LAUTERBACH
DEVELOPMENT TOOLS

Debugging via Infineon DAS Server

MANUAL

Debugging via Infineon DAS Server

Debugging via Infineon DAS Server

Version 06-Jun-2024

Introduction

This document describes the TRACE32 support for the Infineon DAS server.

The intended use case is to use TRACE32 together with an emulation system connected to the DAS server.

NOTE:	TRACE32 requires an installation of the DAS server which is available from Infineon. Please refer to the Infineon homepage for system requirements of the DAS server.

Related Documents

- **"T32Start"** (app_t32start.pdf): The T32Start application assists you in setting up multicore / multiprocessor debug environments, and software-only debug environments. T32Start is only available for Windows.

  For more information about software-only debug environments, please refer to: **"Software-only Debugging (Host MCI)"** (app_t32start.pdf).
- **"TriCore Debugger and Trace"** (debugger_tricore.pdf)
- **"GTM Debugger and Trace"** (debugger_gtm.pdf)

Contacting Support

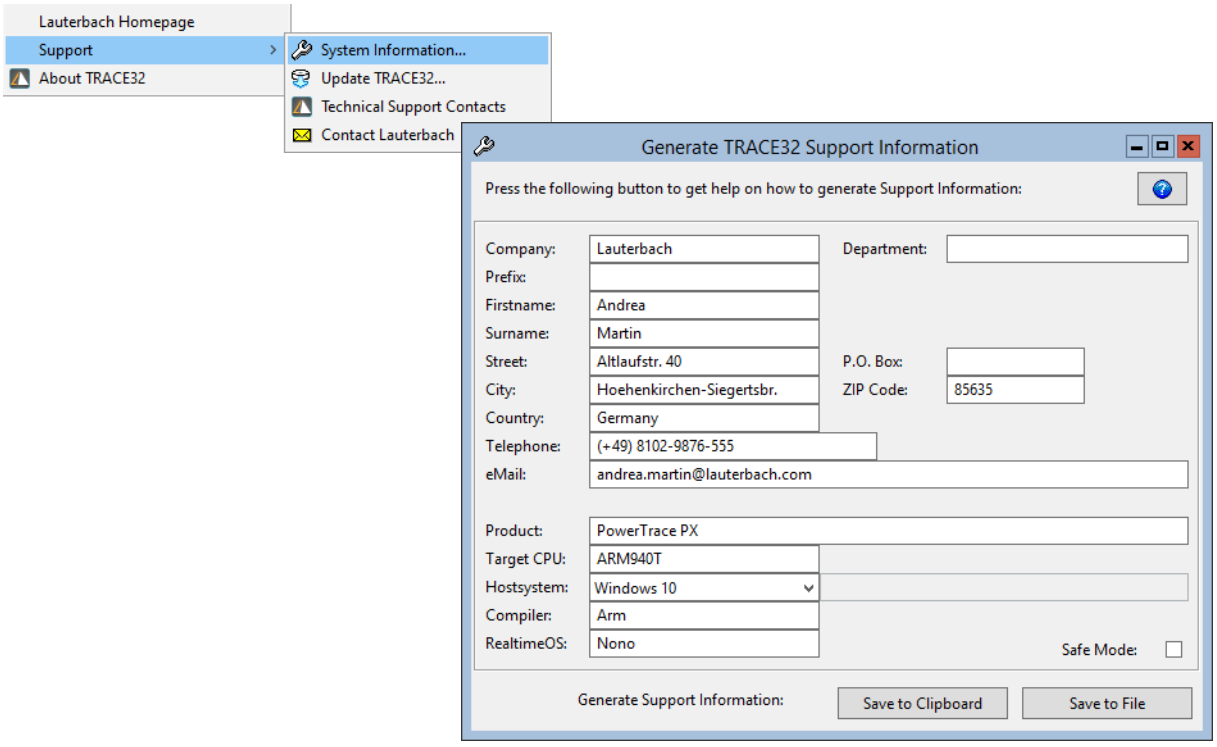
Use the Lauterbach Support Center: https://support.lauterbach.com

- To contact your local TRACE32 support team directly.
- To register and submit a support ticket to the TRACE32 global center.
- To log in and manage your support tickets.
- To benefit from the TRACE32 knowledgebase (FAQs, technical articles, tutorial videos) and our tips & tricks around debugging.

Or send an email in the traditional way to support@lauterbach.com.

Be sure to include detailed system information about your TRACE32 configuration.

1. To generate a system information report, choose **TRACE32** > **Help** > **Support** > **Systeminfo**.



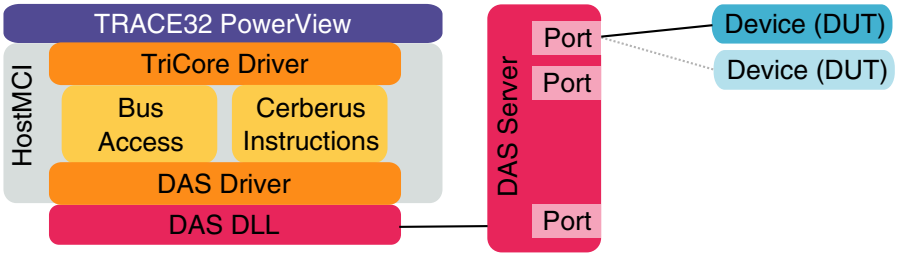
NOTE:	Please help to speed up processing of your support request. By filling out the system information form completely and with correct data, you minimize the number of additional questions and clarification request e-mails we need to resolve your problem.

2. Preferred: click **Save to File**, and send the system information as an attachment to your e-mail.

3. Click **Save to Clipboard**, and then paste the system information into your e-mail.

System Architecture

The following picture illustrates the overall system architecture:



TRACE32 runs completely on a host computer. This includes the debugger back-end (HostMCI) containing the high-performance multicore debug driver; the same driver that is also used together with real Lauterbach hardware.

In the debug back-end, the TriCore driver communicates with the DAS driver through two independent channels (transactors):

- The bus access transactor, which handles accesses to the TriCore system bus(es), and
- The Cerberus instruction transactor, which handles accesses to the TriCore Cerberus module.

The DAS driver communicates through the DAS DLL with the DAS server. The server provides several ports where one or more devices under tests (DUT) can be connected. In TRACE32, the ports are currently referred to as models. TRACE32 can currently debug one device at one port.

PowerView System Configurations

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

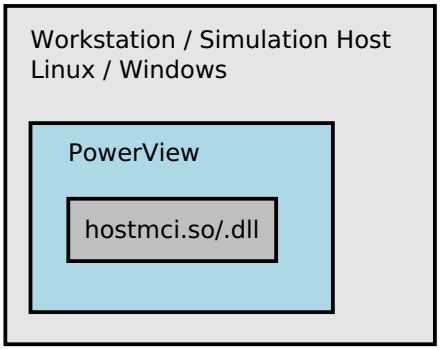
1. A single TRACE32 PowerView instance runs on the same host as the back-end, see Setup 1. This configuration can't handle AMP debug scenarios.
2. Multiple TRACE32 PowerView instances run on the same host as the back-end, see Setup 2.
3. The TRACE32 PowerView instances run on a dedicated workstation;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

The Lauterbach Debug Driver library (`hostmci.so` for Linux/Mac users and `hostmci.dll` for Windows users) can be integrated into the TRACE32 PowerView application or run as a separate process, called `t32mciserver`. Running it as a separate process provides two main benefits:

1. The MCI server can execute on one host, whilst one or more instances of TRACE32 PowerView execute on another host.
2. Multiple instances of TRACE32 PowerView can execute on a single host, sharing the MCI connection.

Setup 1

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

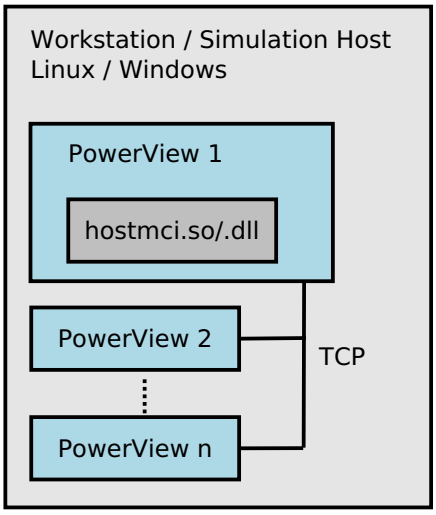


Modify the config.t32 file as follows:

```
PBI=MCILIB                          ; configure system to use hostmci.so
```

Setup 2

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

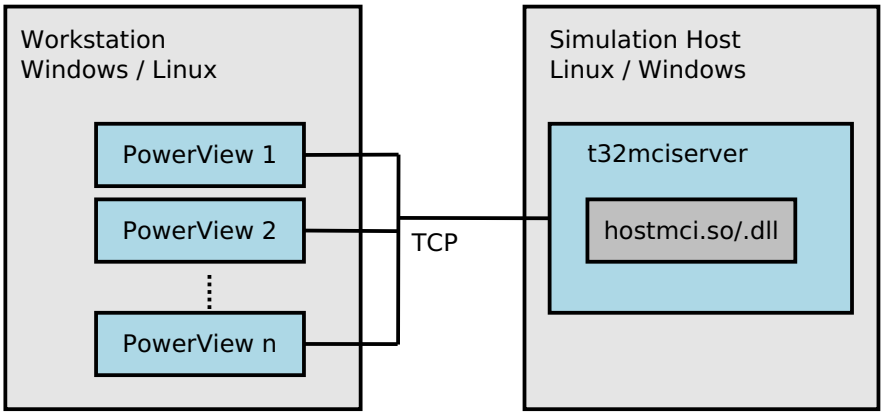


Modify the config.t32 as follows:

```
PBI=MCISERVER                ; set up the usage of hostmci.so and open
PORT=30000                   ; server at 30000 for the first instance.
INSTANCE=AUTO                ; consecutive number of instance or AUTO
```

Setup 3

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



Start t32mciserver on the simulation host:

```
./t32mciserver port=30000        ; start t32mciserver at port 30000
```

Modify the config.t32 file as follows:

```
PBI=MCISERVER                    ; set up connection to t32mciserver
NODE=192.168.0.1                 ; connect to IP 192.168.0.1
PORT=30000                       ; at port 30000
INSTANCE=AUTO                    ; consecutive number of instances
DEDICATED                        ; avoid to fall into Setup2 case
```

Linux example: To start TRACE32 PowerView with a specific config file, use e.g.:

```
bin/pc_linux/t32mtc -c config.t32
```

Windows example: To start TRACE32 PowerView with a specific config file, use e.g.:

```
bin/windows/t32mtc.exe -c config.t32
```

System Initialization via the TRACE32 PowerView GUI

The following step-by-step procedure describes how to connect to the Infineon DAS server by using the TRACE32 PowerView GUI. Alternatively, all steps described below can also be executed via the TRACE32 command line, see **"System Initialization via the TRACE32 Command Line"**, page 11.

Prerequisites:

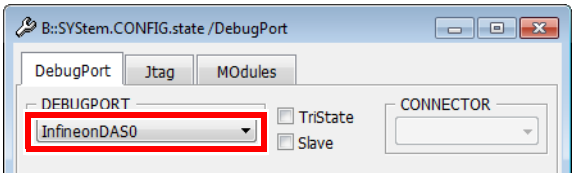
- You have configured and started TRACE32, as described in **"PowerView System Configurations"**, page 6.
- The DAS server works with the default configuration `HOST=localhost`. If the DAS server uses a host other than `localhost`, then you need to inform TRACE32 about the used host with the command **SYStem.InfineonDAS.SERVERCONFIG**.

To initialize the system via the TRACE32 PowerView GUI:

1. Open the **SYStem.CONFIG.state** window by typing at the TRACE32 command line:

```
SYStem.CONFIG.state /DebugPort
```

2. From the **DEBUGPORT** drop-down list, select the debug back-end **InfineonDAS**.



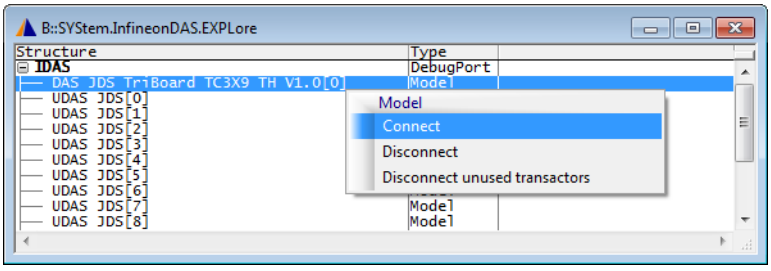
You can close the **SYStem.CONFIG.state** window, since we do not need it anymore in this step-by-step procedure.

Selecting the debug back-end **InfineonDAS** activates the **SYStem.InfineonDAS** commands. Now the **SYStem.InfineonDAS.EXPLore** window can be used to select port and device interactively as described in the following steps:

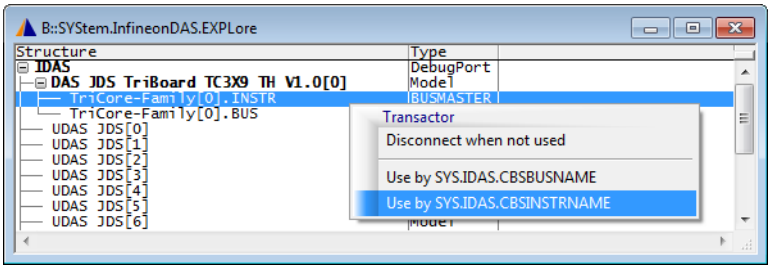
3. Open the **SYStem.InfineonDAS.EXPLore** window by typing at the TRACE32 command line:

```
SYStem.InfineonDAS.EXPLore
```

4. Right-click the model you want to connect to, and then select **Connect** from the popup menu.



5. Expand the entry by clicking the plus sign.



6. To select the Cerberus instruction transactor, right-click the entry for your device ending with ".INSTR", and then select **Use by SYS.IDAS.CBSINSTRNAME** from the popup menu.

   - If the transactor has already been selected, the popup menu displays the option **Do not use by SYS.IDAS.CBSINSTRNAME**.

7. To select the bus access transactor, right click the entry for your device ending with ".BUS", and then select **Use by SYS.IDAS.CBSBUSNAME** from the popup menu.

   - If the transactor has already been selected, the popup menu displays the option **Do not use by SYS.IDAS.CBSBUSNAME**.

Result: You have established a connection to the Infineon DAS server.

Next Steps: You can now select the CPU of the target with the **SYStem.CPU** command and execute the **SYStem.Up** command. You are then ready to debug and trace the target, see **Related Documents**.

Tip

You can save the above configuration as a PRACTICE script (*.cmm) using the **STOre** command:

```
STORE ~~~/my-file.cmm SYStem  ;save script to the temporary directory
                              ;of TRACE32
```

The configuration can be reproduced by calling the PRACTICE script with the **DO** or **CD.DO** command as follows:

```
CD.DO ~~~/my-file.cmm
```

System Initialization via the TRACE32 Command Line

Prerequisites:

- You have configured and started TRACE32, as described in **"PowerView System Configurations"**, page 6.

To initialize the system via the TRACE32 command line:

1. Select the debug back-end **InfineonDAS**.

```
SYStem.CONFIG.DEBUGPORT InfineonDAS
```

2. Configure the DAS server you want to use.

```
SYStem.InfineonDAS.SERVERCONFIG "HOST=localhost"
```

3. Select the DUT you want to use.

```
SYStem.InfineonDAS.MODELNAME "DAS JDS TriBoard TC3X9 TH V1.0[0]"

SYStem.InfineonDAS.CBSINSTRNAME "TriCore-Family[0].INSTR"

SYStem.InfineonDAS.CBSBUSNAME "TriCore-Family[0].BUS"
```

   - The name for `SYStem.InfineonDAS.MODELNAME` is the port name returned from the DAS sever plus an index in square brackets.
   - The name for `SYStem.InfineonDAS.CBSINSTRNAME` is the name of the device as reported from the DAS server plus an index in square brackets and the suffix `.INSTR`.
   - The name for `SYStem.InfineonDAS.CBSBUSNAME` is the name of the device as reported from the DAS server plus an index in square brackets and the suffix `.BUS`.

4. Continue with the normal core configuration; for example, as described in **"Debugging"** (debugger_tricore.pdf). A minimal script would be:

```
SYStem.CPU TC399XE-Astep
SYStem.Up
```

Tip

You can save the above configuration as a PRACTICE script (*.cmm) using the **STOre** command:

```
STORE ~~~/my-file.cmm SYStem  ;save script to the temporary directory
                              ;of TRACE32
```

The configuration can be reproduced by calling the PRACTICE script with the **DO** or **CD.DO** command as follows:

```
CD.DO ~~~/my-file.cmm
```

Keep the Graphical User Interface Responsive

Due to slow RTL simulation, small operations such as reading the state or showing memory dumps take a long time. This chapter describes how to adjust the virtual time scale to ultra-slow simulators and how to reduce screen flicker caused by slow RTL simulation. To keep the user interface smooth multiple tuning options can be set.

The most important setting is **SETUP.URATE** to configure the update rate of the TRACE32 windows. The processors state is also polled by this rate.

```
SETUP.URATE 10s                    ; screen will be updated every 10s
```

To avoid screen update while PRACTICE scripts are running:

```
SCREEN.OFF                         ; switch off update of the windows when
                                   ; a PRACTICE script is executed

SCREEN                             ; trigger a manual update of the windows
                                   ; inside a PRACTICE script
```

To switch off state polling when the CPU is stopped, the command **SYStem.POLLING** can be used, but the debugger can't detect when another CPU changes the state from stopped to running e.g. by soft reset.

```
SYStem.POLLING DEF OFF             ; disable processor state polling when
                                   ; stopped
```

The command **MAP.UpdateOnce** can be used to read memory regions only one time after a break is detected.

```
MAP.UpdateOnce 0x0++0x1000         ; read memory of regions 0x0--0x1000
                                   ; only one time after break
```

For analysis and data display purposes it is recommended that you use the code from the TRACE32 virtual memory (VM:) instead of the code from the target memory. Therefore, the code needs to be copied to the virtual memory when an *.elf file is being loaded.

```
Data.Load.ELF *.elf /VM            ; download code to target and copy it to
                                   ; VM:
Data.List VM:                      ; open source window, but use VM: memory

Onchip.Access VM                   ; use VM memory for trace analysis
```

Timing Adaption

TRACE32 software includes of a set of efficient low-level driver routines to access the target. These routines have a certain timing that must be adjusted to ultra-slow simulators that can be million times slower than real silicon. In general, there are code parts that pause the execution, wait until a time-out is reached or just use a certain point of time.

For example, when the simulation is 1,000,000 times slower than real time, these commands can be used to adjust the timing in most cases:

```
; configure usage of model time base instead host base to avoid timeouts
; while the emulation is paused.
SYStem.VirtualTiming.TimeinTargetTime ON
SYStem.VirtualTiming.PauseinTargetTime ON

;make the pauses and timeouts 100 times shorter
SYStem.VirtualTiming.TimeScale 0.01

;this will limit any pause statements to 10us target time
SYStem.VirtualTiming.MaxPause 10us

;this will limit any small time-out to read register to 1ms
SYStem.VirtualTiming.MaxTimeout 1ms
```

The following timing **SYStem** commands are available:

SYStem.VirtualTiming.MaxPause	Limit pause
SYStem.VirtualTiming.MaxTimeout	Override time-outs
SYStem.VirtualTiming.PauseinTargetTime	Set up pause time-base
SYStem.VirtualTiming.PauseScale	Multiply pause with a factor
SYStem.VirtualTiming.TimeinTargetTime	Set up general time-base
SYStem.VirtualTiming.TimeScale	Multiply time-base with a factor
SYStem.VirtualTiming.HardwareTimeout	Can disable hardware timeout
SYStem.VirtualTiming.HardwareTimeoutScale	Multiply hardware timeout
SYStem.VirtualTiming.InternalClock	Base for artificial time calculation
SYStem.VirtualTiming.OperationPause	Insert a pause after each action to slow down timing.

SYStem.InfineonDAS Configure the InfineonDAS debug port

The **SYStem.InfineonDAS** command group allows to configure the back-end for DAS. The command group is available after **InfineonDAS0** has been selected as debug port.

```
;selecting the DAS back-end activates the SYStem.InfineonDAS commands
SYStem.CONFIG.DEBUGPORT InfineonDAS0
```

See also

- SYStem.InfineonDAS.CBSBUSNAME
- SYStem.InfineonDAS.CBSINSTRNAME
- SYStem.InfineonDAS.CONNECT
- SYStem.InfineonDAS.DISCONNECT
- SYStem.InfineonDAS.EXPLore
- SYStem.InfineonDAS.InfineonDAPNAME
- SYStem.InfineonDAS.MODELNAME
- SYStem.InfineonDAS.SERVERCONFIG
- SYStem.state

SYStem.InfineonDAS.CBSBUSNAME Bus access transactor

Format: **SYStem.InfineonDAS.CBSBUSNAME** *<name>* | **""**

Sets the name of the transactor for system bus accesses (same as selecting **Use by...** from the popup menu in the **SYStem.InfineonDAS.EXPLore** window).

If an empty string is passed, then the setting is reset (same as selecting **Do not use by...** from the popup menu in the **SYStem.InfineonDAS.EXPLore** window).

See also

- SYStem.InfineonDAS

SYStem.InfineonDAS.CBSINSTRNAME Cerberus instruction transactor

Format: **SYStem.InfineonDAS.CBSINSTRNAME** *<name>* | **""**

Sets the name of the transactor for Cerberus instructions (same as selecting **Use by...** from the popup menu in the **SYStem.InfineonDAS.EXPLore** window).

If an empty string is passed, then the setting is reset (same as selecting **Do not use by...** from the popup menu in the **SYStem.InfineonDAS.EXPLore** window).

See also

- SYStem.InfineonDAS

SYStem.InfineonDAS.CONNECT Connect to DAS server

Format: **SYStem.InfineonDAS.CONNECT** [**/TRY**]

Connects to the DAS server.

TRY Forces the command to continue quietly when the connection could not be established.

See also

- SYStem.InfineonDAS

SYStem.InfineonDAS.DISCONNECT Disconnect from the server

Format: **SYStem.InfineonDAS.DISCONNECT** ["*<transactor_name>*"] [**/UNUSED**]

Disconnects from the DAS server and disables the periodic re-connection tries.

<transactor_name> Disconnects a named transactor when it is not used anymore.

UNUSED Disconnects from all transactors that are not used anymore.

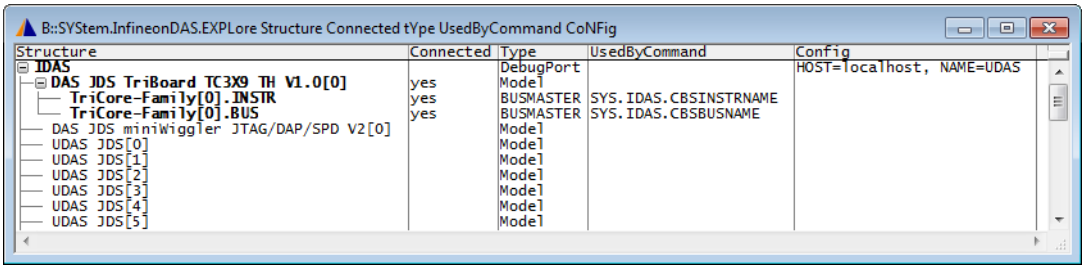
See also

- SYStem.InfineonDAS

Format: **SYStem.InfineonDAS.EXPLore** [**DEFault** | *<column>* …]

<column>: **Structure**
Connected
tYpe
UsedByCommand
CoNFig

Opens the **SYStem.InfineonDAS.EXPLore** window, where you can explore available models and transactors. The list of available devices is read from the DAS server:



DEFault	Without parameter or with the parameter **DEFault**, only the columns **Structure** and **Type** are displayed.
Structure	Shows the name of the models (port instances) and associated transactors.
Connected	A **yes** indicates that TRACE32 has established a connection to the models (port instance) / transactor. A no indicates that no connection has been established.
Type	Type of the entry.
UsedByCommand	Only for transactors: This column displays the command that was executed to set the names of the transactors. See **SYStem.InfineonDAS.CBSBUSNAME** and **SYStem.InfineonDAS.CBSINSTRNAME**.
CoNFig	Only for the root node. The configuration set by **SYStem.InfineonDAS.SERVERCONFIG**.

See also

- SYStem.InfineonDAS

Format: **SYStem.InfineonDAS.InfineonDAPNAME** *<name>* | **""**

Sets the name of the transactor for low-level DAP telegrams (same as selecting **Use by...** from the popup menu in the **SYStem.InfineonDAS.EXPLore** window). This is only required for special use cases.

See also

- SYStem.InfineonDAS

SYStem.InfineonDAS.MODELNAME Select port instance

Format: **SYStem.InfineonDAS.MODELNAME "***<port_name>***[***<index>***]"**

Selects a specific instance of a DAS server port. The name is the port name reported by the DAS server plus an index in square brackets.

The available models can be explored using the **SYStem.InfineonDAS.EXPLore** window.

Example:

```
SYStem.InfineonDAS.MODELNAME "DAS JDS TriBoard TC3X9 TH V1.0[0]"
```

See also

- SYStem.InfineonDAS

Format:	**SYStem.InfineonDAS.SERVERCONFIG** "*<options>*"
<options>:	*<option0>*=*<value0>*, *<option1>*=*<value1>*, …

Configures options to connect to the DAS server. The list of options is a comma-separated list of option name and value pairs. The following options are available:

HOST	Domain name or IP address of the server host. Default: **localhost**.
NAME	Server name. Default: **UDAS**.
START	**AUTO** will automatically start the DAS server if it is not already running. **NO** will only connect to running servers.

Example:

```
SYStem.InfineonDAS.SERVERCONFIG "HOST=127.0.0.1, NAME=UDAS, START=NO"
```

See also

- SYStem.InfineonDAS