Add...	Adds an item to the selected node.
Delete	Deletes the selected node.
Up	Moves up the selected node.
Down	Moves down the selected node.
Instances...	Opens a new window that displays details about all started TRACE32 instances.
Save and Exit	Saves modifications and exits T32Start.
Save	Saves modifications.
Help	Opens this document.

Context Sensitive Menu

Depending on the selected item, a subset of the following options is available:

Add	Adds an item to the selected node.
Start	Starts the selected node.
Up	Moves up the selected node.
Down	Moves down the selected node.
File	Every tree item can be loaded and stored into or from a ***.ts2** file. Analog to the copy and paste function, the loaded data can replace a tree item or it can be added as sub-item. At the top level node the function provides a method to backup or restore the complete **Configuration Tree**.
Clipboard	Every tree item has menu items for transferring data from and to the Clipboard. The data is handled as ASCII text. In contrast to the mouse actions, two methods for the paste command can be chosen. **Paste and add** will add the clipboard data as sub-items of the selected node. **Paste and replace** will replace the tree item.
Delete	Deletes selected node.
Clear Subitems	Clears all sub-items of selected node.
Select Item by ID...	Opens a window that displays the ID of each selected item. Refer to **"References to Tree Items"**, page 36 for more information.
Reset	• **Reset T32 History**: resets the TRACE32 history. Refer to the description of the commands **HISTory** and **AutoSTOre** for more information. • **Reset to Default Advanced Settings**: copies the settings from **Default Advanced Settings** to the node **Advanced Settings** of the selected configuration. • **Reset Paths to Default Advanced Settings**: copies the settings from **Default Advanced Settings > Paths** to the node **Advanced Settings > Paths** of the selected configuration. • **Reset unused Built-in Start-up Scripts**.
Setup	Register T32Start as default program for **.ts2** file extension.

Show Start Environment	Opens a windows displaying the start parameter as a batch job as well as the configuration file. The **Batch Job** field shows the DOS command script which can be used in order to start the TRACE32 instance. The **Configuration File** field displays the configuration file contents. The shown file names are valid also after closing T32Start. 
Create Start Link...	Opens a dialog in order to create links to T32Start on the Windows Desktop or Windows Start Menu: After the link is executed, T32Start will appear and start the connected item and all sub-items. After starting, T32Start will be closed automatically based on the settings.
Create Config Files	Creates a configuration file in the temporary directory. The **Information** window will pop-up displaying the file name and path of the created configuration file. 

Mouse Actions

The tree items can also be modified by mouse actions with drag & drop.

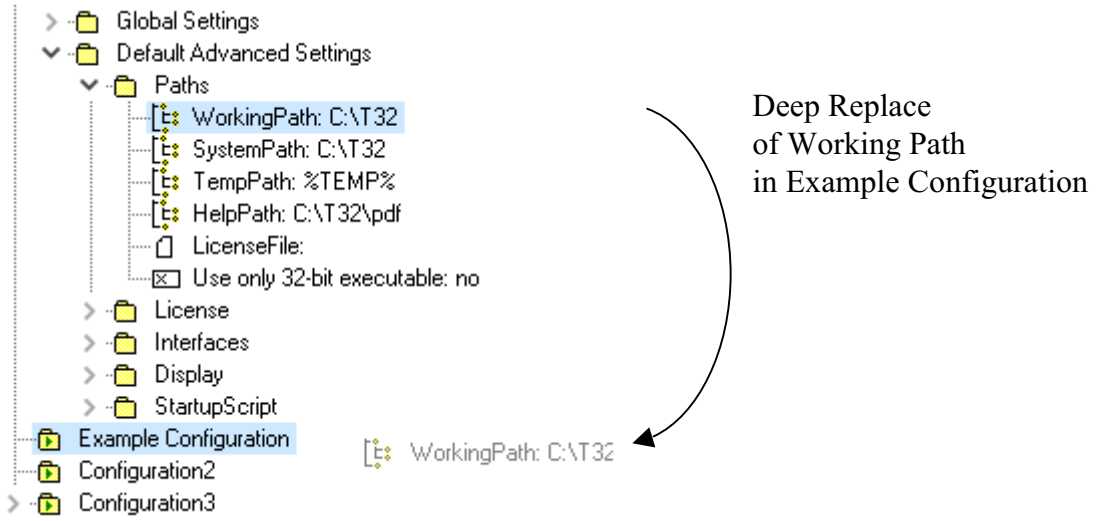
Actions with two Items involved

The possible actions take affect only on the two involved source and destination item. The matrix below shows the four cases that are possible in this case. The shift key switches between the move and copy mode. The append and replace mode is derived by the program automatically depending on the concrete situation.

Destination \ Source	Move	Copy
Replace		
Add		

Deep Replace

It is possible to replace none-deletable sub-trees by a template tree. This can be done by dragging the template tree and dropping it on the root of the tree, where the compatible sub-trees are to be replaced.



Configuration Tree: Settings

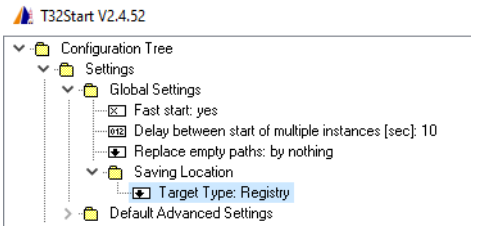
The ***Configuration Tree*** is the root of the tree and contains ***Configurations*** or ***Configuration Containers***. Both item types can be created and deleted from the tree. The ***Configuration Tree*** does also contain the ***Settings*** tree, where default settings are stored.

For details on Configuration Containers and Configurations refer to the chapter **"Configuration Container and Configuration"** in T32Start, page 21 (app_t32start.pdf).

This chapter introduces the items of the ***Setting Tree***.

Global Settings

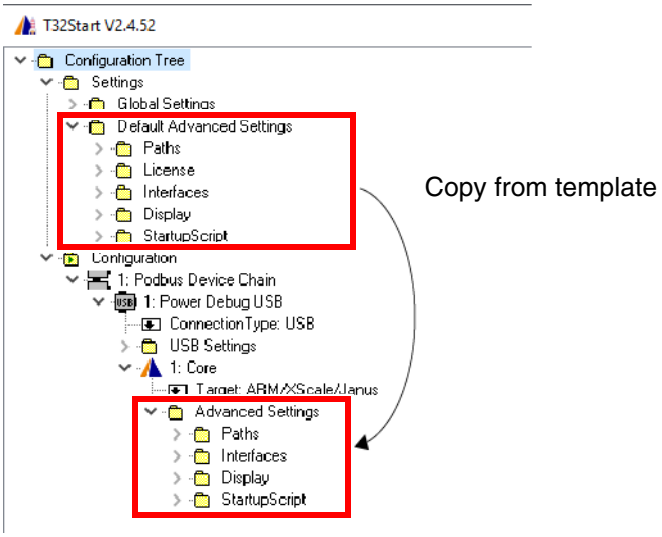
The ***Global Settings*** tree contains general settings.



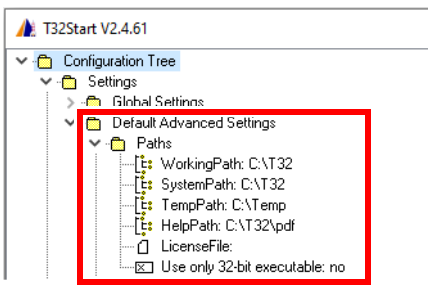
Global Settings	Description
Fast start	When set to yes, the next TRACE32 PowerView instance within a **Configuration** is directly started without an additional delay.
Delay between start of multiple instances	Defines the wait time in seconds between the start of two TRACE32 instances. This delay is only used if **Fast start** is set to no.
Replace empty paths	When set to ***by program directory***, T32Start suggests a TRACE32 installation in the current working directory whenever an item in the ***Advanced Setting*** sub-tree is left empty. The option ***by standard directory*** just suggests the settings from the ***Default Advanced Settings***.
Saving Location / Target Type	Target Type ***Registry*** will store the entire **Configuration Tree** into the Windows Registry under the `HKEY_CURRENT_USER` key.
Saving Location / File	When Target Type is ***File***, T32Start stores the **Configuration Tree** to the specified file. This will consume less memory in the registry and makes it possible to share the settings between different users. Evironment variables quoted by % characters are evaluated. The current configuration files can specified temporarly by command line arguments. Refer to **"Command Line Arguments"**, page 57 for more information. The file **t32start.default.ts2** is used as current configuration file when the file exists in the current working directory or in the system directoy of T32Start.

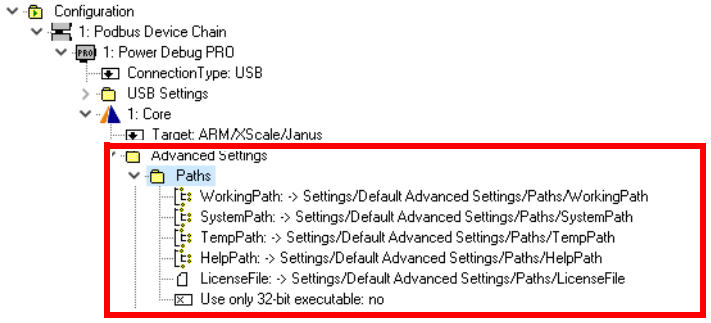
Default Advanced Settings

Items with a Lauterbach logo on their left side represent TRACE32 instances. TRACE32 instances need a number of **Advanced Settings**. Every time a new instance is created, the **Advanced Settings** are copied from the **Default Advanced Settings** located in the **Settings** tree.



Paths





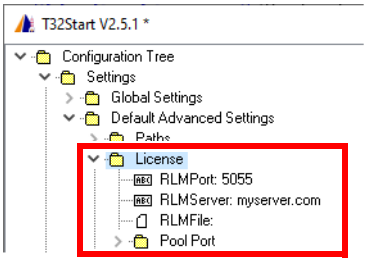
Item	Description
WorkingPath	Active directory after starting the TRACE32 instance. T32Start will change to this directory before TRACE32 PowerView is started. The selected directory will thus be the current working directory in the started TRACE32 PowerView instance. You can display this directory in TRACE32 PowerView using the command **PWD**.
SystemPath	Directory where the executable and system files of TRACE32 are located.
TempPath	Directory, where temporary files can be created. The source files are copied to the temporary directory while debugging.

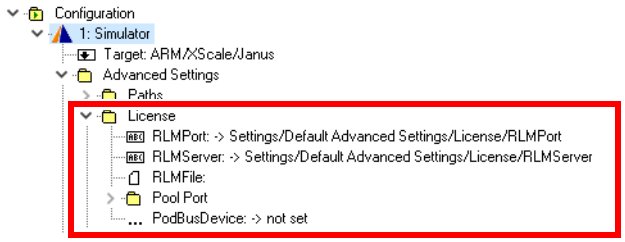
HelpPath	Directory where the pdf-files for the TRACE32 online help are located.
LicenseFile	Directory where a license file can be located. A license file provides the software maintenance keys.
Use only 32-bit executable	If set to yes, T32Start will start the 32-bit executable located under `bin\windows` instead of the 64-bit executable located under `bin\windows64`. This could be e.g. needed if a 32-bit DLL has to be loaded in TRACE32 PowerView.

Windows environment variables can be assigned to tree items designed to contain paths and file names. For example, if the Windows environment variable `%TEMP%` is assigned to the tree item **TempPath**, then TRACE32 and other applications share the user's default temporary directory. In addition to the Windows environment variables, there is the T32Start environment variable `%WORKINGDIR%`. It points to the initial working directory of T32Start.

License

The License node defines the floating license parameters for software-only TRACE32 tools. Please refer to **"Floating Licenses"** (floatinglicenses.pdf) for more information.

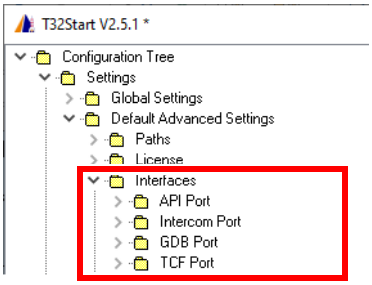


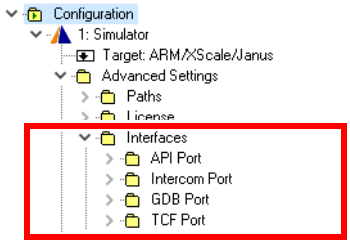


Item	Description
RLMPort RLMServer	The Floating License Client (RLM Client) needs to know which (RLM) Server to contact and which port number should be used to get the license.
RLMFile	Sets a license file (*.lic) which includes the floating license parameters.
Pool Port	TCP/IP port for license pool. Refer to the chapter **"Floating License Pools"** in Floating Licenses, page 19 (floatinglicenses.pdf) for more information.
PodBusDevice	Licenses the TRACE32 Simulator using a connected hardware-based debugger. This option is only available for the Simulator. Refer to **"Instruction Set Simulator"**, page 54 for more information.

Interfaces

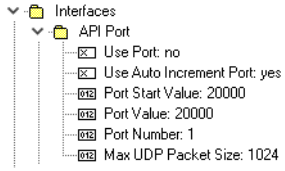
Defines the communication parameters between multiple TRACE32 PowerView user interfaces as well as the parameters to connect to TRACE32 PowerView using an external program via Remote API, GDB or TCF.





API Port

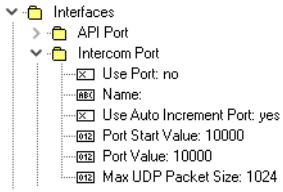
Defines the parameters for the TRACE32 Remote API for TCP/IP and UDP. Even if the configuration is performed for both protocols, it is recommended to use TCP/IP. Please refer to **"API for Remote Control and JTAG Access in C"** (api_remote_c.pdf) for more information and **"Controlling TRACE32 via Python 3"** (app_python.pdf).



Item	Description
Use Port	Lets the TRACE32 instance listen to the UDP/TCP port. Starting from the TRACE32 release 09.2020, the API supports per default TCP socket streams. Previous TRACE32 versions only support a communication via UDP sockets.
Use Auto Increment Port	To ensure unique port numbers it is possible to assign an automatically created port number to the TRACE32 instance. The port numbers are then unique within a single **Configuration**.
Port Start Value	When **Use Auto Increment Port** is false, the port number can be specified here. If the TRACE32 instance is the first one to be started in the **Configuration**, **Port Start Value** is the start value of the increasing port number set.
Port Value	The field is read-only and displays the resulting port number.
Port Number	Number of API connections to one single TRACE32 PowerView instance.
Max UDP Packet Size	Specifies the maximum data package length for UDP. No operation for TCP.

InterCom Port

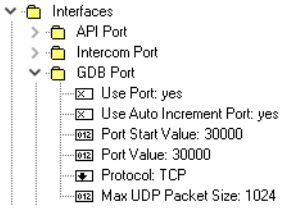
Defines the parameter for start/stop synchronization and for the communication between multiple TRACE32 PowerView instances. Refer for more information to the description of the **InterCom** command group.



Item	Description
Use Port	Lets the TRACE32 instance listen to the UDP port.
Name	Assign a name to the TRACE32 PowerView interface. This name can then be used with the **InterCom** commands. The selected name can be displayed in TRACE32 PowerView using the **SYnch.state** command.
Use Auto Increment Port	To ensure unique port numbers it is possible to assign an automatically created port number to the TRACE32 instance. The port numbers are unique within a single **Configuration** then.
Port Start Value	When **Use Auto Increment Port** is false, the port number can be specified here. If the TRACE32 instance is the first one to be started in the Configuration, **Port Start Value** is the start value of the increasing port number set.
Port Value	The field is read-only and displays the resulting port number.
Max UDP Packet Size	Specifies the maximum data package length for UDP.

GDB Port

Define the parameters for controlling TRACE32 PowerView as GDB Back-End. Please refer to **"TRACE32 as GDB Back-End"** (backend_gdb.pdf) for more information.

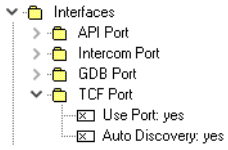


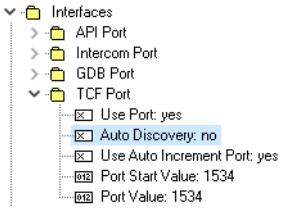
Item	Description
Use Port	Lets the TRACE32 instance listen to the UDP/TCP port.
Use Auto Increment Port	To ensure unique port numbers it is possible to assign an automatically created port number to the TRACE32 instance. The port numbers are unique within a single **Configuration** then.

Port Start Value	When **Use Auto Increment Port** is false, the port number can be specified here. If the TRACE32 instance is the first one to be started in the Configuration, **Port Start Value** is the start value of the increasing port number set.
Port Value	The field is read-only and displays the resulting port number.
Protocol	TCP (default) or UDP.
Max UDP Packet Size	Specifies the maximum data package length for UDP. No operation for TCP.

TCF Port

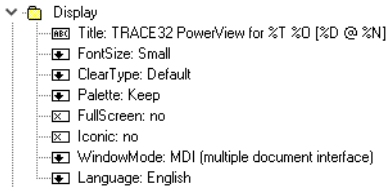
Defines the parameter for controlling TRACE32 PowerView via Target Communication Framework (TCF) from an Eclipse-based interface. Please refer to **"TRACE32 as TCF Agent"** (app_tcf_setup.pdf) for more information.





Item	Description
Use Port	Lets the TRACE32 instance listen to the TCP port.
Auto Discovery	Enable/disable the TCF discovery. If the discovery is disabled, a TCF port number must be specified. The TCF discovery is a mechanism where agents advertise their peers by sending UDP packets to other agents.
Use Auto Increment Port	To ensure unique port numbers it is possible to assign an automatically created port number to the TRACE32 instance. The port numbers are unique within a single **Configuration** then.
Port Start Value	When **Use Auto Increment Port** is false, the port number can be specified here. If the TRACE32 instance is the first one to be started in the Configuration, **Port Start Value** is the start value of the increasing port number set.
Port Value	The field is read-only and displays the resulting port number.

Display Settings



Item	Description
Title	Sets the window title of the TRACE32 instance. Generic placeholders are specified to include details such as the core architecture (%T) or the device name (%D) into the tille. A list of all available placeholder is displayed, when the Title item is selected.
Font Size	Selects the used font size used by the TRACE32 instance (**Normal**, **Small** or **Large**).
Clear Type	**Yes**: Cleartype display of fonts is switched ON if it is supported by the OS. The monospaced truetype font "Lucida Console" is used as basic font and should be installed. **No**: Cleartype display of fonts is switched OFF. TRACE32 fonts are used (t32font.fon).
Palette	Sets up display theme. **Default**: use the default TRACE32 PowerView theme. **Keep**: use the last selected theme. **Dark**: use the TRACE32 PowerView dark theme.
Full Screen	If set to true, the TRACE32 instance is started in full screen mode.
Iconic	If true, the TRACE32 instance is iconic after starting.
WindowMode	The following TRACE32 window modes are available: FDI, MDI, and MTI. For descriptions of the window modes, click the blue hyperlinks.
Language	Set up the language used by TRACE32 (English or Japanese).

Startup Script

When a TRACE32 instance starts, the PRACTICE script **autostart.cmm** is executed, which then calls the following scripts:

- **system-settings.cmm** (from the TRACE32 system directory, usually C:\t32)
- **user-settings.cmm** (from the user settings directory: on Windows %APPDATA%\TRACE32 or ~/.trace32 otherwise)
- **work-settings.cmm** (from the current working directory)

In T32Start you can specify an **additional** PRACTICE script which is automatically started afterwards.

Item	Description
Source	T32Start supports two types of start-up scripts: • **File** • **Built-in Script** When **File** is chosen as **Source**, the script assigned to the **File** item will be executed.
File	If the **Source** item is set to **File**, specify the start-up script here.
Parameters	Set the parameters that are passed to the command script from **File** or **Built-in Script**. The parameter syntax is specified by the PRACTICE command **ENTRY**.
Built-in Script	The start-up script can be edited and stored directly in T32Start. Select the item and press the **Edit** button to edit the script.
Safe Start	Suppresses the automatic execution of any PRACTICE script after starting TRACE32. This allows you to test or debug the scripts that are normally executed automatically.

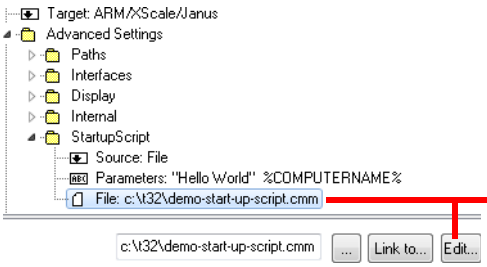
Example:

This example is for demo purposes only. It assumes that the following settings are made in the **Startup Script** tree item:

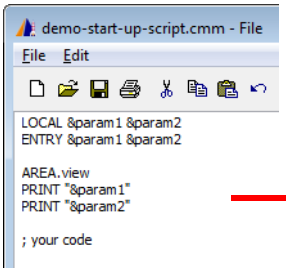
- **Source**: File
- **Parameters**: `"Hello World" %COMPUTERNAME%`
- **File**: c:\t32\demo-start-up-script.cmm

When TRACE32 is started via T32Start, the parameters are passed to the specified PRACTICE start-up script (*.cmm). In this example, the script is programmed to open an **AREA.view** window in TRACE32 and display the parameters.

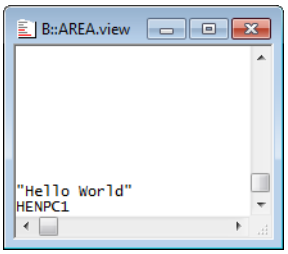
T32Start



Start-up script (demo)



TRACE32



`%COMPUTERNAME%` is an environment variable of Windows. For information about the environment variables of T32Start, see **"References to Tree Items"**, page 36.

Configuration Container and Configuration

A **Configuration** is the top most startable item in the configuration tree. It contains other startable items e.g. **Podbus Device Chain** or **Simulator**. When a Configuration is started, all startable sub-items are started one after the other. The start order is defined by the numeration of the items. Numerated items can be moved up or down with the buttons in the device specific pull-down menu.

It is possible to organize **Configurations** in **Configuration Containers**. Configuration Container can have other **Configuration Container** as sub-items.

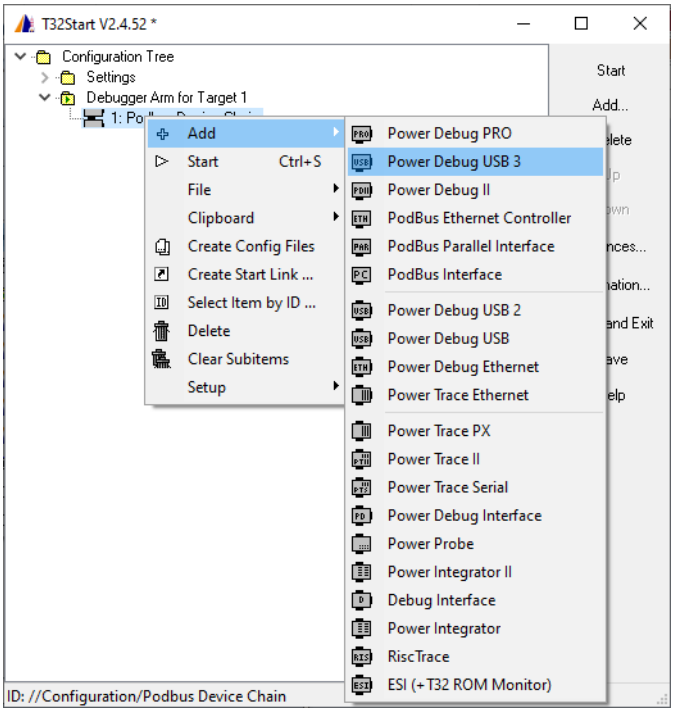
The following items can be added to a **Configuration**:

- **Podbus Device Chain**
- **MicroTrace**
- **MCI Server / MCI Lib Debugger**: refer to **"Software-only Debugging (Host MCI)", page 29**.
- **Simulator**: **TRACE32 Instruction Set Simulators**.
- **GDB Debugger**: refer to **"TRACE32 as GDB Front-End"** (frontend_gdb.pdf) for more information.
- **Host Process Debugger**: refer to **"Native Process Debugger"** (windows_debugger.pdf) for more information.
- **Serial ROM Monitor**: allow to create a configuration for a TRACE32 serial ROM Monitor (legacy).
- **Arbitrary Program**: allow to specify an external program that will be started together with the TRACE32 instances.
- **Note**: Allows to add a comment or a note to the configuration.
- **URL**: Allows to add an HTML link to the configuration. This allows to easily link to information relevant to the configuration.

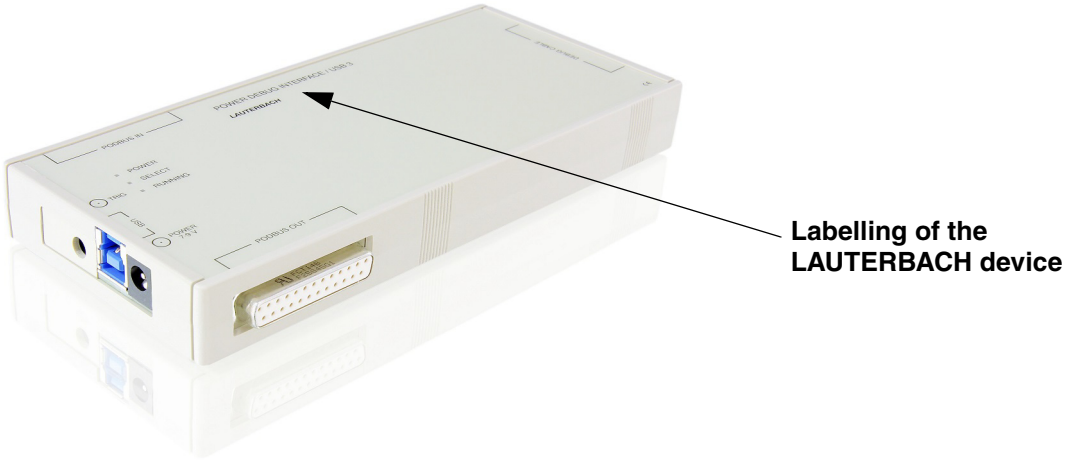
Podbus Device Chain

Several LAUTERBACH devices can be arranged in a **Podbus Device Chain**. Podbus is a proprietary bus used by LAUTERBACH to connect several devices. The order and type of LAUTERBACH devices has to be modelled as sub-items in the **Podbus Device Chain**.

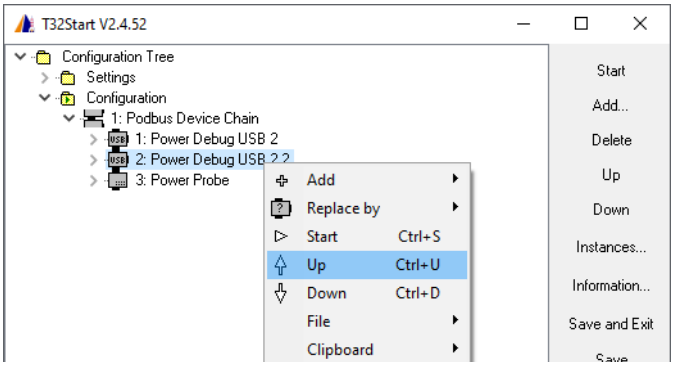
The **Podbus Device Chain** defines which LAUTERBACH devices are used for debugging.



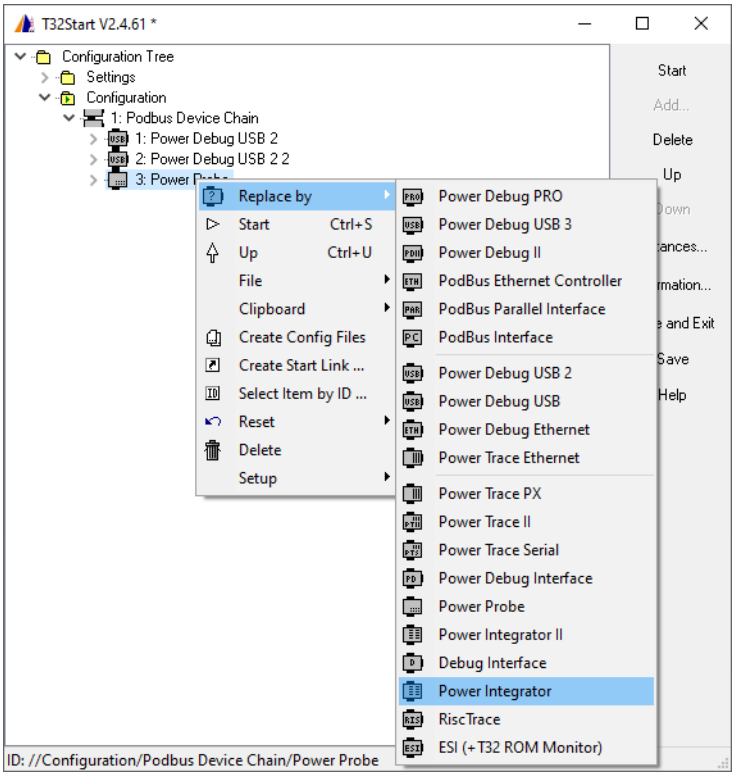
The correct name of your LAUTERBACH device can be read from the labelling.



The device which contains the host interface has to be the first device in the **Podbus Device Chain**. The **Up** and **Down** buttons can be used to change the device order.



Every device has a menu item **Replace by** to exchange it by a another device. While the exchange takes place the settings of the device are kept if possible.

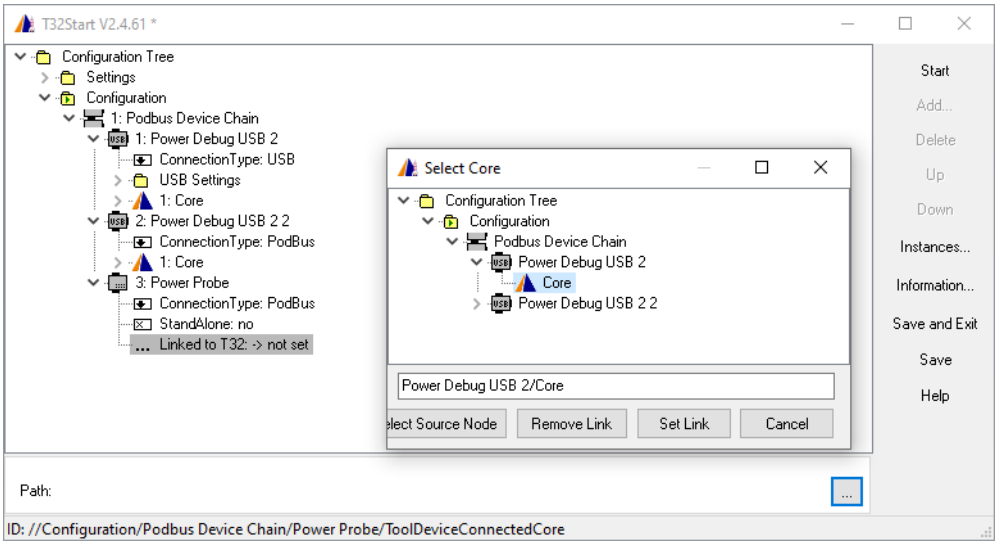


Please note that LAUTERBACH devices which have a **PODBUS SYNC** connector (instead of **PODBUS IN**), require an own power supply and connection to the host (USB or Ethernet). These devices cannot be added to a single **Podbus Device Chain**. Each device has to be configured as a separate **Podbus Device Chain**. The type of the PODBUS connection is printed on the device.

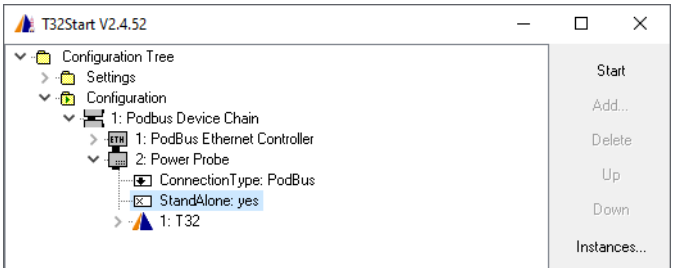
Debug devices can have a **Core** as sub-item. If a multicore (single device solution) debug environment is configured, a debug device can have more than one core as sub-item.

There are devices (e.g. Power Probe, Power Integrator) that can be used as add-ons for a debug process. In this case they are controlled by the TRACE32 instance assigned to the debug process for a specific **Core**. Therefore these add-on devices has to be linked to the appropriate **Core**.

In the following example the **Power Probe** is linked to the core assigned to the first Power Debug USB II.



The Power Probe and the Power Integrator can also be used as stand-alone devices. In this case an appropriate host interface is required.



Connection Type

The **Connection Type** defines the interface to the host. The available connection types depend on the selected device. Each TRACE32 device supports a subset of the following connection types:

- **USB**
- **Ethernet**
- **Citrix**
- **USB Proxy**: allows to communicate with a Power Debug Interface USB from a remote PC using the command line tool **t32tcpusb.** Please refer for more information to **"Example: Remote Control for POWER DEBUG INTERFACE / USB"** in TRACE32 Installation Guide, page 46 (installation.pdf).
- **Parallel**: only supported by the Podbus Parallel Interface.
- **Podbus / Podbus Express**: when more than one module is added to a single PodBus Device Chain, Podbus or Podbus Express is the only available connection type, depending on the device type, for devices which are not at the first position in the Podbus Device Chain.

USB Settings

Option	Description
Device Name	If multiple devices are connection to the host via USB, you can address each device using its device name. You can display and change the device name in the TRACE32 PowerView user interface using the menu **Misc > Interface Config** or via the TRACE32 command line using the **IFCONFIG.state** command.
Connection Mode	Defines the behavior when a debugger module, connected via USB, is already in use. Please note that this setting only have effect when the option **Exclusive** is set to **yes**. • **Normal**: a warning window appear. The connection is closed after confirmation. • **Auto Abort**: the TRACE32 executable will be closed automatically without any user interaction. • **Query Connect**: the user will be asked if the connection shall be forced. • **Auto Connect**: the TRACE32 executable will automatically take over control over the debugger module, even if the debugger is already in use. • **Auto Retry**: the TRACE32 executable will wait until the current TRACE32 session ends. Please refer for more information to **"Parameters for the PBI Driver with LAUTERBACH Tools"** in TRACE32 Installation Guide, page 44 (installation.pdf).
Exclusive	Tells TRACE32 that there can be only one TRACE32 PowerView instance to connect.

USB Proxy Settings

Please refer for more information to **"Example: Remote Control for POWER DEBUG INTERFACE / USB"** in TRACE32 Installation Guide, page 46 (installation.pdf).

Option	Description
Device Name	If multiple devices are connection to the host via USB, you can address each device using its device name.
Node Name	IP address of PC that runs t32tcpusb.
Port	Port number that was specified when t32tcpusb was started.

The options Max UDP Packet Size, Packet Burst Limitation, Compression and Delay can be useful when the net work connection is slow or many routers are involved.

Option	Description
Node Name / IP Address	Network address for Podbus device. You display and edit these settings in the TRACE32 PowerView user interface using the menu **Misc > Interface Config** or via the TRACE32 command line using the **IFCONFIG.state command.**
Port	Sets the UDP communication port from the PC to the debugger module (default is 20000). For AMP, all instances must use the same port number.
Host Port	Defines the UDP communication port from the debugger module to the PC (default is PORT+n).
Increment Port	When set to yes, **Port** and **Host Port** will be automatically incremented for each additional **Core** within a single **Configuration**.
Max UDP Packet Size	The network can have a limited UDP packet size below the default of 1024 bytes. Setup this value to limit the maximum packets the TRACE32 will send through the network.
Packet Burst Limitation	Sends only very small packets.
Compression	If enabled reduces the packet size by compression.
Delay	Delay time between sending two UDP packets. This can avoid packet order changing when connection is established through internet.
Exclusive	Tells TRACE32 that there can be only one TRACE32 PowerView instance to connect.

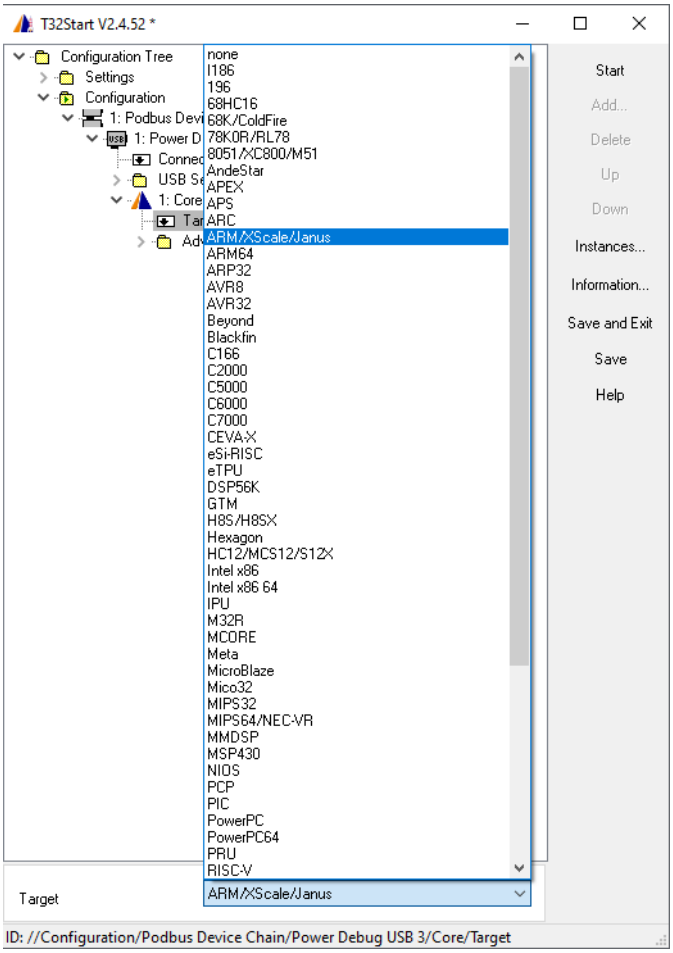
Parallel Port Settings

Option	Description
Port	Port number e.g. for LPT2 the port number is 2.
Access Mode	TRACE32 driver mode. PARPORT is the most recommended option. Choose LPT if PARPORT does not work.
Connection Mode	Parallel port interface mode according to your BIOS settings. Valid modes are: Standard, ECP, EPP.

Option	Description
Port	IO Communication Port

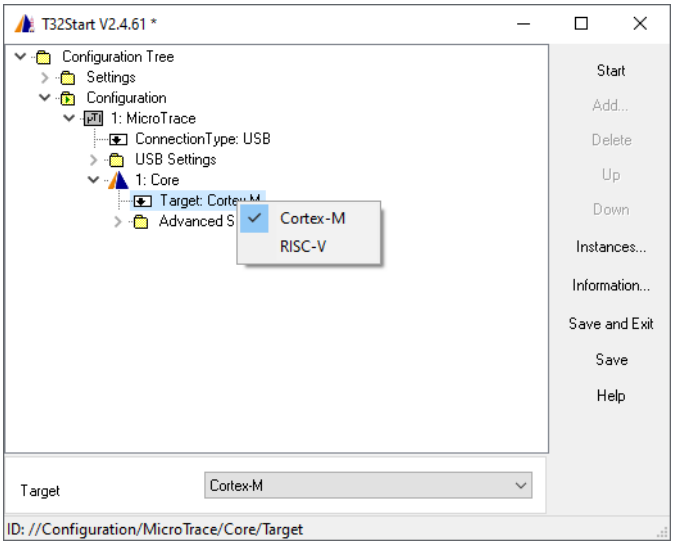
Target Option

The **Target** sub-item of a TRACE32 instance defines the executable to be started to support the target architecture of the dedicated debug process.



The TRACE32 µTrace for Cortex-M and RISC-V 32-bit processors can only be used a stand-alone device and cannot be part of a **PodBus Device Chain**. The µTrace only offers a USB connection to the host. The available USB settings are the same described under **"USB Settings"**, page 25.

One or multiple cores can be assigned to the µTrace. The only available architectures under **Target** are Cortex-M and RISC-V.



Software-only Debugging (Host MCI)

The Lauterbach TRACE32 Integrated Debug Environment supports debug sessions *with* and *without* hardware. Without hardware means software-only debugging - without TRACE32 debugger hardware.

This chapter describes how to configure software-only debug environments in T32Start for use in TRACE32 PowerView, the graphical user interface of TRACE32.

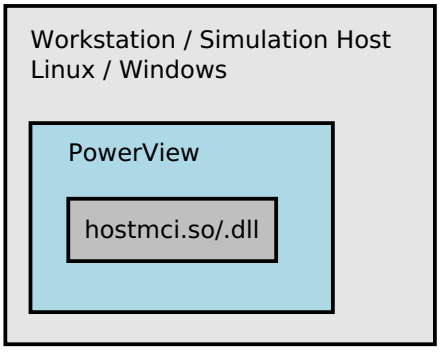
The TRACE32 PowerView instances can be set up in different ways.

1. A single TRACE32 PowerView instance runs on the same host as the back-end, see Setup 1. This configuration can't handle AMP debug scenarios.
2. Multiple TRACE32 PowerView instances run on the same host as the back-end, see Setup 2.
3. The TRACE32 PowerView instances run on a dedicated workstation; the back-end runs on another host, see Setup 3.

The T32Start application assists you in configuring the desired setup. This way you do not need to manually edit any config.t32 file. Simply choose the setup you need, and then follow the cross-reference at the bottom of the chosen setup diagram.

Setup 1

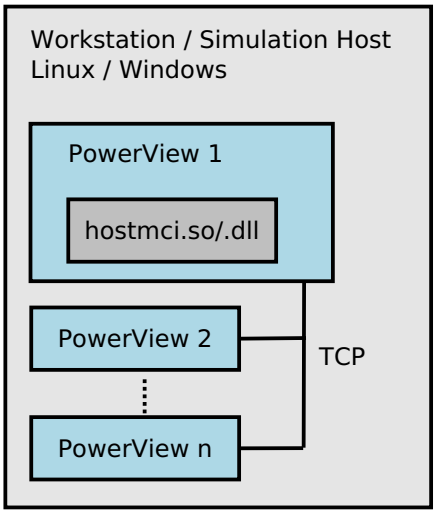
Setup with a single TRACE32 PowerView instance running on the same host as the back-end:



For step-by-step instructions on how to configure the above setup in T32Start, see **"Debug Environment for Setup 1 (Single Instance)"**, page 31.

Setup 2

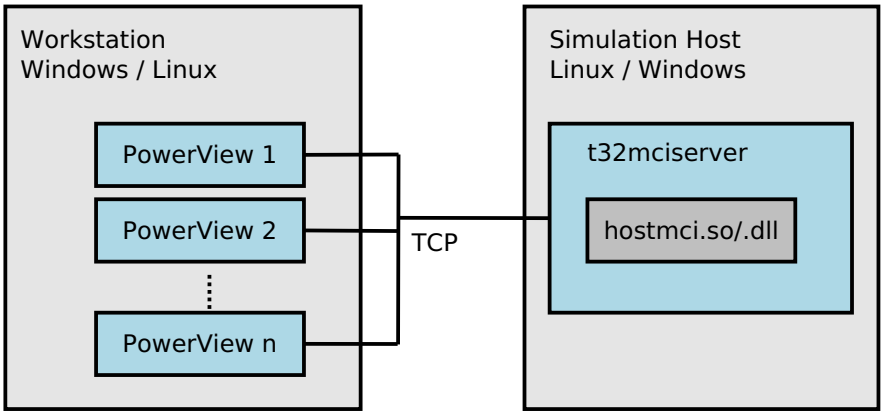
Setup with multiple TRACE32 PowerView instances (AMP) running on the same host as the back-end:



For step-by-step instructions on how to configure the above setup in T32Start, see **"Debug Environment for Setup 2 (Integrated Server)"**, page 32.

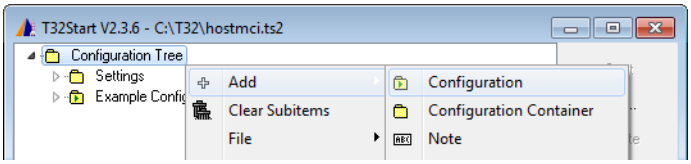
Setup 3

Setup with multiple TRACE32 PowerView instances (AMP) running on another host:



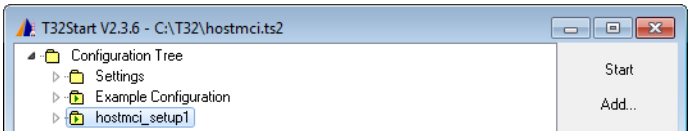
For step-by-step instructions on how to configure the above setup in T32Start, see **"Debug Environment for Setup 3 (Dedicated Server)"**, page 33.

1. In the **T32Start** window, right-click **Configuration Tree**, point to **Add**, and then select **Configuration**.



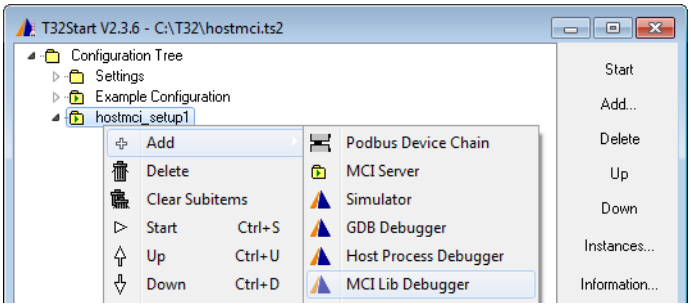
A tree item named **Configuration***<number>* is displayed.

2. Press **F2** to rename the new **Configuration***<number>* tree item to a meaningful name, for example, **hostmci_setup1**.

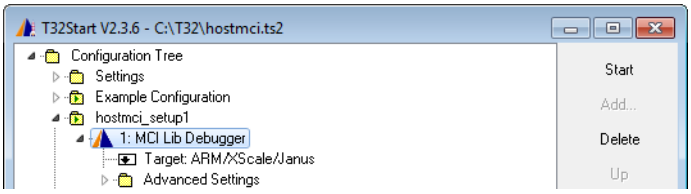


MCI Lib Debugger and Target

3. Right-click the renamed tree item, point to **Add**, and then select **MCI Lib Debugger**.



4. Click the little triangle next to **MCI Lib Debugger** to expand the tree node and navigate to **Target**.



5. Right-click **Target**, and then select the target name you want from the **Target** drop-down list.

This completes the steps for setup 1. The remaining steps depend on your project, see **Advanced Settings** below.

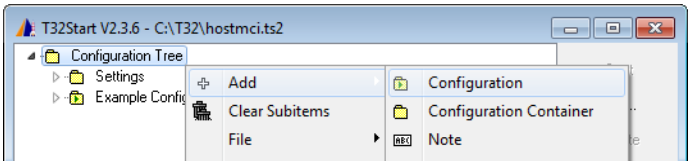
6. Configure the **Advanced Settings** as required by your environment.

   For a description of the tree items under **Advanced Settings**, see "**Advanced Settings and Default Advanced Settings**".

7. Click **Save** when you are done.

Debug Environment for Setup 2 (Integrated Server)

1. In the **T32Start** window, right-click **Configuration Tree**, point to **Add**, and then select **Configuration**.

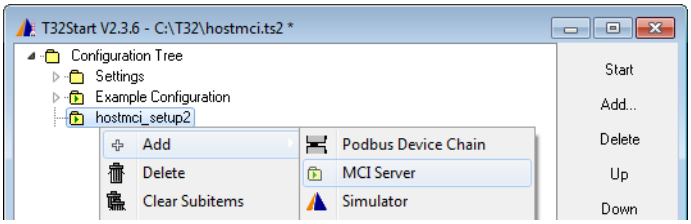


   A new tree item named **Configuration**<*number*> is displayed.

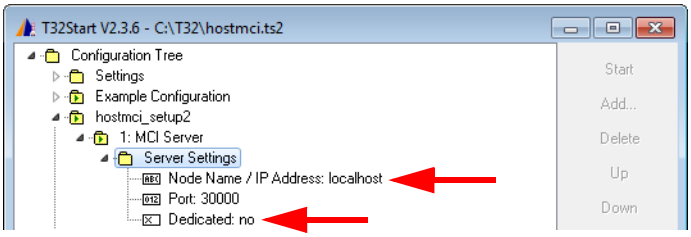
2. Press **F2** to rename the tree item to a meaningful name, for example, **hostmci_setup2**.

MCI Server and Server Settings (Setup 2)

3. Right-click the renamed tree item, point to **Add**, and then select **MCI Server**.



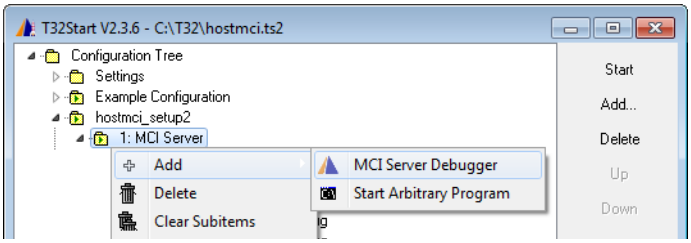
4. Navigate from **MCI Server** to **Server Settings** by clicking the little triangles next to the tree items.



5. Under **Server Settings**, make these settings:

   - **Node Name / IP Address**: Leave empty or type `localhost`
   - **Port**: Type any free port number
   - **Dedicated**: `no`

6. Click the **MCI Server** tree item, point to **Add**, and then select **MCI Server Debugger**.



7. Click the little triangle next to **MCI Server Debugger** to navigate to **Target**.

8. Right-click **Target**, and then select the target name you want from the **Target** drop-down list.

9. Repeat the steps in this section for each **MCI Server Debugger** required for your project.

This completes the steps for setup 2. The remaining steps depend on your project, see **Advanced Settings** below.

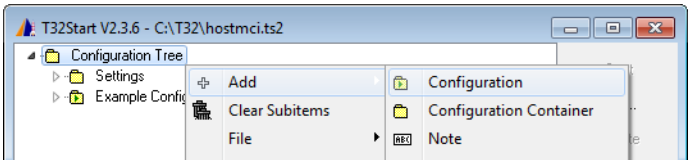
Advanced Settings (Setup 2)

10. For each **MCI Server Debugger**, configure the **Advanced Settings** as required for your project. For a description of the tree items under **Advanced Settings**, see "**Advanced Settings and Default Advanced Settings**".

11. Click **Save** when you are done.

Debug Environment for Setup 3 (Dedicated Server)

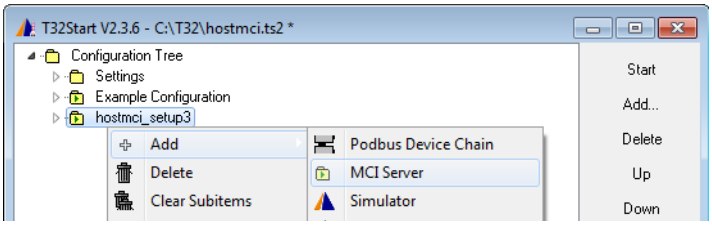
1. In the **T32Start** window, right-click **Configuration Tree**, point to **Add**, and then select **Configuration**.



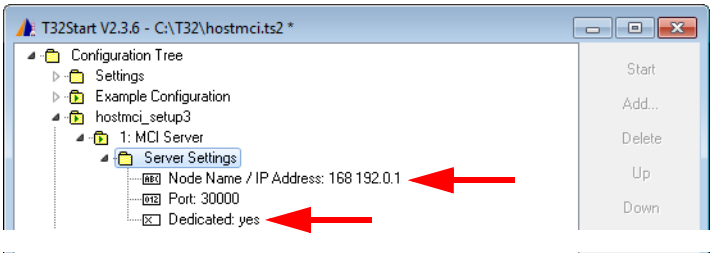
A tree item named **Configuration***<number>* is displayed.

2. Press **F2** to rename the tree item to a meaningful name, for example, **hostmci_setup3**.

3. Right-click the renamed tree item, point to **Add**, and then select **MCI Server**.



4. Navigate from **MCI Server** to **Server Settings** by clicking the little triangles next to the tree items.

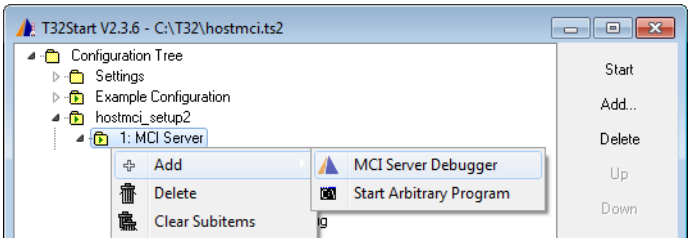


5. Under **Server Settings**, make these settings:

   - **Node Name / IP Address**: Type IP address of the workstation where t32mciserver runs
   - **Port**: Type any free port number
   - **Dedicated**: `yes`

MCI Server Debugger and Target (Setup 3)

6. Click the **MCI Server** tree item, point to **Add**, and then select **MCI Server Debugger**.



7. Click the little triangle next to **MCI Server Debugger** to navigate to **Target**.
8. Right-click **Target**, and then select the target name you want from the **Target** drop-down list.
9. Repeat the steps in this section for each **MCI Server Debugger** required for your project.

This completes the steps for setup 3. The remaining steps depend on your project, see **Advanced Settings** below.

10. For each **MCI Server Debugger**, configure the **Advanced Settings** as required for your project.

    For a description of the tree items under **Advanced Settings**, see "**Advanced Settings and Default Advanced Settings**".

Click **Save** when you are done.

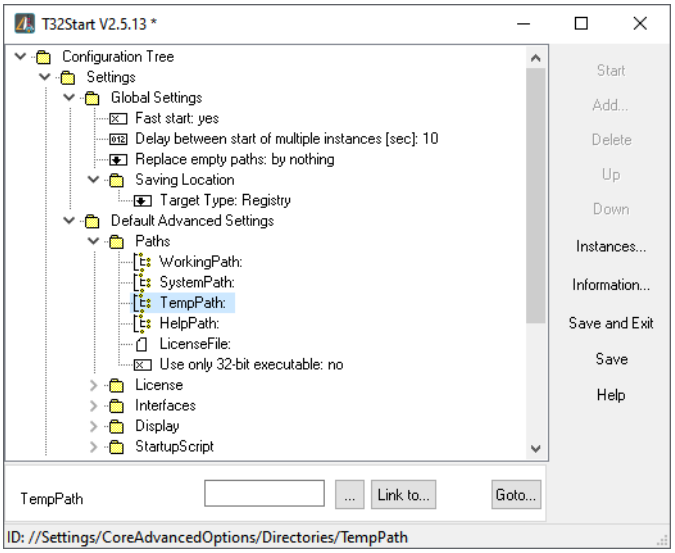
References to Tree Items

Tree items can contain references to other tree items. The reference between a source item and a destination item can be a relative or an absolute path.
For example, the reference `%//Settings/CoreAdvancedOptions/Directories/SystemPath%` points to the tree item **Default Advanced Settings** > **Paths** > **SystemPath**. The substring `//` is interpreted as root of the **Configuration Tree**. Delimiters are marked as single slash, and two dots return to the top item.

To create a reference to another tree item in T32Start:

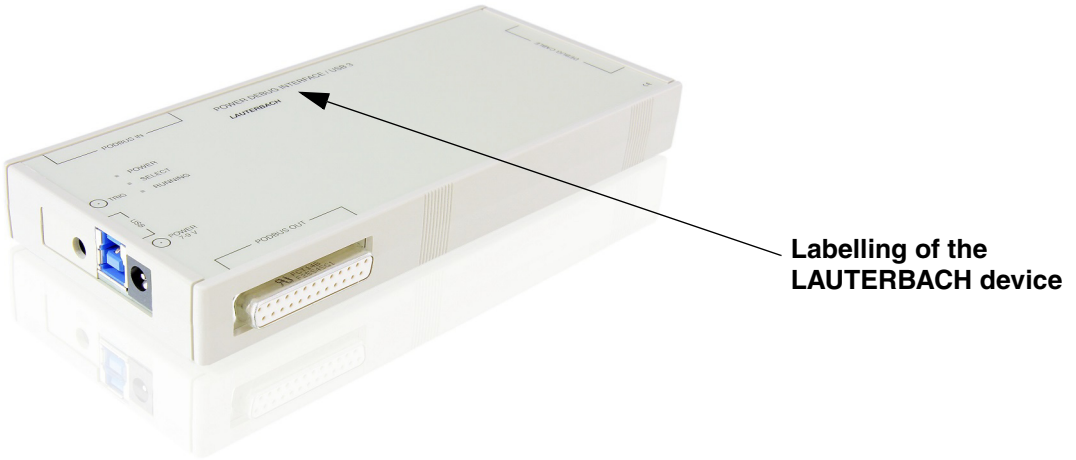
1. Select the destination item you want.
2. Click the **Link to** button to browse the entire tree.
3. Navigate to the source item you want.
4. Click **OK** to create the reference.

The absolute path of any item is displayed in the status line of T32Start. To copy the absolute path, right-click the status line, and then select **Copy ID**.



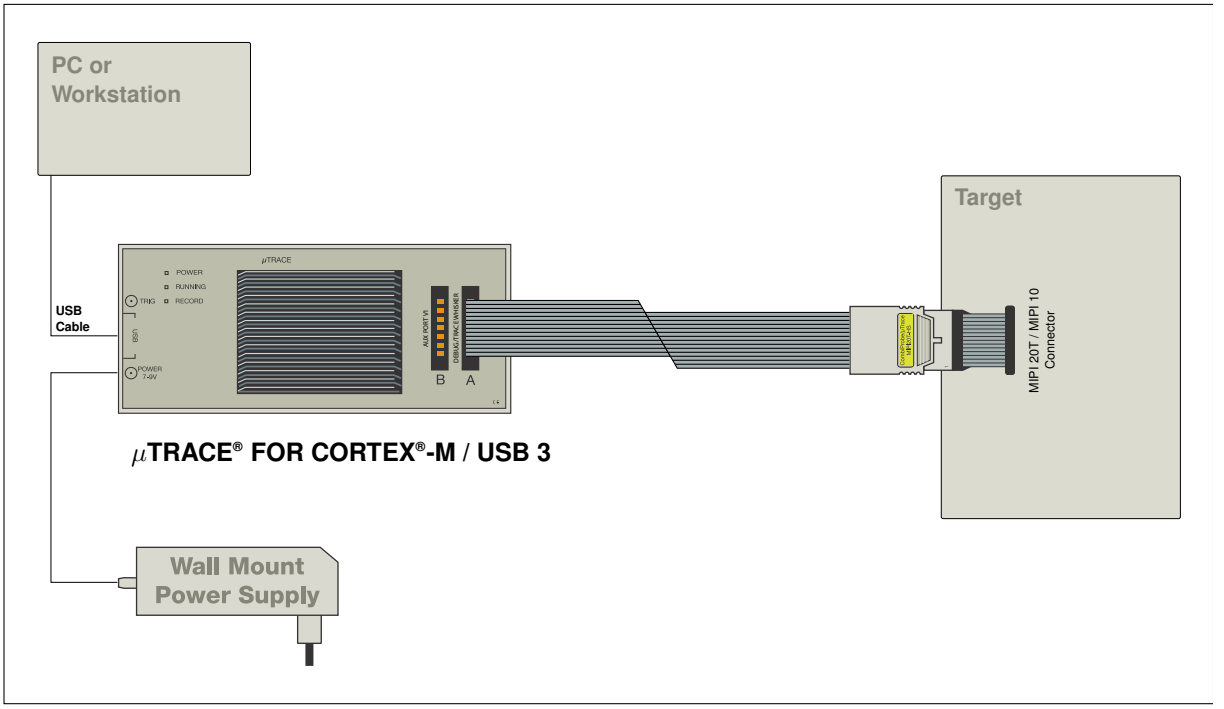
Hardware-based TRACE32 Tools

The correct name of your LAUTERBACH device can be read from the labelling.

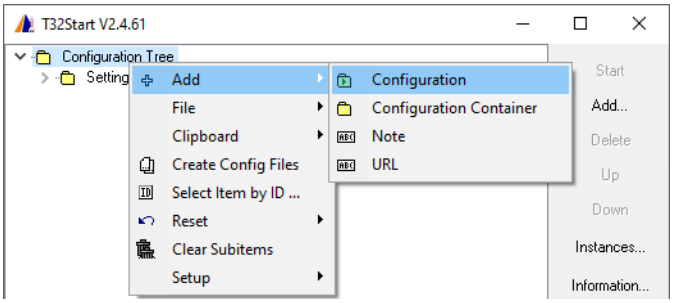


Single Core Debugging and Tracing (MicroTrace)

The following example describes how to start a TRACE32 instance to debug and trace a Cortex-M via a µTrace (MicroTrace).

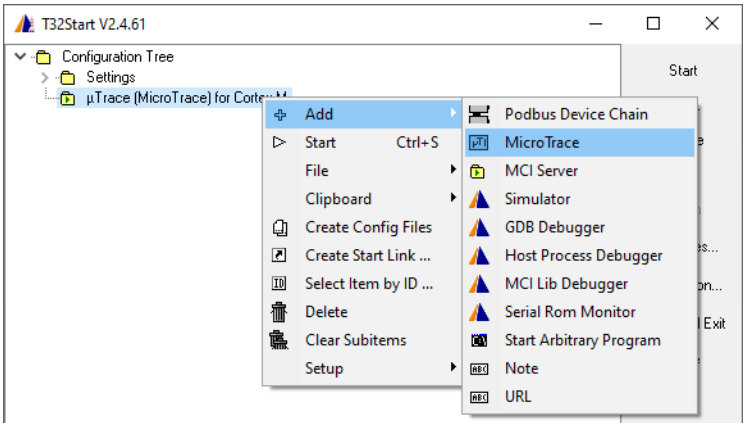


1. Add a new **Configuration**. Use a **right-click** to open the context menu.

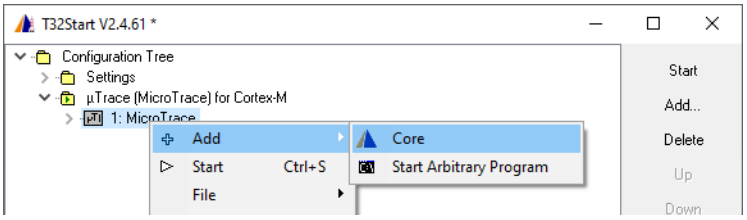


2. Rename the **Configuration** by using the function key **F2**.

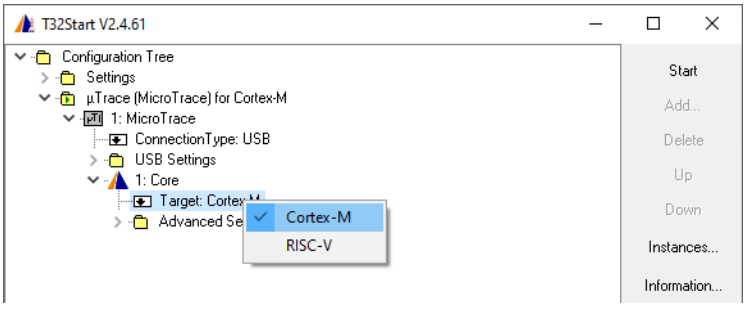
3. Add a **MicroTrace** to the configuration.



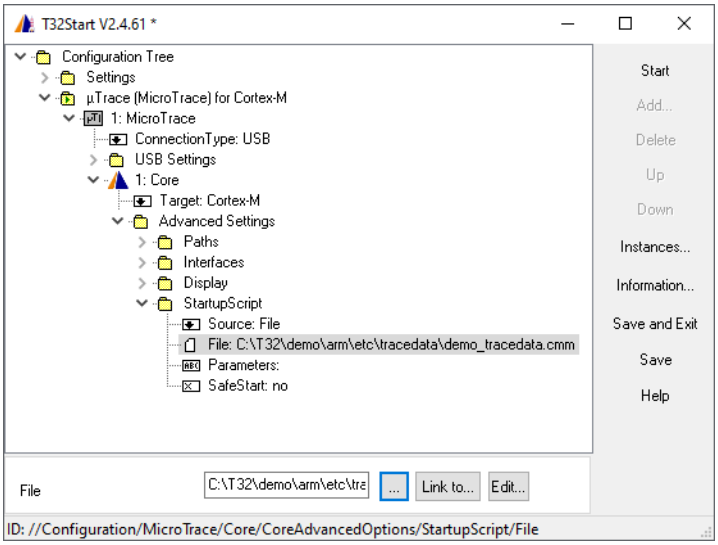
4. Assign a Core to the **MicroTrace**.



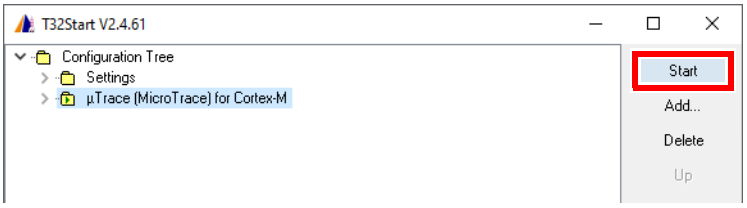
5. Open the **Core** branch to ospecify the target core.



6. Optionally define a start-up script under **Advanced Settings > Startup Script**



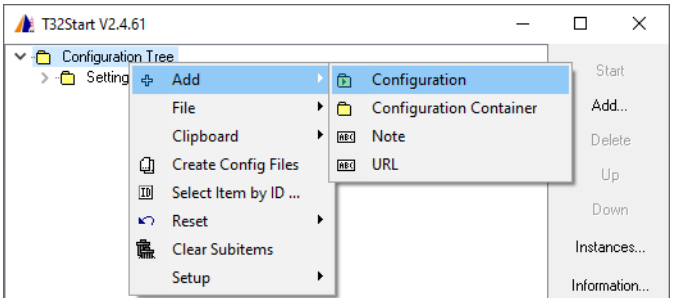
7. Close the complete configuration tree and start the TRACE32 instance.



Multicore Debugging and Tracing (MicroTrace)

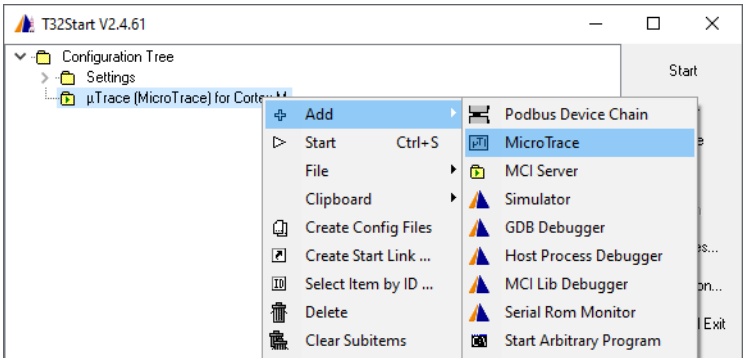
The following example describes how to start a TRACE32 instances to debug and trace a multiple core Cortex-M via a µTrace (MicroTrace).

1. Add a new **Configuration**. Use a **right-click** to open the context menu.

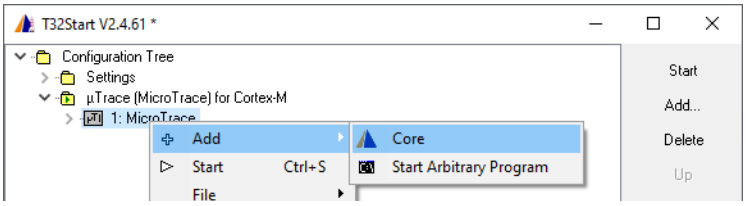


2. Rename the **Configuration** by using the function key **F2**.

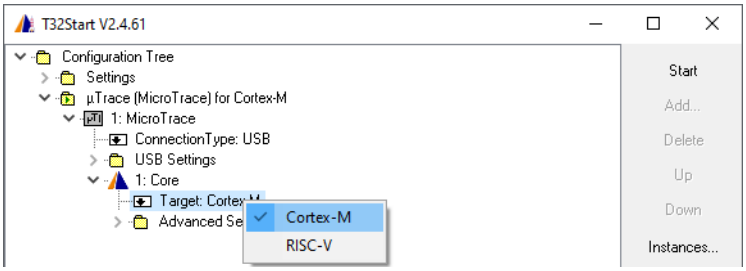
3. Add a **MicroTrace** to the Configuration.



4. Assign the Core to the **MicroTrace**.

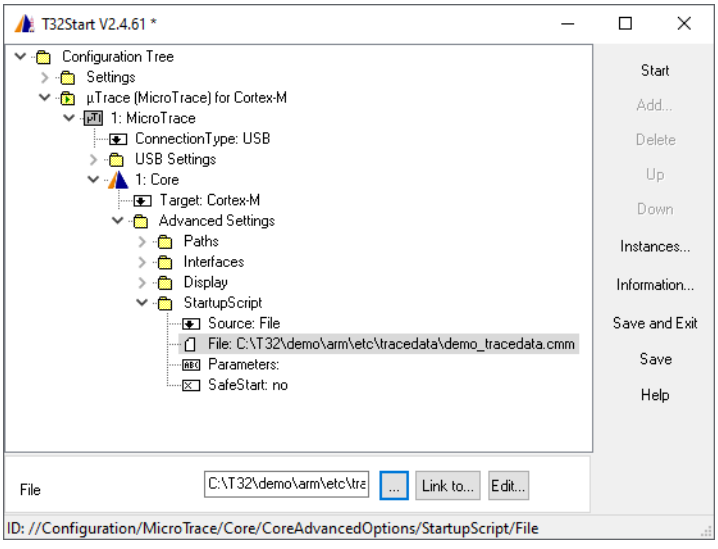


Open the **Core** branch to specify the target core.

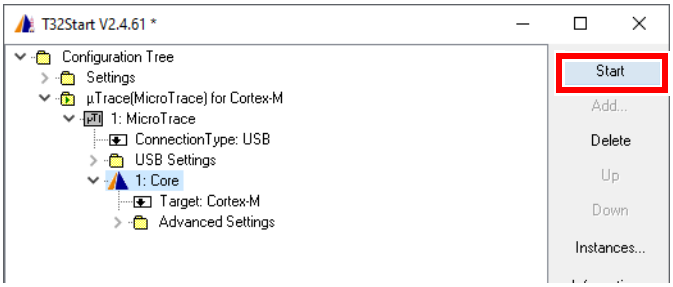


Now the basic configuration is done.

5. Optionally define a start-up script under **Advanced Settings > Startup Script**



6. Start the first TRACE32 instance.



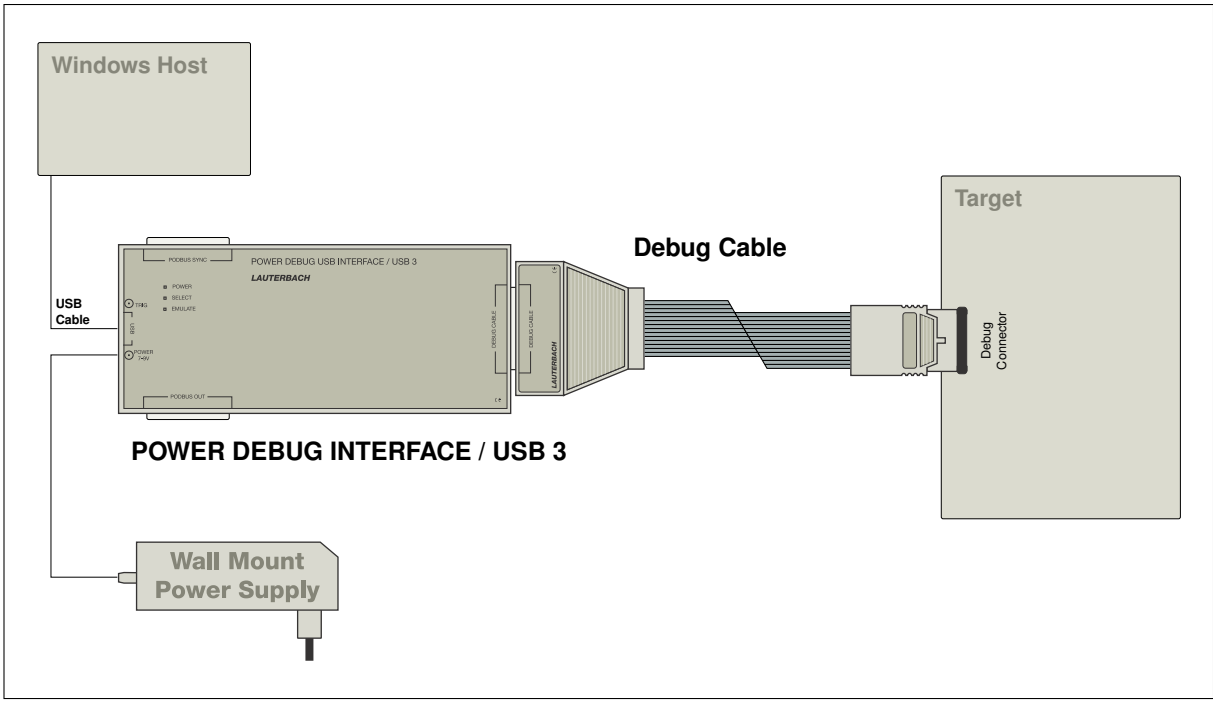
7. Start the second TRACE32 instance from the first instance using the command **TargetSystem.NewInstance**

```
TargetSystem.NewInstance mySecondInstance /ARCHitecture Arm
```

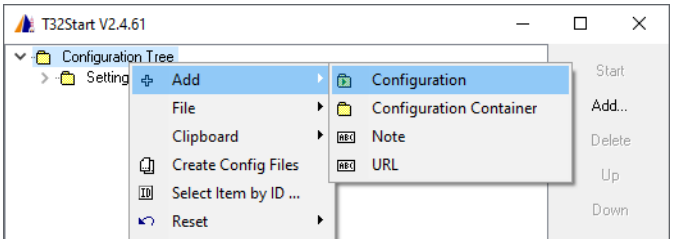
As an alternative to using the **TargetSystem.NewInstance** command, you can assign a second core to the **MicroTrace** and set Arm as target option. The advantage of using the command **TargetSystem.NewInstance** is that the linkage of the TRACE32 instances is automatically set up and does not need to be configured manually in T32Start.

Single Core Debugging (USB 3)

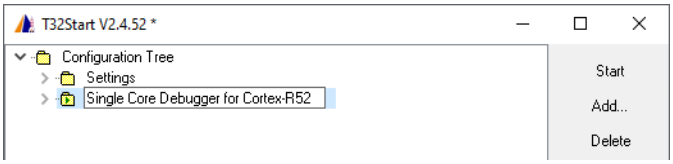
The following example describes how to start a TRACE32 instance to debug a Cortex-R (Arm architecture) via a POWER DEBUG USB INTERFACE / USB 3. The configuration steps are the same if a Debug Cable or a CombiProbe is connected to the PowerDebug module.



1. Add a new **Configuration**. Use a **right-click** to open the context menu.

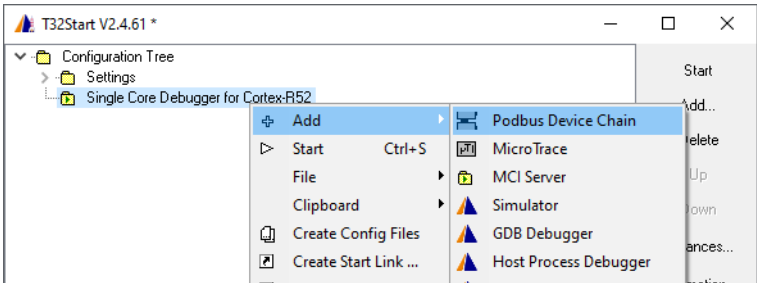


2. Rename the **Configuration** by using the function key **F2**.

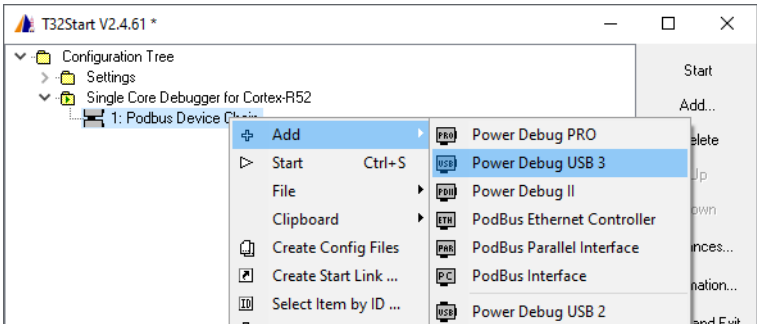


3. Create a **Podbus Device Chain**.

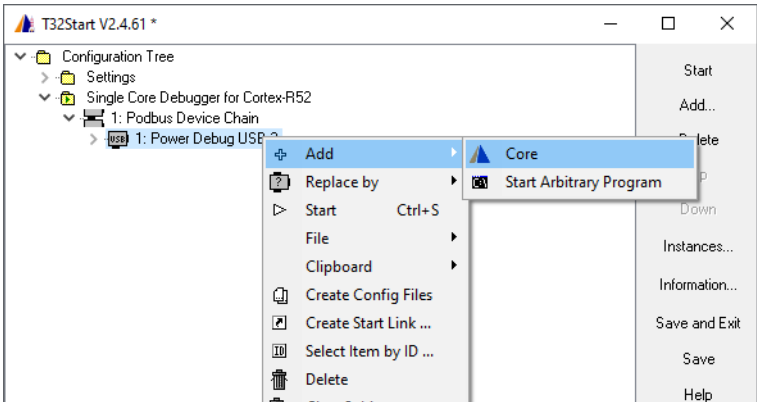
   The Podbus Device Chain defines which LAUTERBACH devices are used for debugging.



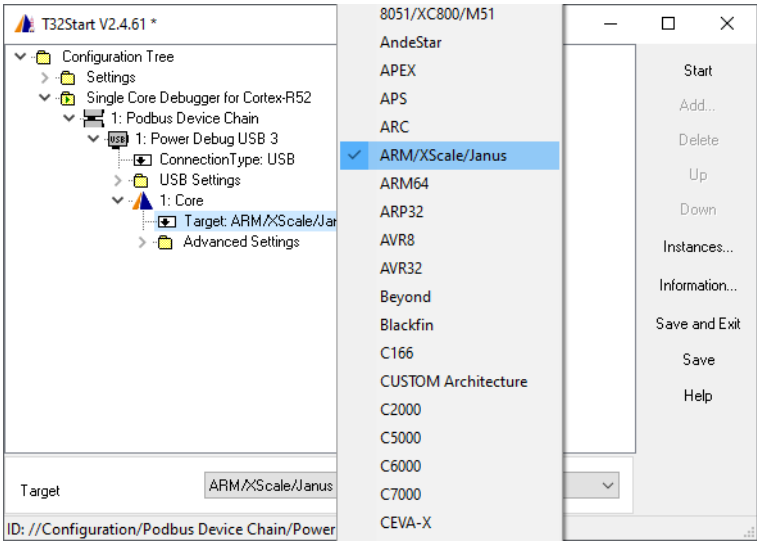
4. Add the Power Debug USB3 to the **Podbus Device Chain**.



5. Assign a Core to the **Power Debug USB 3**.

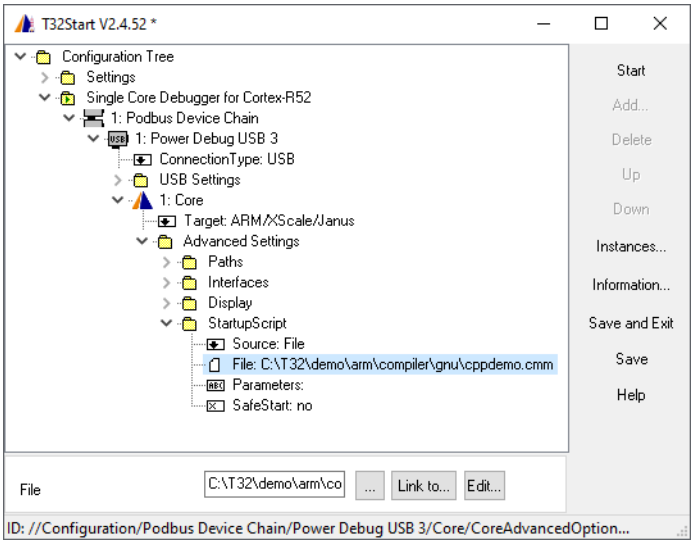


6. Open the **Core** branch to specify Arm as architecture for debugging.

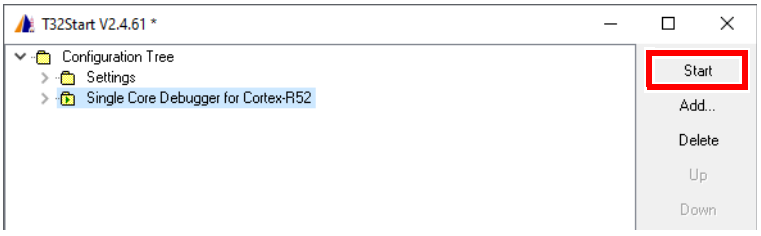


Now the basic configuration is done.

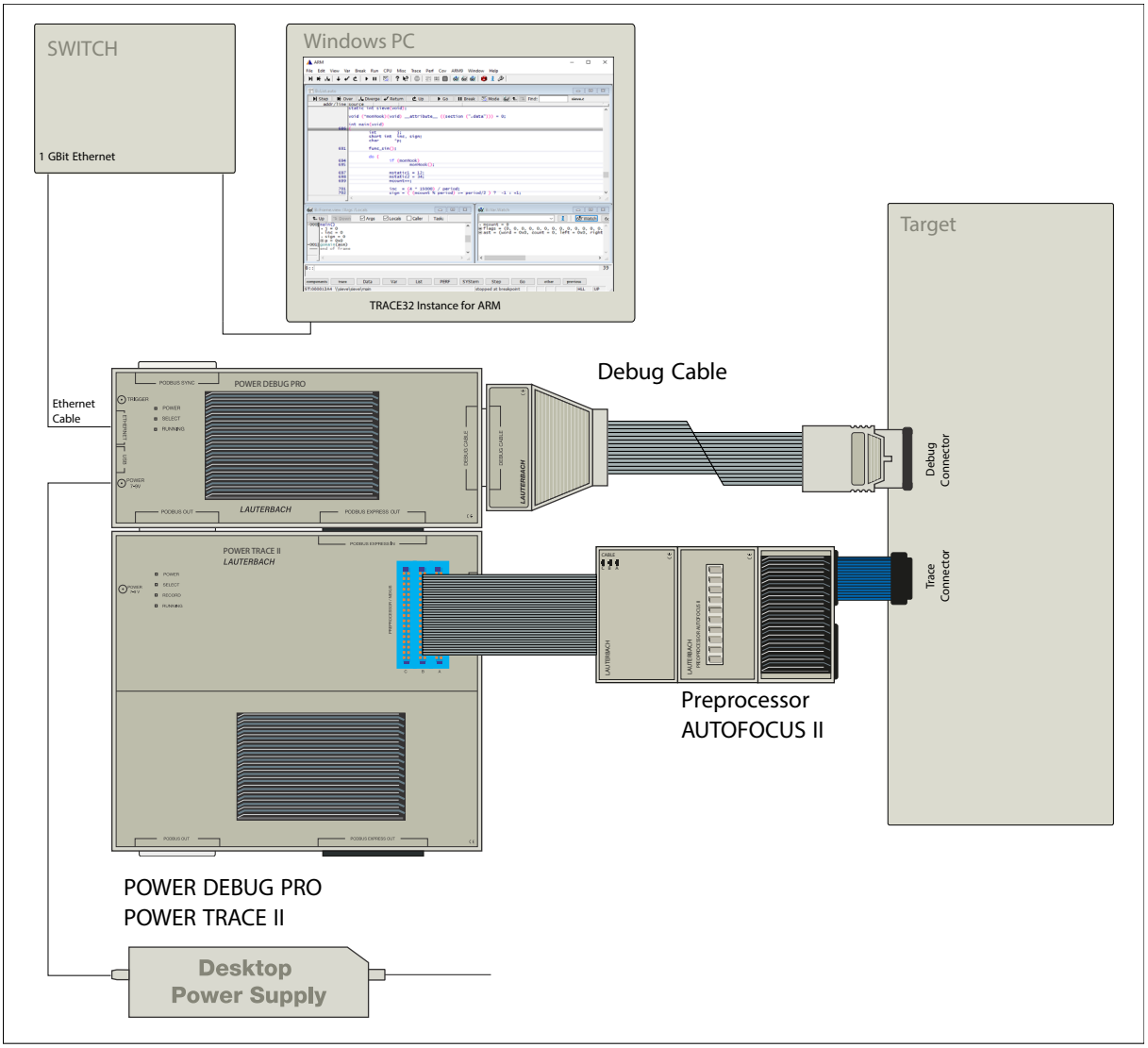
7. Optionally define a start-up script under **Advanced Settings > Startup Script**



8. Close the complete configuration tree and start the TRACE32 instance to debug the Arm target.

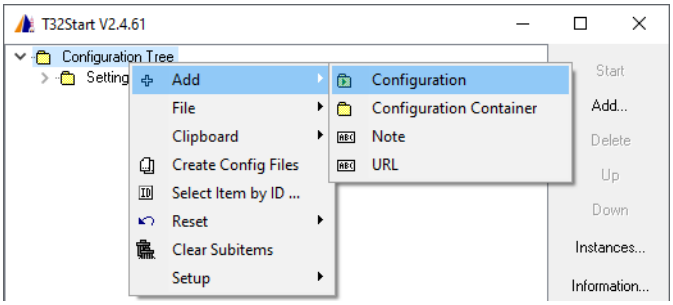


The following example describes how to start a TRACE32 instance to debug and trace a Cortex-R core (Arm architecture) via a PowerDebug PRO and a PowerTrace II module.

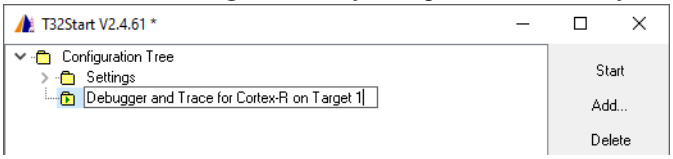


Power supply has to be connected to POWERDEBUG PRO only. Power (voltage) is passed on by POWERDEBUG PRO to POWERTRACE II. The power jack at the POWERTRACE II is reserved for future use only.

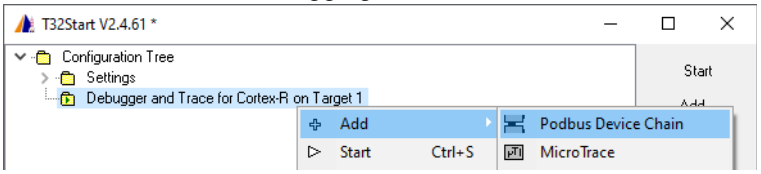
1. Add a new **Configuration**. Use a **right-click** to open the context menu.



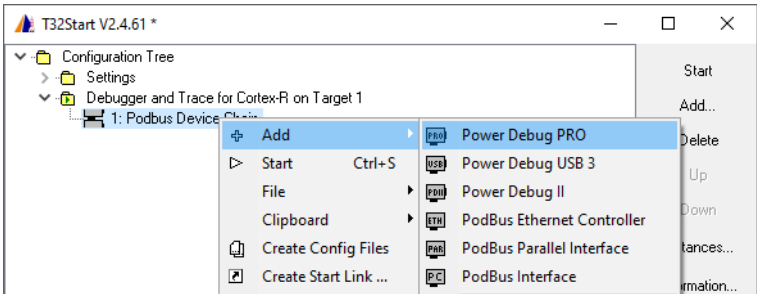
2. Rename the **Configuration** by using the function key **F2**.



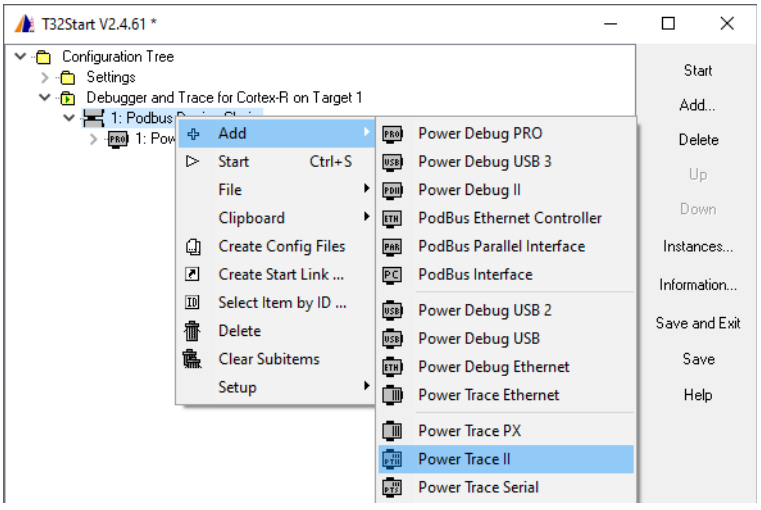
3. Create a **Podbus Device Chain**. The Podbus Device Chain defines which LAUTERBACH devices are used for debugging.



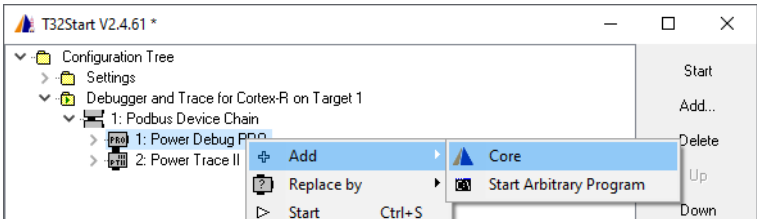
4. Add the POWER DEBUG PRO to the **Podbus Device Chain**.



5. Add the POWER TRACE II to the **Podbus Device Chain**.

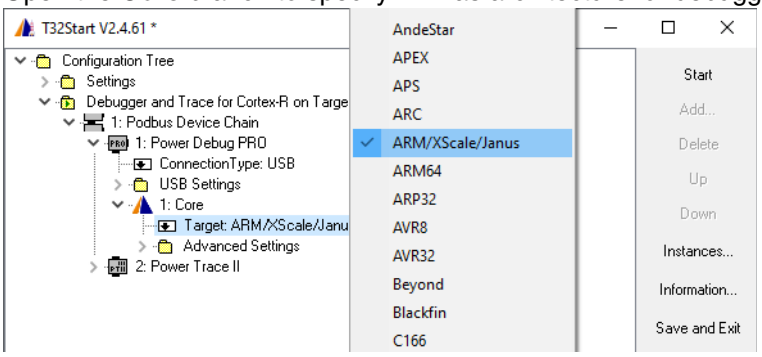


6. Assign a Core to the **Power Debug PRO**. The reason for this assignment is that the TRACE32 driver software/JTAG handler runs on the Power Debug PRO module.



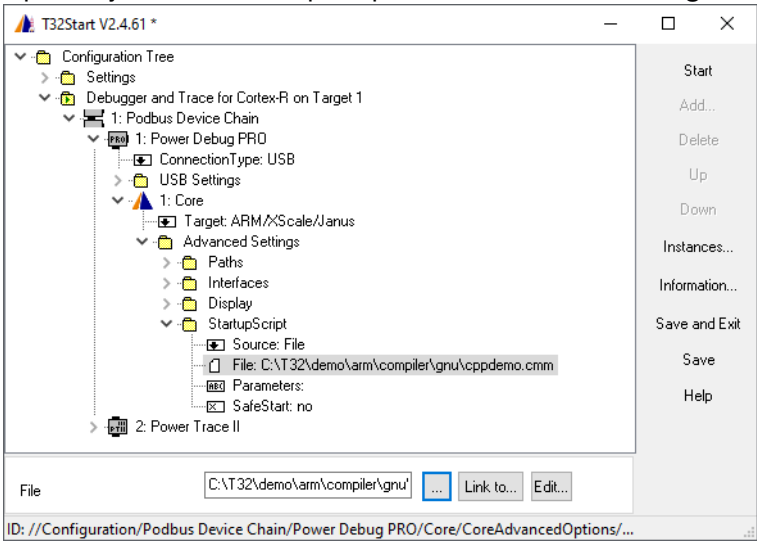
Because two devices are used for debugging, both of them have to be included in the **Podbus Device Chain**.

7. Open the **Core** branch to specify Arm as architecture for debugging.

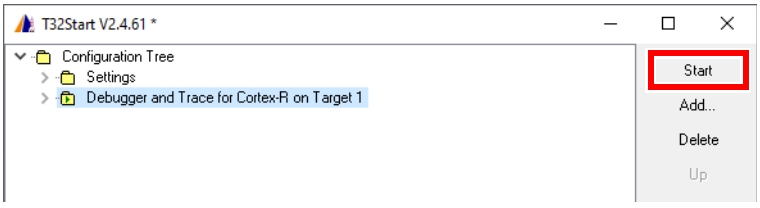


Now the basic configuration of POWER DEBUG PRO is done. No extra configurations are needed for PowerTrace II.

8. Optionally define a start-up script under **Advanced Settings > Startup Script**



9. Close the complete configuration tree and start the TRACE32 instance to debug the Arm target.



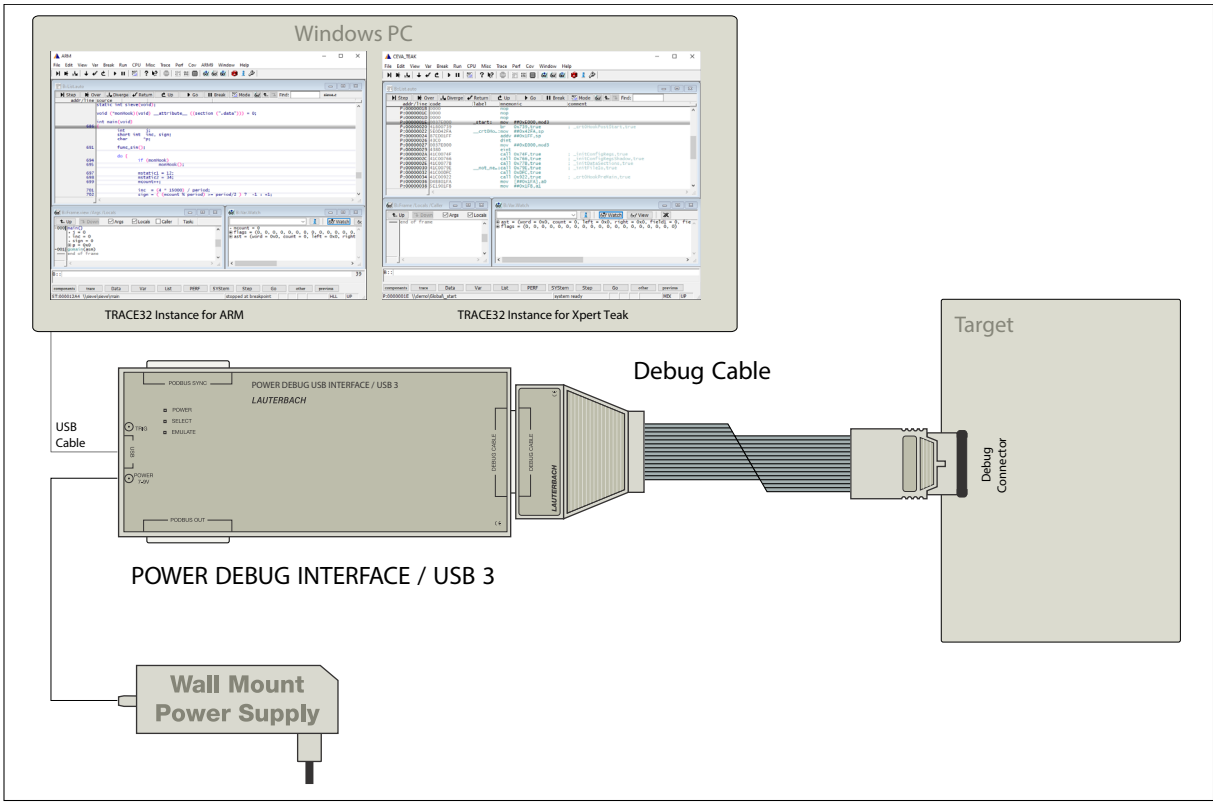
Multicore Debugging (heterogenous AMP)

The term **Multicore Debugging** is used if there are multiple cores on one chip which use a joint JTAG interface for debugging.

To debug cores of different architectures (heterogenous AMP debugging), a separate TRACE32 instance must be opened for each core. The following example describes how to start two TRACE32 instances

- one to debug a Cortex-R
- one to debug an Xpert Teak

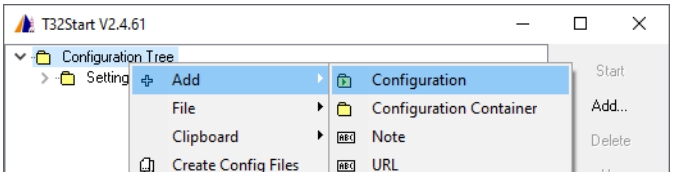
On a multicore chip. Both cores are debugged via a joint JTAG interface by using a POWER DEBUG INTERFACE / USB 3.



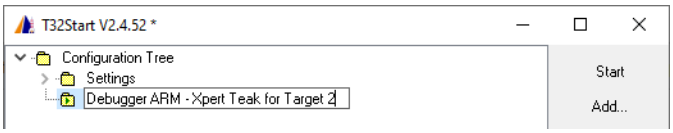
If you want to debug two identical cores, you can also do this with two TRACE32 instances (homogeneous AMP).

If you have also connected a PowerTrace and both cores export trace information via a joint trace port, you can also configure multicore debugging and tracing as described.

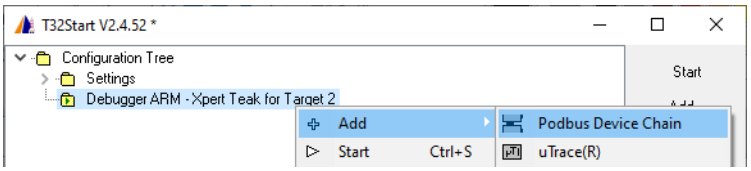
1. Add a new **Configuration**.



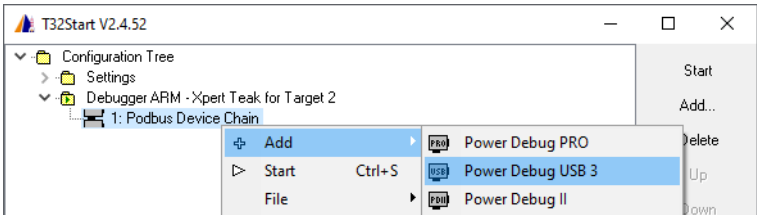
2. Rename the **Configuration** by using the function key **F2**.



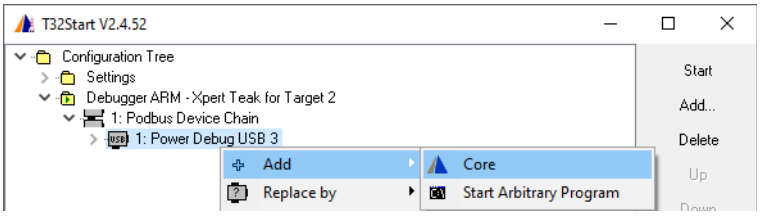
3. Create a **Podbus Device Chain**. The Podbus Device Chain defines which LAUTERBACH devices are used for debugging.



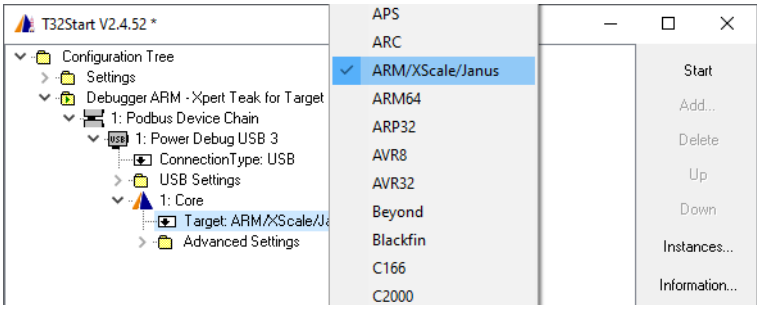
4. Add the Power Debug USB3 to the **Podbus Device Chain**.



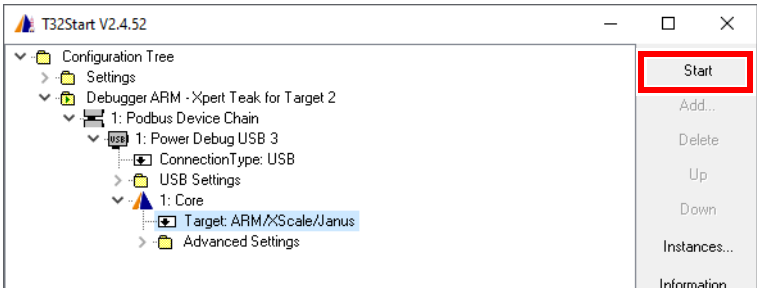
5. Assign the first Core to the **Power Debug USB 3**.



6. Open the **Core** branch to specify Arm as architecture for debugging.



7. Start the first TRACE32 instance



8. Start the second TRACE32 instance from the first instance using the command **TargetSystem.NewInstance**

```
TargetSystem.NewInstance mySecondInstance /ARCHitecture TEAK
```

As an alternative to using the **TargetSystem.NewInstance** command, you can assign a second core to the **Power Debug USB 3** and set Teak/TeakLite/OAK as target option. The advantage of using the command **TargetSystem.NewInstance** is that the linkage of the TRACE32 instances is automatically set up and does not need to be configured manually in T32Start.

Linkage Between TRACE32 Instances

A linkage between the TRACE32 instances is recommended for the following purposes:

- **Start/stop synchronization**

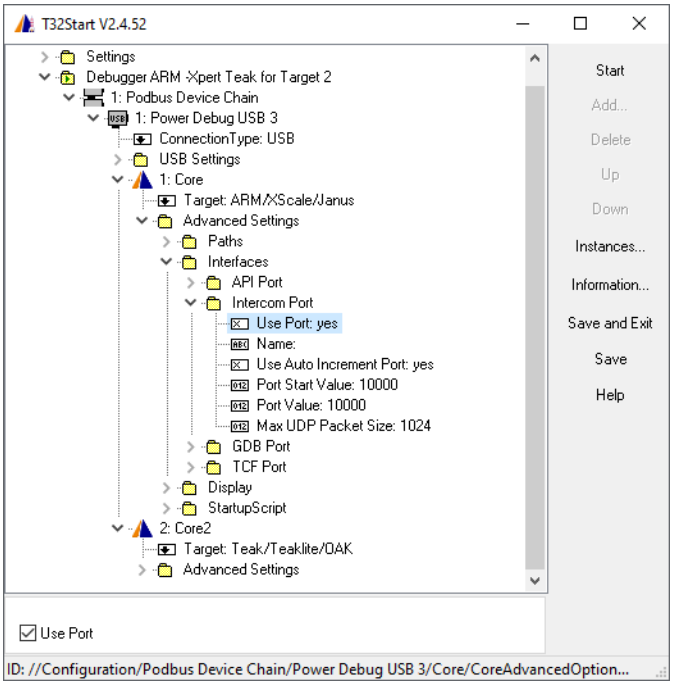
  The start/stop synchronization has to be prepared before the start of the TRACE32 instances by configuring the **InterCom Port** for each core. This in independent of the implementation of the start/stop synchronization.

- **Communication between the TRACE32 instances**

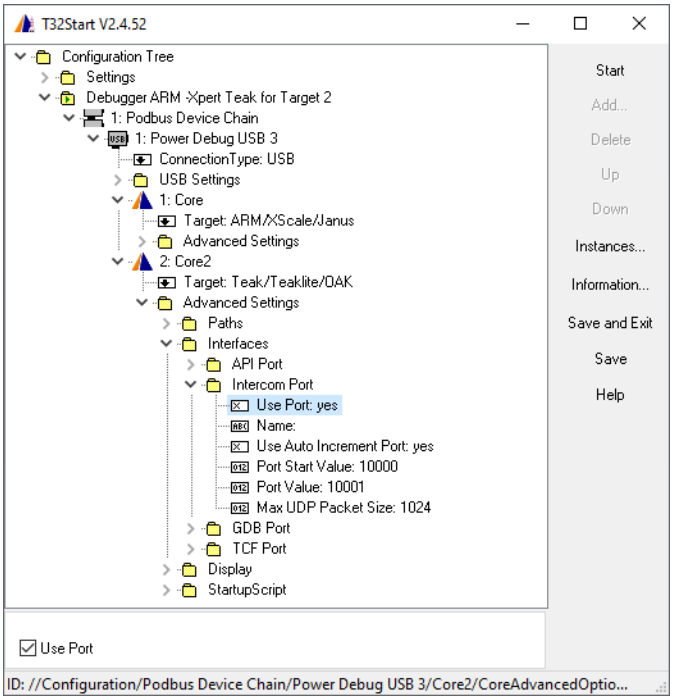
  The individual TRACE32 instances can communicate via the **InterCom** command. This communication has to be prepared before the start of the TRACE32 instances by configuring the **InterCom Port** for each core.

The following settings are only necessary if two cores are assigned to the debug module in T32Start.

1. Open the **Advanced Settings** branch for the first **Core** to configure the **InterCom Port** and set **Use Port** to yes. The default **Port Value** is 10000.



2. Open the **Advanced Settings** branch for **Core 2** to configure the **InterCom Port** and set **Use Port** to yes. Since **Use Auto Increment Port** is set to yes the **Port Value** for the second core is 10001.



The term **Multiprocessor Debugging** is used if there are multiple micro-controllers on the target. Each micro-controller is debugged via its own JTAG interface.

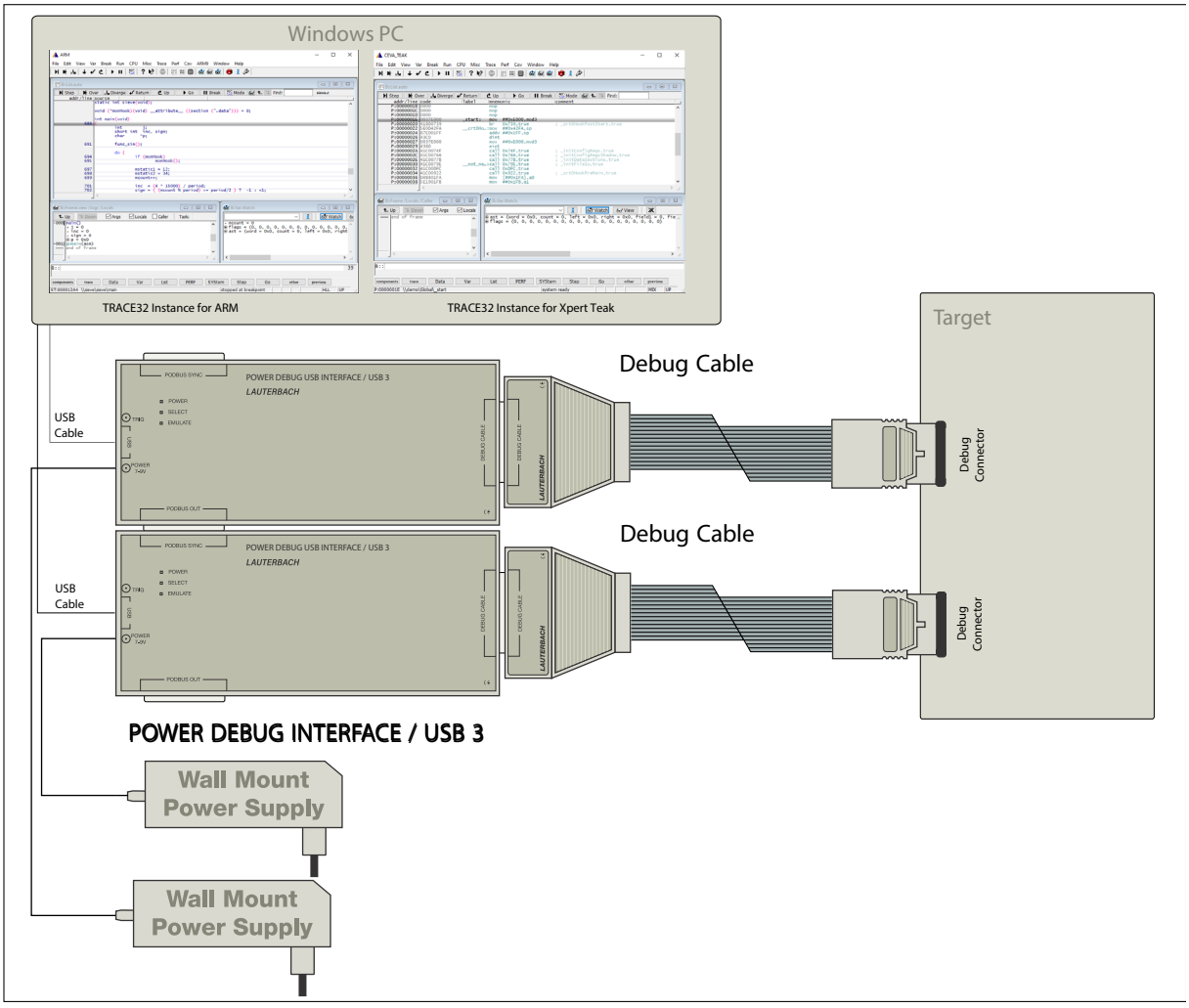
The following example describes how to start two TRACE32 instances:

- To debug an Arm via a POWER DEBUG INTERFACE / USB3
- To debug an Xpert Teak via a second POWER DEBUG INTERFACE / USB3

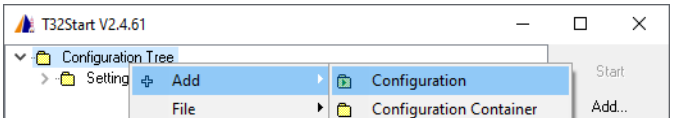
The USB 3 module has a PODBUS SYNC connector. This type of module must be connected directly to the host computer. In the example below, both PowerDebug USB 3 modules are connected to the host via USB.

Nevertheless, it makes sense to plug both modules together. This has the following advantages:

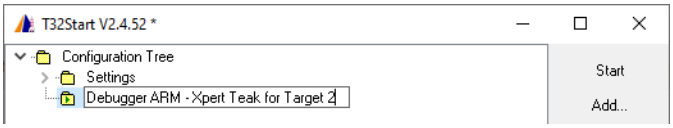
- The debug modules share a PODBUS and can trigger each other as described in **"Interaction Between Independent PODBUS Devices"** in General Commands Reference Guide T, page 517 (general_ref_t.pdf).
- The debug modules have a common time base. This becomes relevant when using TRACE32 functions that assign a timestamp, for example the **SNOOPer** command.



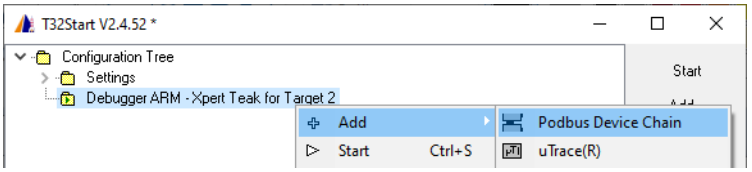
1. Add a new **Configuration**.



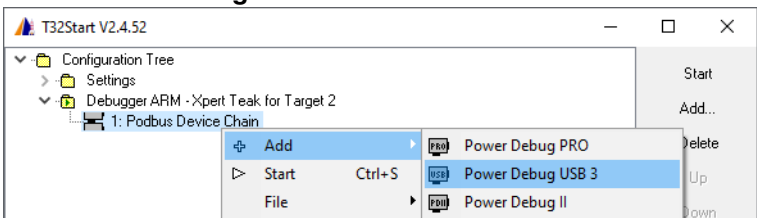
2. Rename the **Configuration** by using the function key **F2**.



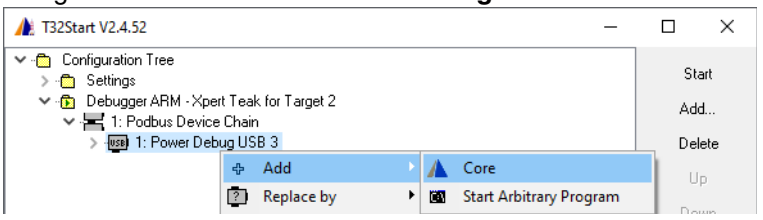
3. Create a **Podbus Device Chain**.



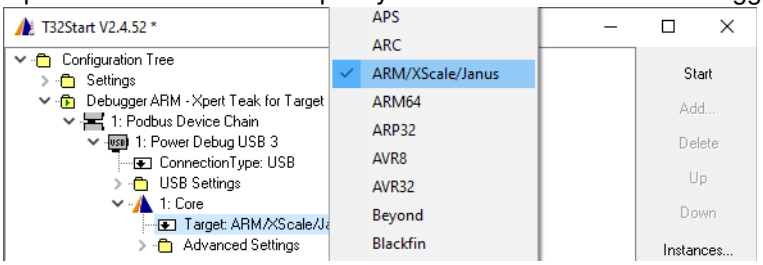
4. Add a **Power Debug USB3** to the **Podbus Device Chain**.



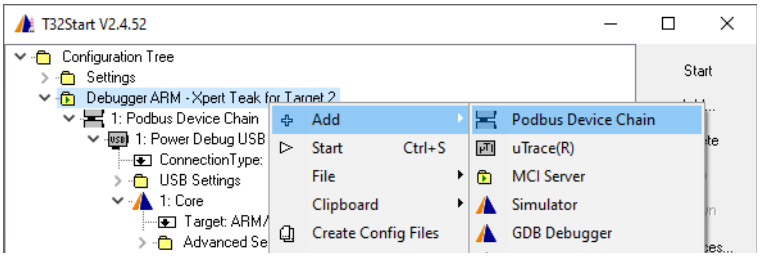
5. Assign the first Core to the **Power Debug USB3**.



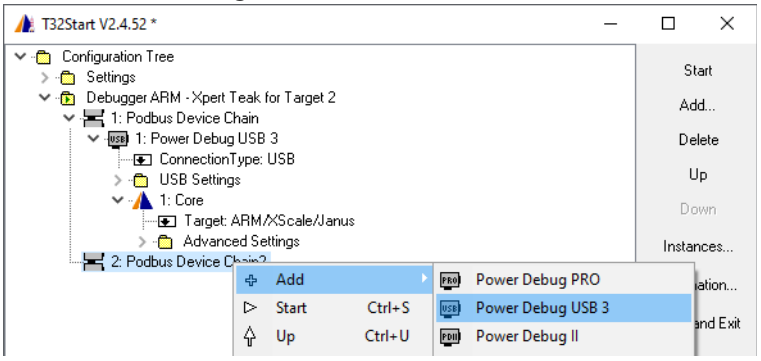
6. Open the **Core** branch to specify Arm as architecture for debugging.



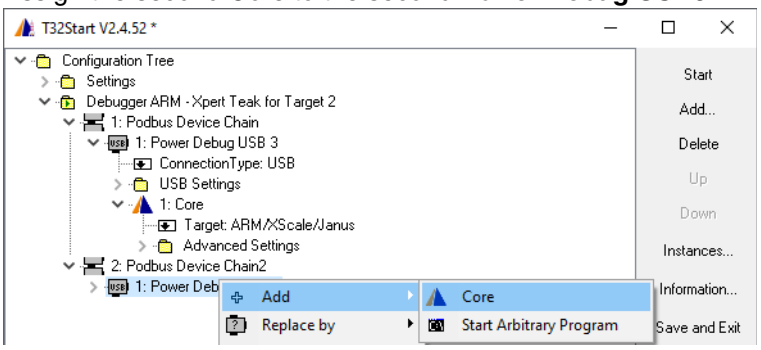
7. Create a second **Podbus Device Chain**.



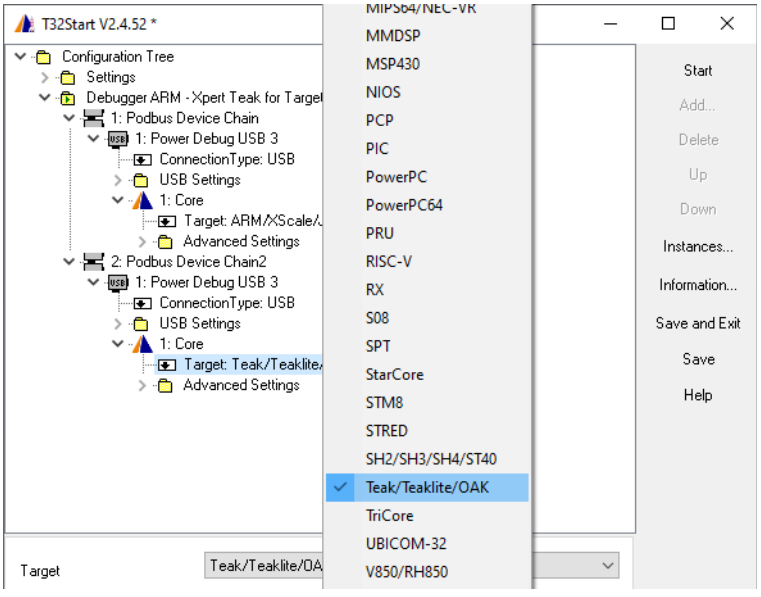
8. Add a **Power Debug USB3** to the second **Podbus Device Chain**.



9. Assign the second Core to the second **Power Debug USB3**.



Open the **Core** branch to specify Teak as architecture for debugging.



Now the basic setup to start two TRACE32 instances to debug an Arm and an XpertTeak two separate JTAG connectors is done.

10. Link the two TRACE32 instances via the **InterCom** command as described in **"Linkage Between TRACE32 Instances"**, page 49.

Instruction Set Simulator

The following example describes how to set up T32Start in order to start a TRACE32 Set Simulator.

A TRACE32 Instruction Set Simulator can be licensed in different ways:

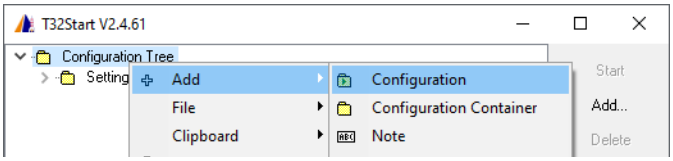
1. No license is required for interactive debugging in the TRACE32 PowerView GUI, loading RAM dump and analyzing trace data previously saved with the **Trace.SAVE** command.
2. The TRACE32 Instruction Set Simulator allows to perform 50 script commands/API operations after the first "**single-step**" or "**Go**". If you want to perform further script commands/API operations a TRACE32 Simulator License is needed.

There are two ways to obtain a simulator license:

1. You have a debug module with the appropriate license and a valid maintenance contract that you do not need right now, then you can use this debugger as license source.
2. You have access to a license server that provides you with a suitable license.

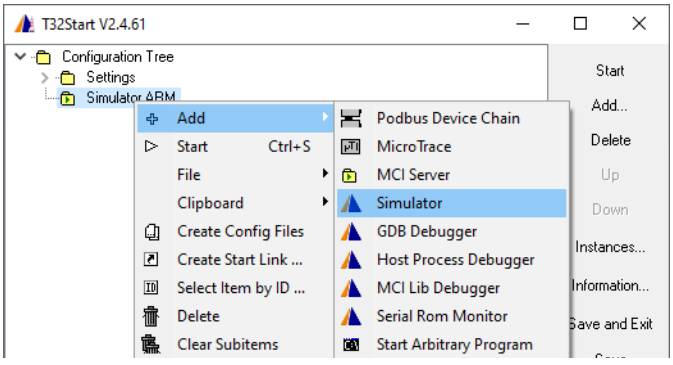
Configuration Steps

1. Add a new **Configuration**. Use a **right-click** to open the context menu.

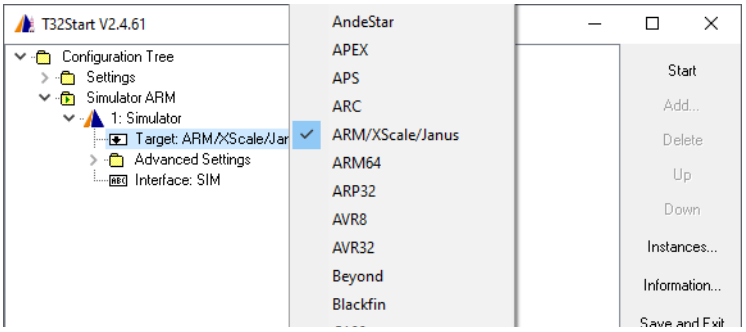


2. Rename the **Configuration** by using the function key **F2**.

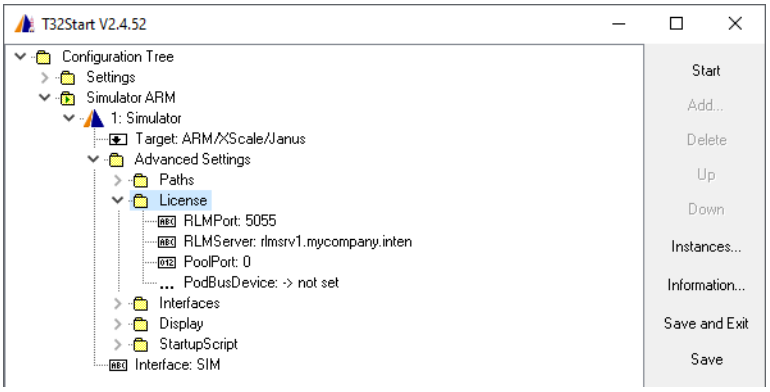
3. Add a **Simulator** to the Configuration.



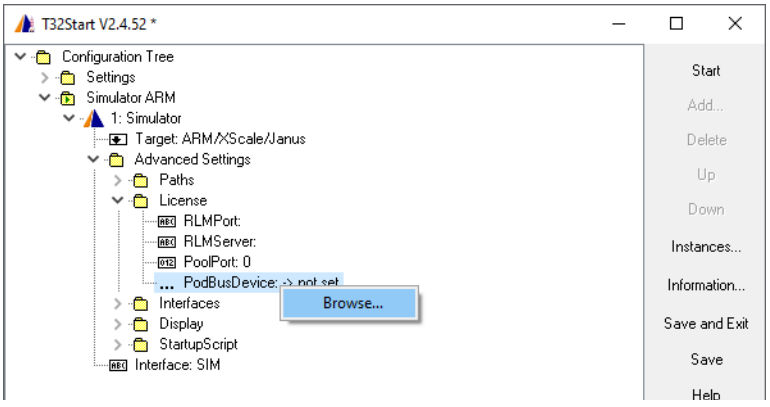
4. Set up the target architecture.



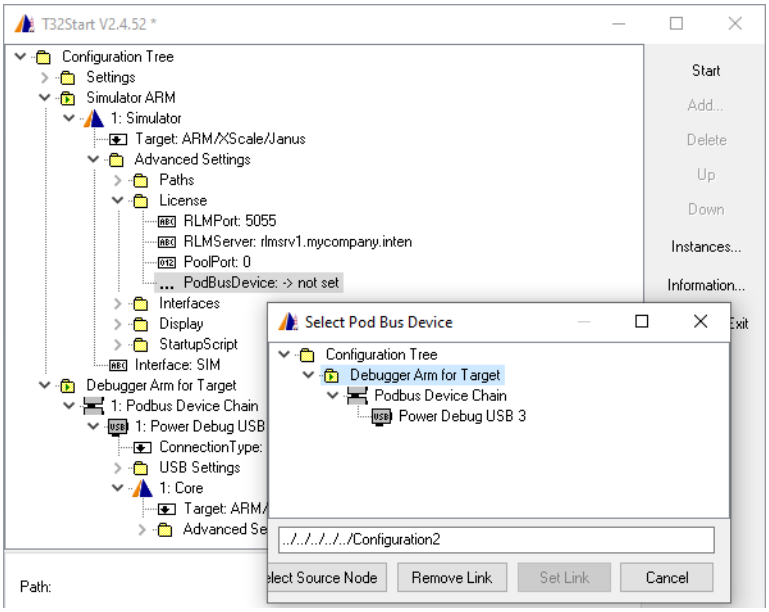
5. If you have a floating license for the Simulator, configure the RLM settings of the license under **Advanced Settings > License** by setting the RLM Port and Server.



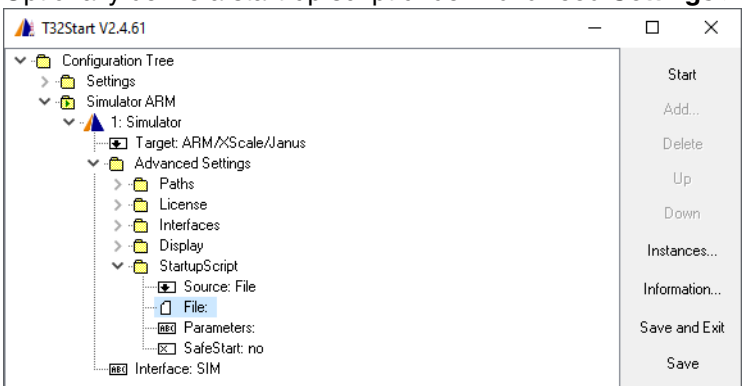
If you have a hardware-based TRACE32 debugger, you can also use it to license the TRACE32 Instruction Set Simulator. Do in this case a right-mouse click on **...PodBusDevice** then select **Browse**.



Select then the **Pod Bus Device** and click **Select Source Node**.



6. Optionally define a start-up script under **Advanced Settings > Startup Script**



7. Close the complete configuration tree and start the TRACE32 instance.

Command Line Arguments

The Syntax of the command line parameters of T32Start is:

```
T32Start.exe [-QUIT] [[-RUNCFGFILE] <config_file>] [-RUNITEM <id>]
```

Command Line Option	Description
-QUIT	Closes T32Start after any action given by the other options.
-RUNCFGFILE *<config_file>*	Start the configuration saved in the file *<config_file>*. Make sure, that *<config_file>* was derived from a tree item of the type configuration.
<config_file>	Adds the configuration saved in the file *<config_file>* to the **Configuration Tree**. Make sure, that *<config_file>* was derived from a tree item of the configuration type. There are two scenarios to use this parameter. Both can get active when t32start is started by the registred .ts2 file extension by the Windows Explorer. • The information in the file is appended to the current **Configuration Tree** when the config file contains a **Configuration**. • The <config_file> is used temporarly as current **Configuration Tree** when the config file contains a complete backup.
-RUNITEM *<id>*	Starts a configuration or other startable items, that are already part of the configuration tree. The parameter *<id>* is the absolute logical path to the tree item e.g. //Configuration1. This is the recommended way to start configurations remotely.

Problem / Error Message	Solution
Directory not found / File not found	The consistency between the TRACE32 installation and the *Advanced Settings/Paths* is lost. Check your installation and the *Path* settings.
Loaded file is not of type "Configuration"	The command line parameter *<config_file>* points to a file that was not saved from a tree item of type **Configuration**.
Invalid ID path	The command line parameter *<id>* was not found or references to an item that is not startable. Copy the right ID of a startable item with the context menu at the status line.
Could not write to file	The **Configuration** tree could not be saved to the file. Specify a writable file under *Settings/Global Settings/Saving Location/File.*
string reference contains itself	File paths or directory path can refer to other tree items in order to take their value. Circular references are not allowed. Correct the references.
Cannot find configuration path	A reference could not be resolved. Correct the references.
Cannot find env variable	The environment variable could not be found. Create an environment variable with that name and restart T32Start or correct the references.
Cannot find executable	Probably the wrong *Target* is selected in a TRACE32 instance item (Core, Simulator, …). The executable for that target type is not installed.
No T32 PowerView GUI appears when *Start* is pressed without any error message	Add TRACE32 instance tree items to the sub tree of the startable item.