

General Commands Reference Guide L

General Commands Reference Guide L

TRACE32 Online Help

TRACE32 Directory

TRACE32 Index

TRACE32 Documents	
General Commands	
General Commands Reference Guide L	1
History	6
LA	7
LA	Logic analyzer 7
LA-specific Trace Commands	8
LA.IMPORT	Import trace information 8
LA.IMPORT.CoreByteStream	Import pure single core trace data 10
LA.IMPORT.cycles	Import bus trace data 11
LA.IMPORT.ELA	Import ELA trace data 11
LA.IMPORT.ETB	Import on-chip trace data 11
LA.IMPORT.GUESSWRAP	Guess wrap pointer 13
LA.IMPORT.StartInvalid	Set start of trace as invalid 13
LA.IMPORT.StartValid	Set start of trace as valid 14
LA.IMPORT.STP	Import STP recording from file (nibble) 14
LA.IMPORT.STPByteStream	Import STP recording from file (byte) 15
LA.IMPORT.TARMAC	Import TARMAC trace file 15
LA.IMPORT.TraceFile	Import trace data where processing has failed 16
LA.IMPORT.TracePort	Import off-chip trace data 16
LA.IMPORT.UltraSOC	Import raw UltraSOC flow trace data 17
LA.IMPORT.VCD	Import recorded signals in VCD file format 17
LA.IMPORT.WRAP	Define wrap pointer 18
LA.Mode	Set the trace operation mode 19
Generic LA Trace Commands	20
LA.ACCESS	Define access path to program code for trace decoding 20
LA.Arm	Arm the trace 20
LA.AutoArm	Arm automatically 20
LA.AutoInit	Automatic initialization 20
LA.BookMark	Set a bookmark in trace listing 20
LA.Chart	Display trace contents graphically 20
LA.CLOCK	Clock to calculate time out of cycle count information 21
LA.ComPare	Compare trace contents 21
LA.ComPareCODE	Compare trace with memory 21

LA.DISable	Disable the trace	21
LA.DRAW	Plot trace data against time	21
LA.EXPORT	Export trace data for processing in other applications	21
LA.FILE	Load a file into the file trace buffer	21
LA.Find	Find specified entry in trace	21
LA.FindAll	Find all specified entries in trace	22
LA.FindChange	Search for changes in trace flow	22
LA.FLOWPROCESS	Process flowtrace	22
LA.FLOWSTART	Restart flowtrace processing	22
LA.GOTO	Move cursor to specified trace record	22
LA.Init	Initialize trace	22
LA.List	List trace contents	22
LA.ListNesting	Analyze function nesting	23
LA.ListVar	List variable recorded to trace	23
LA.LOAD	Load trace file for offline processing	23
LA.OFF	Switch off	23
LA.PROfileChart	Profile charts	23
LA.PROfileSTATistic	Statistical analysis in a table versus time	23
LA.PROTOcol	Protocol analysis	23
LA.PROTOcol.Chart	Graphic display for user-defined protocol	24
LA.PROTOcol.Draw	Graphic display for user-defined protocol	24
LA.PROTOcol.EXPORT	Export trace buffer for user-defined protocol	24
LA.PROTOcol.Find	Find in trace buffer for user-defined protocol	24
LA.PROTOcol.list	Display trace buffer for user-defined protocol	24
LA.PROTOcol.PROfileChart	Profile chart for user-defined protocol	24
LA.PROTOcol.PROfileSTATistic	Profile chart for user-defined protocol	24
LA.PROTOcol.STATistic	Display statistics for user-defined protocol	25
LA.REF	Set reference point for time measurement	25
LA.RESet	Reset command	25
LA.SAVE	Save trace for postprocessing in TRACE32	25
LA.SelfArm	Automatic restart of trace recording	25
LA.SIZE	Define buffer size	25
LA.SnapShot	Restart trace capturing once	25
LA.state	Display trace configuration window	25
LA.STATistic	Statistic analysis	26
LA.Timing	Waveform of trace buffer	26
LA.TRACK	Set tracking record	26
LA.View	Display single record	26
LA.ZERO	Align timestamps of trace and timing analyzers	26
List		27
List	Display modes for programs	27
List.auto	Display program listing	28
List.Asm	Display disassembler	36

List.EXPORT	Export a listing to an XML file	37
List.EXPORT.Asm	Export disassembler listing	37
List.EXPORT.auto	Export source and disassembler listing	38
List.EXPORT.Hll	Export source listing	38
List.EXPORT.Mix	Export source and disassembler listing	39
List.Hll	Display source	41
List.Java	Display Java byte code	42
List.Mix	Disassembler and source	44
LOGGER		45
LOGGER	Trace method LOGGER, recording and analysis commands	45
LOGGER-specific Trace Commands		46
LOGGER.ADDRESS	Software trace address	46
LOGGER.Mode	Set LOGGER operation mode	46
LOGGER.TimeStamp	Configure timestamp usage of LOGGER trace	47
Generic LOGGER Trace Commands		48
LOGGER.ACCESS	Define access path to program code for trace decoding	48
LOGGER.Arm	Arm the trace	48
LOGGER.AutoArm	Arm automatically	48
LOGGER.AutoInit	Automatic initialization	48
LOGGER.BookMark	Set a bookmark in trace listing	48
LOGGER.BookMarkToggle	Toggles a single trace bookmark	48
LOGGER.Chart	Display trace contents graphically	49
LOGGER.ComPare	Compare trace contents	49
LOGGER.DISable	Disable the trace	49
LOGGER.DRAW	Plot trace data against time	49
LOGGER.EXPORT	Export trace data for processing in other applications	49
LOGGER.FILE	Load a file into the file trace buffer	49
LOGGER.Find	Find specified entry in trace	49
LOGGER.FindAll	Find all specified entries in trace	49
LOGGER.FindChange	Search for changes in trace flow	50
LOGGER.FLOWPROCESS	Process flowtrace	50
LOGGER.FLOWSTART	Restart flowtrace processing	50
LOGGER.GOTO	Move cursor to specified trace record	50
LOGGER.Init	Initialize trace	50
LOGGER.List	List trace contents	50
LOGGER.ListNesting	Analyze function nesting	50
LOGGER.ListVar	List variable recorded to trace	51
LOGGER.LOAD	Load trace file for offline processing	51
LOGGER.OFF	Switch off	51
LOGGER.PROfileChart	Profile charts	51
LOGGER.PROfileSTATistic	Statistical analysis in a table versus time	51
LOGGER.PROTOcol	Protocol analysis	51

LOGGER.PROTOcol.Chart	Graphic display for user-defined protocol	51
LOGGER.PROTOcol.Draw	Graphic display for user-defined protocol	52
LOGGER.PROTOcol.EXPORT	Export trace buffer for user-defined protocol	52
LOGGER.PROTOcol.Find	Find in trace buffer for user-defined protocol	52
LOGGER.PROTOcol.list	Display trace buffer for user-defined protocol	52
LOGGER.PROTOcol.PROfileChart	Profile chart for user-defined protocol	52
LOGGER.PROTOcol.PROfileSTATistic	Profile chart for user-defined protocol	52
LOGGER.PROTOcol.STATistic	Display statistics for user-defined protocol	52
LOGGER.REF	Set reference point for time measurement	53
LOGGER.RESet	Reset command	53
LOGGER.SAVE	Save trace for postprocessing in TRACE32	53
LOGGER.SelfArm	Automatic restart of trace recording	53
LOGGER.SIZE	Define buffer size	53
LOGGER.SnapShot	Restart trace capturing once	53
LOGGER.state	Display trace configuration window	53
LOGGER.STATistic	Statistic analysis	53
LOGGER.Timing	Waveform of trace buffer	54
LOGGER.TRACK	Set tracking record	54
LOGGER.View	Display single record	54
LOGGER.ZERO	Align timestamps of trace and timing analyzers	54
LUA		55
LUA	Support for the Lua script language	55
LUA.Data.Loadinput	Load content from a file into the input buffer	55
LUA.Data.Saveoutput	Save output buffer into a binary file	56
LUA.Data.SET	Modify the Lua input buffer	56
LUA.Data.ShowInput	Show current content of the input buffer	57
LUA.Data.ShowOutput	Show current content of the output buffer	57
LUA.Program.List	List the current Lua scripts	57
LUA.Program.LOAD	Load a Lua script to debugger	58
LUA.Program.RESet	Reset the Lua context	58
LUA.Program.RUN	Execute a Lua script	58
LUA.Program.UNLOAD	Remove a Lua script from the debugger	59

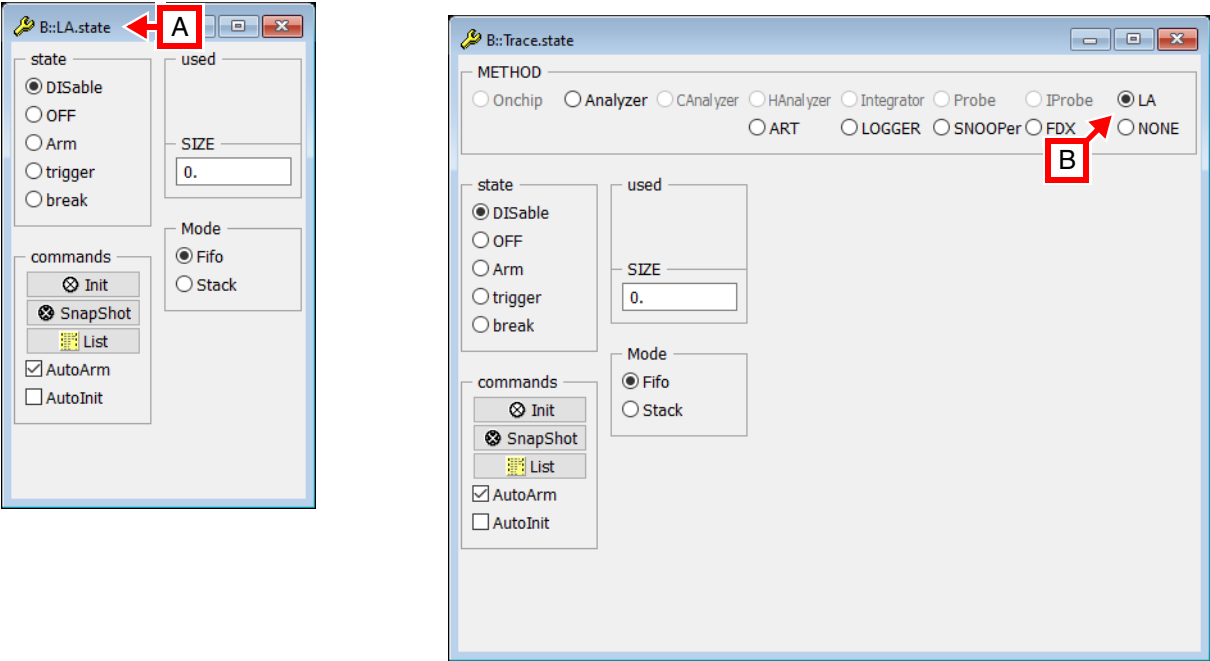
History

- 22-Apr-2024 New option **/NoExec** for [LA.IMPORT.TARMAC](#) command.
- 25-Mar-2024 New option **/COVerableItems** for [List.EXPORT.auto](#) and [List.EXPORT.Mix](#) commands.
- 23-Feb-2024 New command [LA.IMPORT.TARMAC](#).

The trace method LA is used for remote control of logic analyzer systems.

For selecting and configuring the trace method LA, use the TRACE32 command line or a PRACTICE script (*.cmm) or the **LA.state** window [A].

Alternatively, use the **Trace.state** window: click the option **LA** or execute the command **Trace.METHOD LA** in order to select the trace method **LA** [B].



The chapter “**LA-specific Trace Commands**”, page 8 describes the LA-specific configuration commands. While the chapter “**Generic LA Trace Commands**”, page 20 lists the LA trace analysis and display commands, which are shared with other TRACE32 trace methods.

See also

- [Trace.METHOD](#)
- ▲ [‘Generic LA Trace Commands’ in ‘General Commands Reference Guide L’](#)
- ▲ [‘Release Information’ in ‘Legacy Release History’](#)

LA.IMPORT

Import trace information

[\[Example\]](#)

The **LA.IMPORT** command group is used to load trace data from a file into TRACE32 and to analyze it just like data recorded with a TRACE32 trace tool.

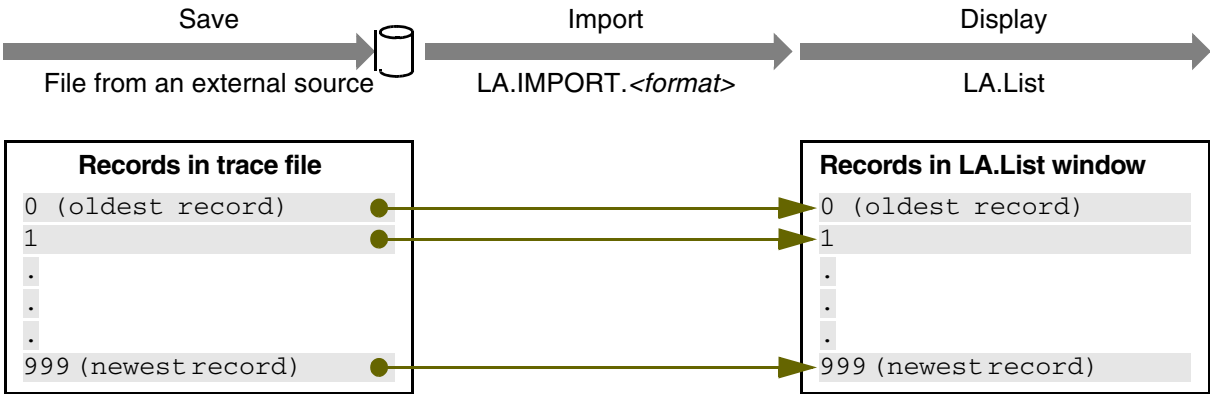
The trace data can be obtained by the application software itself or by another tool or by TRACE32 in a previous debug session in which the processing could not be performed for some reasons.

Trace data successfully obtained and analyzed by TRACE32 can be stored by **<trace>.SAVE** and re-viewed by using the **<trace>.LOAD** command. This is the more convenient way because **<trace>.SAVE** stores a lot of additional information used for the analysis. **LA.IMPORT** imports only the trace raw data. For proper processing you need to inform the debugger about all the trace-relevant circumstances.

All kind of trace postprocessing is only possible with the trace method **LA** (Logic Analyzer). Therefore you need to use **LA.IMPORT** and **LA.*** command group for all analysis commands or better switch the trace method to **LA** (**Trace.METHOD LA**) and use the command group **Trace.*** for all further operations.

LA.IMPORT supports different kinds of trace data and formats. Therefore different commands are provided.

Most trace data is stored in the file in the timely order the data had been generated.



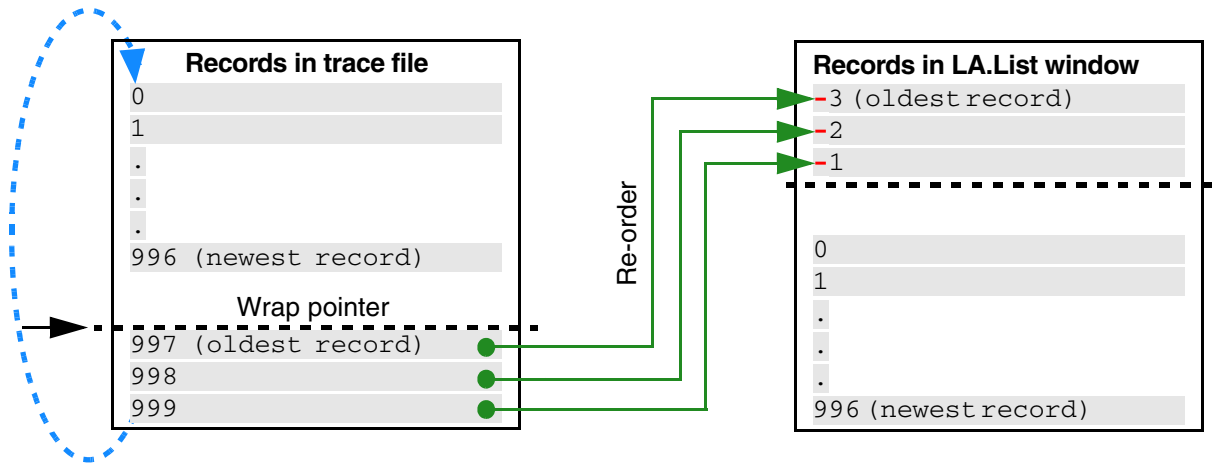
An exception is the on-chip trace buffer, which is typically used as a circular ring buffer overwriting the trace data all the time until the point of interest is reached.

If this buffer is saved into a file, you need to know the wrap pointer for being able to get the data in a timely order. **LA.IMPORT.WRAP** and **LA.IMPORT.GUESSWRAP** will deal with this concern.

Process:



Result:



For post processing trace data loaded by **LA.IMPORT** you need to take the following steps:

1. Start TRACE32 to run as simulator (config.t32 -> **PBI=SIM**). You neither need a debugger hardware nor a target. You can run TRACE32 as debugger as well, but for the postprocessing this is not needed.
2. Adjust all trace relevant settings like for a real target by running the start-up script you used for generating the trace data. For postprocessing an ETMv4 even further setups might be needed which normally the debugger would read out from the ETM module (ETM.COND, ETM.INSTPO, ETM.QE).

If the start-up script is **not** available, then try this:

- At best selecting the chip you are debugging (**SYStem.CPU** ...) is sufficient.
 - For trace data coming from a ARM CoreSight system, all commands describing the trace system on the chip are required (**SYStem.CONFIG** ...).
 - Further all settings for the trace sources done at recording time are needed (e.g. ETM. ...).
3. Load your target application (**Data.LOAD** ...).
 4. Import the trace raw data (**LA.IMPORT** ...).
 5. Now you can use all trace display and analysis functions, e.g.

```
LA.List TP TPC TPINFO DEFault List.NoDummy.OFF ; with diagnostics
```

Example:

```
SYStem.CPU CortexA15

SYStem.CONFIG COREDEBUG.Base 0x82010000
SYStem.CONFIG.ETM.Base 0x8201c000
SYStem.CONFIG.FUNNEL.Base 0x80040000
SYStem.CONFIG.FUNNEL.ATBSource ETM 0
SYStem.CONFIG.ETB.Base 0x80010000

ETM.PortMode.Wrapped
ETM.TraceID 0x55

SYStem.Up

Data.LOAD.Elf myfile.elf

Trace.METHOD.LA
Trace.IMPORT.ETB mydata.bin
Trace.IMPORT.GUESSWRAP

Trace.List TP TPC TPINFO DEFault List.NoDummy.OFF
```

See also

- | | |
|--|--|
| ■ LA.IMPORT.CoreByteStream | ■ LA.IMPORT.cycles |
| ■ LA.IMPORT.ELA | ■ LA.IMPORT.ETB |
| ■ LA.IMPORT.GUESSWRAP | ■ LA.IMPORT.StartInvalid |
| ■ LA.IMPORT.StartValid | ■ LA.IMPORT.STP |
| ■ LA.IMPORT.STPByteStream | ■ LA.IMPORT.TARMAC |
| ■ LA.IMPORT.TraceFile | ■ LA.IMPORT.TracePort |
| ■ LA.IMPORT.UltraSOC | ■ LA.IMPORT.VCD |
| ■ LA.IMPORT.WRAP | ■ <trace>.EXPORT |

LA.IMPORT.CoreByteStream

Import pure single core trace data

Format:

LA.IMPORT.CoreByteStream <file>

Imports pure single core trace data (e.g. for unwrapped single core trace data or x86 IPT traces).

See also

- [LA.IMPORT](#)

Format: **LA.IMPORT.cycles** <file>

Re-imports a file that has been exported with [<trace>.EXPORT.cycles](#). This bus trace data comes from capturing the fetched instructions and data accesses done on an external bus to figure out the program behavior. It works only if no cache is used and if the bus accesses can be captured. This command can be used to import traces from external tools or simulators.

See also

■ [LA.IMPORT](#)

▲ ['Release Information' in 'Legacy Release History'](#)

LA.IMPORT.ELA

Import ELA trace data

Format: **LA.IMPORT.ELA** <file>

Imports a pure binary trace data file obtained from an ARM CoreSight Embedded Logic Analyzer ([ELA](#)).

See also

■ [LA.IMPORT](#)

LA.IMPORT.ETB

Import on-chip trace data

Format: **LA.IMPORT.ETB** <file>

Imports a pure binary trace data file obtained from an on-chip trace buffer like ARM CoreSight ETB, ETF, ETR.

You additionally need to use [LA.IMPORT.WRAP](#) or [LA.IMPORT.GUESSWRAP](#) if the following conditions apply:

- The on-chip trace buffer was used as a circular ring buffer.
- The on-chip trace data was stored as is, it was not read out in the timely order starting from the write pointer position.

LA.Mode FlowTrace will automatically be set when using this command.

See also

■ [LA.IMPORT](#)

▲ ['Release Information' in 'Legacy Release History'](#)

Format: **LA.IMPORT.GUESSWRAP** [<record_number>]

Reformats external trace data loaded to TRACE32 in a timely order. The external trace data of a circular ring buffer is loaded to TRACE32 using [LA.IMPORT.ETB](#). The command **LA.IMPORT.GUESSWRAP** scans the loaded trace data and guesses where the wrap pointer might have been.

Optionally, you can pass a record number where the search for the wrap pointer shall start. Without a parameter it starts from the beginning.

Use [LA.IMPORT.WRAP](#) if you know where the wrap pointer is.

See the figures in the introduction to [<trace>.IMPORT](#).

See also[■ LA.IMPORT](#)[■ LA.IMPORT.WRAP](#)

LA.IMPORT.StartInvalid

Set start of trace as invalid

Format: **LA.IMPORT.StartInvalid**

Reverts the setting done with [LA.IMPORT.StartValid](#).

See also[■ LA.IMPORT](#)

Format: **LA.IMPORT.StartValid** [<address1> <address2> ...]

This command informs the debugger that the start of the loaded trace is valid and that it should not wait for Sync packets.

<address1> The trace is set as valid starting from the given address. On SMP
<address2> ... systems, the first address corresponds to the first core, the second to the
 second core...

See also

■ [LA.IMPORT](#)

LA.IMPORT.STP

Import STP recording from file (nibble)

Format: **LA.IMPORT.STP** <file>

Imports an STP trace from <file> to process it within TRACE32. One trace record is generated per nibble.

In order to unwrap the trace information for processing, TRACE32 needs to know the following information: STM base address and the STP protocol version.

If TRACE32 is aware of the chip characteristic, setting up the chip is sufficient.

Example:

```
SYStem.CPU OMAP4430APP1

LA.IMPORT.STP my_recording.stp

STMLA.List
```

Otherwise the following setup has to be done.

```
SYStem.CONFIG.STM.Base DAP:0xd4161000      ; any base address != 0x0 is
                                           ; fine

SYStem.CONFIG.STM.Mode STPv2                ; specify the STP protocol
                                           ; version

LA.IMPORT.STP my_recording.stp

STMLA.List
```

See also

■ [LA.IMPORT](#)

LA.IMPORT.STPByteStream

Import STP recording from file (byte)

Format:

LA.IMPORT.STPByteStream <file>

Same as [LA.IMPORT.STP](#), but one trace record is generated per byte.

See also

■ [LA.IMPORT](#)

▲ ['Release Information' in 'Legacy Release History'](#)

LA.IMPORT.TARMAC

Import TARMAC trace file

[build 165454 - DVD 02/2024]

Format:

LA.IMPORT.TARMAC <file> [/<option>]

<option>:

NoExec

Imports a trace data file in Tarmac format.

NoExec	Allows the import process to ignore the execution state information (exec/notexec) from the file.
--------	---

See also

■ [LA.IMPORT](#)

Format: LA.IMPORT.TraceFile <file>

Re-imports trace data stored by <trace>.SAVE for re-processing. This is useful if processing was not possible when the trace recording was made. For example if you had no access to the target code at that moment.

Only the trace raw data will be extracted from the saved (*.ad) file.

LA.Mode FlowTrace will automatically be set when using this command.

See also

- LA.IMPORT
- ▲ 'Release Information' in 'Legacy Release History'

Format: LA.IMPORT.TracePort <file>

Imports a pure binary trace data file from an external trace port like an ARM CoreSight TPIU. Unlike on-chip trace data, off-chip trace data includes synchronization packages and depend on the port size of the trace port.

LA.Mode FlowTrace will automatically be set when using this command.

See also

- LA.IMPORT
- ▲ 'Release Information' in 'Legacy Release History'

Format: **LA.IMPORT.UltraSOC** *<file>*

This command allows to load raw UltraSOC flow trace data.

See also

■ [LA.IMPORT](#)

LA.IMPORT.VCD

Import recorded signals in VCD file format

Format: **LA.IMPORT.VCD** *<file>*

Imports a VCD (Value Change Dump) file, which is an industrial standard format for waveforms (not for program trace). It is used for visualizing and analyzing the captured signals in the [<trace>.Timing](#) window.

See also

■ [LA.IMPORT](#)

Format: **LA.IMPORT.WRAP** *<record_number>*

Reformats external trace data loaded to TRACE32 in a timely order. The external trace data of a circular ring buffer is loaded to TRACE32 using [LA.IMPORT.ETB](#).

<record_number> You pass the *<record_number>* of the first trace record in time (wrap pointer). This is the write pointer location of a circular ring buffer the moment the data has been stored.

NOTE: On a CoreSight trace, the write pointer points to a 32-bit value. You need to multiply this value by 4 because each CoreSight trace record is 8 bit in size.

Use [LA.IMPORT.GUESSWRAP](#) if you do not know where the wrap pointer is.

See the figures in the introduction to [LA.IMPORT](#).

See also

■ [LA.IMPORT](#)

■ [LA.IMPORT.GUESSWRAP](#)

Format:	LA.Mode [<i><mode></i>]
<i><mode></i> :	Fifo Stack FlowTrace

Selects the trace operation mode.

Fifo	If the trace is full, new records will overwrite older records. The trace records always the last cycles before the break.
Stack	If the trace is full recording will be stopped. The trace always records the first cycles after starting the trace.
FlowTrace	FlowTrace mode.

See also

■ [<trace>.Mode](#)

LA.ACCESS Define access path to program code for trace decoding

See command [`<trace>.ACCESS`](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 131).

LA.Arm Arm the trace

See command [`<trace>.Arm`](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 134).

LA.AutoArm Arm automatically

See command [`<trace>.AutoArm`](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 135).

LA.AutoInit Automatic initialization

See command [`<trace>.AutoInit`](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 140).

LA.BookMark Set a bookmark in trace listing

See command [`<trace>.BookMark`](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 140).

LA.Chart Display trace contents graphically

See command [`<trace>.Chart`](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 144).

LA.CLOCK

Clock to calculate time out of cycle count information

See command [`<trace>.CLOCK`](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 191).

LA.ComPare

Compare trace contents

See command [`<trace>.ComPare`](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 192).

LA.ComPareCODE

Compare trace with memory

See command [`<trace>.ComPareCODE`](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 194).

LA.DISable

Disable the trace

See command [`<trace>.DISable`](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 197).

LA.DRAW

Plot trace data against time

See command [`<trace>.DRAW`](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 201).

LA.EXPORT

Export trace data for processing in other applications

See command [`<trace>.EXPORT`](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 212).

LA.FILE

Load a file into the file trace buffer

See command [`<trace>.FILE`](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 233).

LA.Find

Find specified entry in trace

See command [`<trace>.Find`](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 235).

See command [**<trace>.FindAll**](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 237).

See command [**<trace>.FindChange**](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 238).

See command [**<trace>.FLOWPROCESS**](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 241).

See command [**<trace>.FLOWSTART**](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 241).

See command [**<trace>.GOTO**](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 244).

See command [**<trace>.Init**](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 246).

See command [**<trace>.List**](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 248).

See command [**<trace>.ListNesting**](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 263).

LA.ListVar**List variable recorded to trace**

See command [**<trace>.ListVar**](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 266).

LA.LOAD**Load trace file for offline processing**

See command [**<trace>.LOAD**](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 270).

LA.OFF**Switch off**

See command [**<trace>.OFF**](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 278).

LA.PROfileChart**Profile charts**

See command [**<trace>.PROfileChart**](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 283).

LA.PROfileSTATistic**Statistical analysis in a table versus time**

See command [**<trace>.PROfileSTATistic**](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 322).

LA.PROTOcol**Protocol analysis**

See command [**<trace>.PROTOcol**](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 339).

See command [**<trace>.PROTOcol.Chart**](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 339).

See command [**<trace>.PROTOcol.Draw**](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 341).

See command [**<trace>.PROTOcol.EXPORT**](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 342).

See command [**<trace>.PROTOcol.Find**](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 343).

See command [**<trace>.PROTOcol.list**](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 344).

See command [**<trace>.PROTOcol.PROfileChart**](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 347).

See command [**<trace>.PROTOcol.PROfileSTATistic**](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 348).

See command [**<trace>.PROTOcol.STATistic**](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 350).

LA.REFSet reference point for time measurement

See command [**<trace>.REF**](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 357).

LA.RESetReset command

See command [**<trace>.RESet**](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 357).

LA.SAVESave trace for postprocessing in TRACE32

See command [**<trace>.SAVE**](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 358).

LA.SelfArmAutomatic restart of trace recording

See command [**<trace>.SelfArm**](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 362).

LA.SIZEDefine buffer size

See command [**<trace>.SIZE**](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 373).

LA.SnapShotRestart trace capturing once

See command [**<trace>.SnapShot**](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 373).

LA.stateDisplay trace configuration window

See command [**<trace>.state**](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 376).

See command [`<trace>.STATistic`](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 378).

LA.Timing

Waveform of trace buffer

See command [`<trace>.Timing`](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 499).

LA.TRACK

Set tracking record

See command [`<trace>.TRACK`](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 502).

LA.View

Display single record

See command [`<trace>.View`](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 504).

LA.ZERO

Align timestamps of trace and timing analyzers

See command [`<trace>.ZERO`](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 505).

The **List** command group displays a program listing:

- Source code and object code in the **List.Mix** window
- Object code only in the **List.Asm** window
- Source code only in the **List.Hll** window
- Java byte code in the **List.Java** window

In addition, the program listing can be exported with **List.EXPORT**.

See also

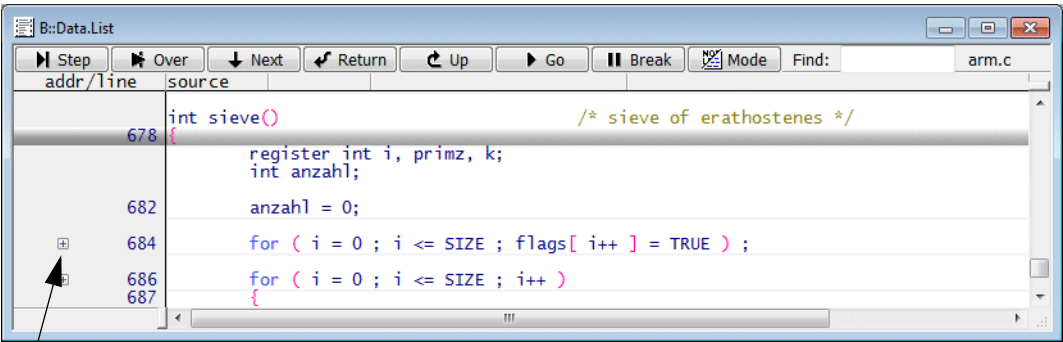
- | | | | |
|--------------------|---------------------|-----------------------|------------------------|
| ■ List.Asm | ■ List.auto | ■ List.EXPORT | ■ List.Hll |
| ■ List.Java | ■ List.Mix | ■ Data.dump | ■ Data.LOAD |
| ■ Go | ■ SETUP.DIS | ■ SETUP.LISTCLICK | ■ SETUP.sYmbol |
| ■ SETUP.TIMEOUT | ■ Step | ■ WinOverlay | □ ADDRESS.isPHYSICAL() |
| □ ADDRESS.OFFSET() | □ ADDRESS.SEGMENT() | □ ADDRESS.STRACCESS() | □ ADDRESS.WIDTH() |
| □ sYmbol.END() | □ sYmbol.EXIT() | | |
- ▲ 'Release Information' in 'Legacy Release History'

Format:	List.auto [<i><address></i> <i><range></i>] [<i>/<option></i>] Data.List (deprecated)
<i><option></i> :	Mark <i><break></i> Flag <i><flag></i> (EF) DIVERGE COVerage CACHE Track TOrder SOrder ISTAT [<i><parameter></i>]
<i><flag></i> :	Read Write NoRead NoWrite
<i><break></i> :	Program Hll Spot Read Write Alpha Beta Charly Delta Echo
<i><parameter></i> :	DEFault ALL CLOCKS TCLOCKS SAMPLES COVerage

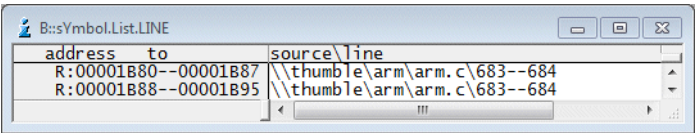
The display format (assembler, mixed or HLL) of the **List.auto** window is selected dynamically, depending on the current debug mode. You can change the debug mode by clicking the **Mode** button in the toolbar of a **List.*** window or by using the **Mode** command.

If no address is specified, the window tracks the value of the program counter (PC). The window is only scrolled, if the bar moves outside of a predefined subwindow. The display format may be specified with the **SETUP.DIS** command.

If the source listing is displayed in HLL mode, some code lines may be marked with a tree button. This tree button indicates that the compiler generated assembler code at more than one address location for this source code line. This is very common for **for-while** loops as well as for code compiled with a high optimization level.



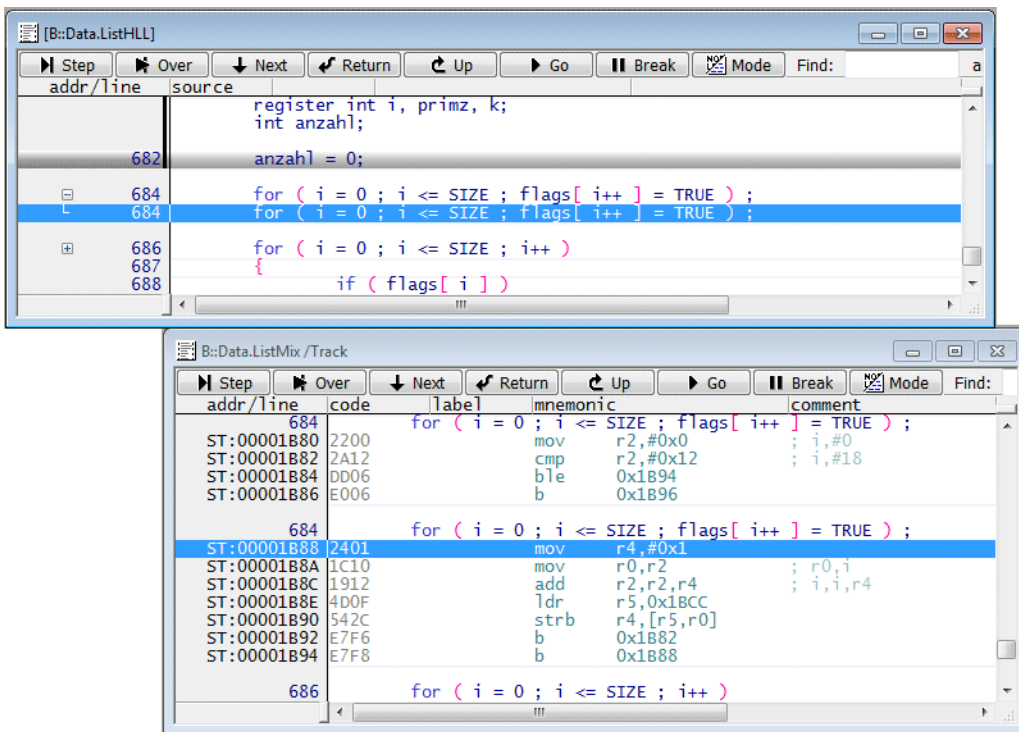
tree button



Assembler code at two address locations is generated for the HLL line number 683--684

If you want to inspect this in detail, the following commands might be helpful:

Data.ListHll	; display the source code in HLL ; mode (source order)
Data.ListMix /Track	; display the source code in Mixed ; mode (target line order)



If you push the tree button to get detailed information, a duplicate of the original HLL line is generated for each assembler code address location.

If you now select one of the duplicates, the cursor in the **Data.ListMix** window points automatically to the corresponding assembler code. This feature is enabled by the **/Track** option.

Shaded lines indicate that the source information is not on disk or in the wrong directory. See command **Symbol.SPATH**. For an example of shaded lines in a **List** window, see screenshot in **List.Mix**.

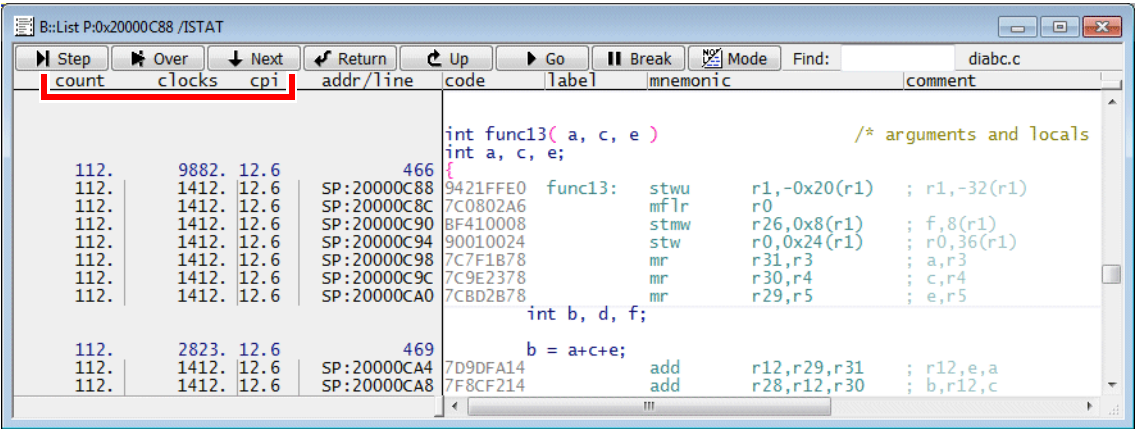
Mark	The Mark option highlights individual lines, depending on the breakpoint type.
MarkPC	The MarkPC option highlights all HLL source lines belonging to the current PC. See example .
Flag	Mark a line with specific flag memory contents, e.g. Read
DIVERGE	This option is mainly intended for internal diagnostic purposes: It attaches tags to executed and not executed ASM and HLL lines. The tags are displayed in the scale area of the List.* windows. You can open the scale area by dragging the slider control to the right. - For information about the tags, see example. - See also Step.Diverge.
COVerage	Displays trace based code coverage information. The COVerage option highlights all code lines that have not been fully covered. For more information, see COVerage .
CACHE	Displays cache hit information and marks currently cached code.
Track	Tracks the window to the reference position of other windows. The window tries first to track to the PROGRAM reference, and if this reference is not valid, it tracks to the DATA reference.
TOrder	List source lines in target line order . This is the default for assembly and mixed mode displays.
SOrder	List source lines in source line order . This is the default for source level displays.
ISTAT [<i><parameter></i>]	Display source listing together with the information provided by the instruction trace database (ISTATistic.ListFunc). Instructions that have not been executed are highlighted in yellow. - For a description of the <i><parameters></i>, see table below. ISTAT used without parameter, see example. ISTAT used with the COVerage parameter, see example.

Default	Display the default information provided by the ISTAT database.
ALL	Display all information provided by the ISTAT database.
CLOCK	Display the clock and CPI information provided by the ISTAT database.
TCLOCK	(only for special purposes)
SAMPLES	Display recorded samples, time and ratio.
COVeRage	Display the code coverage information provided by the ISTAT database.

Examples

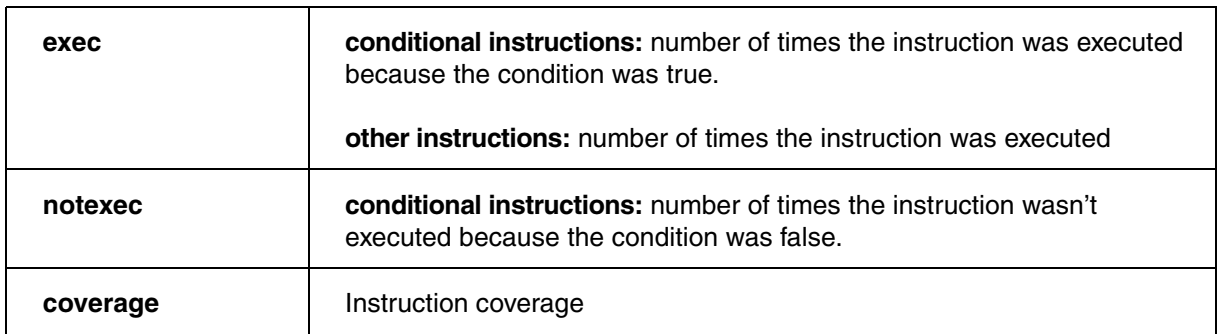
Example for the ISTAT option - without <parameter>

```
Data.ListAsm func13 /ISTAT           ; list instruction run-time
                                     ; statistic
```



count	Total number of instruction executions
clocks	Total number of clocks for the instruction
cpi	Average clocks per instruction

```
; list instruction coverage
```



```
Data.List                                ; display source listing around the
                                         ; current PC

Data.List /Mark Program                 ; display source listing, bold print
                                         ; all instructions / HLL lines on a
                                         ; yellow background if a program
                                         ; breakpoint is set

Data.List /Mark                         ; remove bold printing on yellow
                                         ; background
```

```

Data.List /Track          ; track the window to a reference, e.g.
                           ; analyzer

Data.List Register(a0)    ; follow the register A0 of the CPU

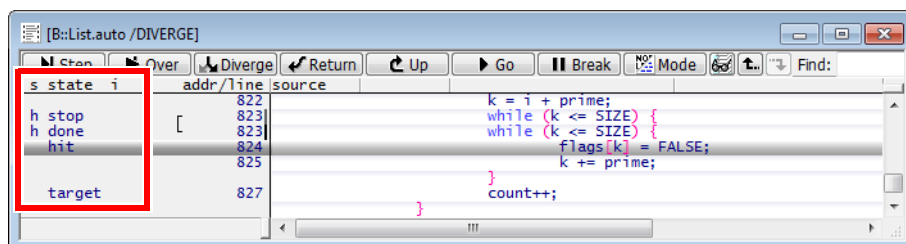
PRINT Data.Long(d:0x200)  ; prints the memory contents of addresses
                           ; D:0x200..0x203
                           ; Do not mix up the command Data.List
                           ; with the function Data.Long()

                           ; Short form of the command is D.L
                           ; Short form of the function is D.L()

```

Example for the DIVERGE option

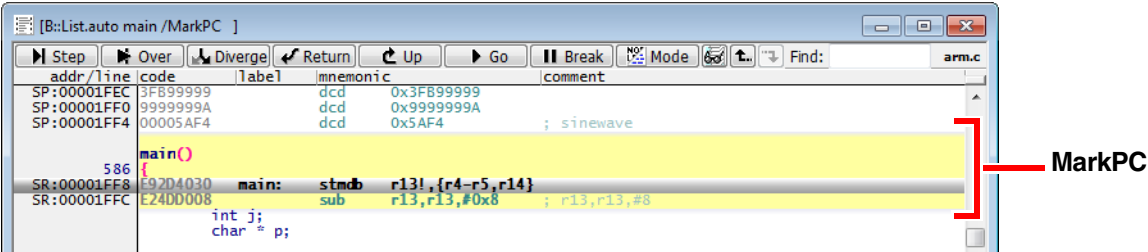
[\[Back\]](#)



Tags in the columns s, state, and i

h	Line in HLL mode (Mode.Hll).
a	Address in ASM mode (Mode.Asm).
stop	Reached by a breakpoint.
done	An executed line or address.
hit	A reached line or address that has not yet been executed, e.g. in a linear code sequence.
target	This line or address is a possible target of the next Step.Diverge . Once reached, target is replaced with hit .
i	Indirect branch instruction.

```
Register.Set PC main      ; set the Program Counter to the label main
List.auto main /MarkPC    ; highlight all instructions related to the
                          ; current HLL line
```



See also

- List
 - List.Java
 - SETUP.sYmbol
 - ADDRESS.SEGMENT()
 - sYmbol.EXIT()
- List.Asm
 - List.Mix
 - SETUP.TIMEOUT
 - ADDRESS.STRACCESS()
- List.EXPORT
 - SETUP.DIS
 - ADDRESS.isPHYSICAL()
 - ADDRESS.WIDTH()
- List.Hll
 - SETUP.LISTCLICK
 - ADDRESS.OFFSET()
 - sYmbol.END()
- ▲ 'Release Information' in 'Legacy Release History'

Format:

List.Asm

[<address>] [/<option>]

Data.ListAsm

(deprecated)

<option>:

Mark

<break>

Flag

<flag>

(EF)

COVERage

CACHE

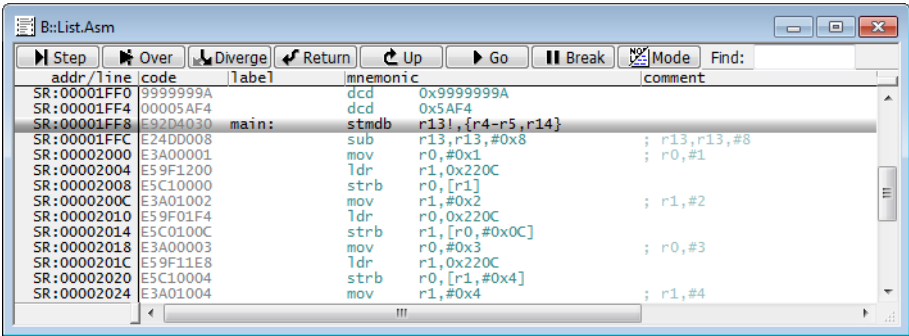
Track

MarkPC

TOrder | SOrder

IgnoreSymbols

Displays the program in disassembled format. The functionality is the same as the [Data.List](#) command.



<option>	For a description of the options, see List.auto command.
IgnoreSymbols	Let the disassembler ignore any symbols for deciding at which byte of the machine code the disassembling should start. This option does normally only make sense on architectures with different instruction lengths.

See also

- [List](#)
- [List.Mix](#)
- [List.auto](#)
- [Mode](#)
- [List.Hll](#)
- ☐ [DEBUGMODE\(\)](#)
- [List.Java](#)

Using the **List.EXPORT** command group, you can export a source or disassembler listing or both listings to an XML file.

In addition, TRACE32 provides an XSL transformation template for formatting the XML file. The formatting is automatically applied to the XML file when it is opened in an external browser window. Prerequisite: The XSL file is placed in the same folder as the XML file.

For demo scripts, see [List.EXPORT.Mix](#).

See also

- [List.EXPORT.Asm](#)
- [List.EXPORT.Hll](#)
- [List.EXPORT.Mix](#)
- [List](#)
- [List.EXPORT.auto](#)
- [COVerage.EXPORT](#)
- [ISTATistic.EXPORT](#)
- [List.auto](#)

List.EXPORT.Asm

Export disassembler listing

Format:

List.EXPORT.Asm *<file> <range> [/<option>]*
Data.ListEXPORTAsm (as an alternative)

<option>:

COVerage | ISTATistic Append | NoData

Creates an XML file containing the disassembler listing. For an example and a description of the options, see [List.EXPORT.Mix](#).

See also

- [List.EXPORT](#)

Format:	List.EXPORT.auto <file> <range> [/<option>] Data.ListEXPORT (as an alternative)
<option>:	COVerage ISTATistic TOrder SOrder Append NoData COVerableItems

Creates an XML file containing the source listing and the disassembler listing - same as [List.EXPORT.Mix](#). For an example and a description of the options, see [List.EXPORT.Mix](#).

See also

- [List.EXPORT](#)

Format:	List.EXPORT.HII <file> <range> [/<option>] Data.ListEXPORTHII (as an alternative)
<option>:	COVerage ISTATistic Append NoData

Creates an XML file containing just the source listing. For an example and a description of the options, see [List.EXPORT.Mix](#).

See also

- [List.EXPORT](#)

Format:	List.EXPORT.Mix <file> <range> [/<option>] Data.ListEXPORTMix (as an alternative)
<option>:	COV erage ISTAT istic SO rder TO rder A ppend No Data COV erable I tems

Creates an XML file containing the source listing and the disassembler listing.

<file>	Name of the XML file that stores a listing of the source and disassembler code. The file extension *.xml can be omitted.
<range>	Address filter for exporting the specified range.
A ppend	Appends the listing to an existing XML file - without overwriting the current file contents.
COV erage	Listing additionally contains code coverage information.
COV erable I tems	Exports only coverage results for measurable source lines.
ISTAT istic	Listing additionally contains information provided by the ISTATistics module.
No Data	Excludes data-only sections from the XML output.
SO rder	Export the source code lines in source order .
TO rder	Export the source code lines in target order (default).

Example 1: The prerequisites for the following example are that the debug symbols have already been loaded, the address bookmarks have been created, and trace data has been recorded.

```
COverage.ADD                ;update the coverage database
Data.List /COverage /Track  ;display source listing
COverage.ListFunc           ;display coverage for HLL functions

;export all bookmarks
BookMark.EXPORT "~~/list.xml"

;export the source listing of the functions "main" and "sieve"
List.EXPORT.Mix "~~/list.xml" main /COverage /Append
List.EXPORT.Mix "~~/list.xml" sieve /COverage /Append

;for demo purposes: let's open the unformatted result in TRACE32
EDIT "~~/list.xml"

;place the transformation template in the same folder as the XML file
COPY "~~/demo/coverage/single_file_report/t32transform.xml" \
    "~~/t32transform.xml"

;you can now open the formatted result in an external browser window
OS.Command start iexplore.exe "file:///C:/t32/list.xml"
```

The tildes ~~ expand to your TRACE32 system directory, by default c:\t32.

Example 2: A more complex demo script is included in your TRACE32 installation. To access the script, run this command:

```
B::CD.PSTEP ~~/demo/coverage/example.cmm
```

See also

■ [List.EXPORT](#)

Format:

List.Hll [<address>] [/<option>]

Data.ListHll (as an alternative)

<option>:

Mark <break>

Flag <flag> (EF)

COverage

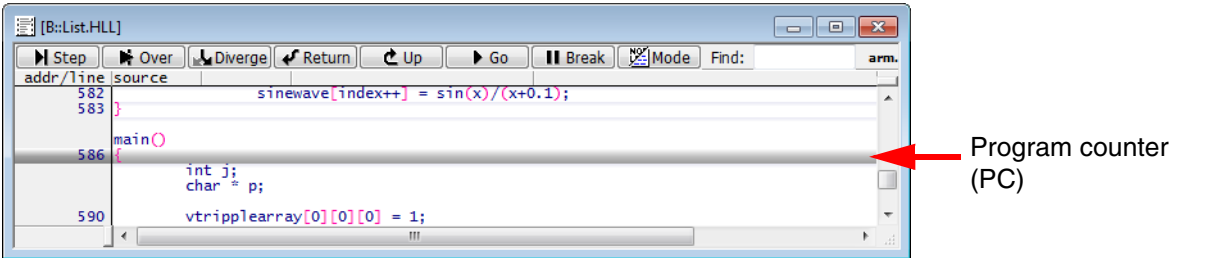
CACHE

Track

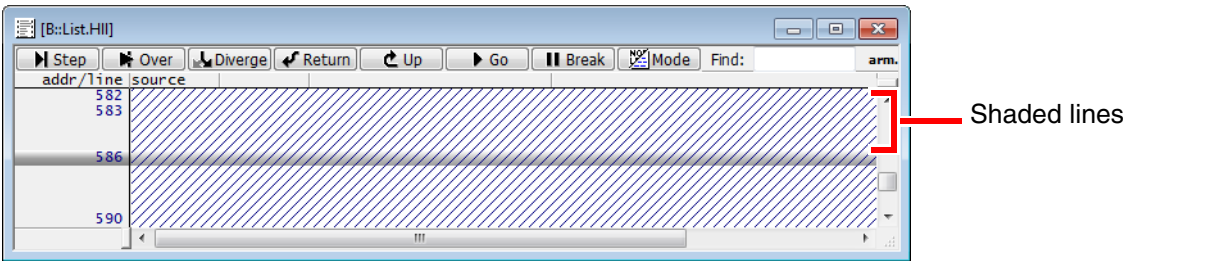
MarkPC

TOrder | SOrder

Displays the program in source format. The functionality is the same as the [Data.List](#) command. If the starting address is not an HLL (High Level Language) line, assembler code is displayed to the next HLL line found in the code segment.



Shaded lines indicate that the source information is not on disk or in the wrong directory. See command [sYmbol.SPATh](#).



<option>	For a description of the options, see List.auto .
----------	---

See also

- List

List.Mix
- List.Asm

Mode
- List.auto

DEBUGMODE()
- List.Java

Format:	List.Java [<i><address></i>] [<i>/<option></i>] Data.ListJava (as an alternative)
<i><option></i> :	Mark <i><break></i> Flag <i><flag></i> (EF) COVerage CACHE Track TOrder SOrder

Displays the program in Java byte code format. The functionality is the same as the **Data.List** command.

This command is NOT required when an address range is specified as Java byte code area with the Java byte code debugger or when the processor implements a special byte code mode (e.g. ARM Jazelle).

<i><option></i>	For a description of the options, see List.auto .
-----------------------	--

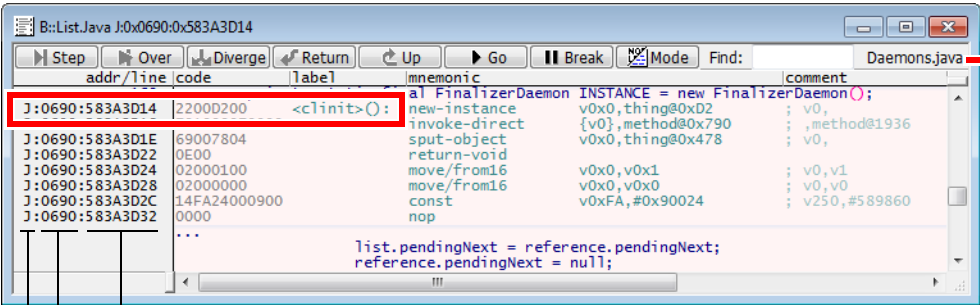
Android is used in this example:

```
;list all running tasks by magic number, task name, task ID, etc.
TASK.List.tasks

;change view to a task by specifying the magic number, task name or ID
Frame.TASK 0xEFD29700

;display a source listing at this address for the task specified above
List.Java J:0x0690:0x583A3D14

;alternatively, use the label to display the same source listing.
List.Java `<clinit>()``           ;enclose the label in backticks `...`
```



Java file name:
Click to open the
symbol browser.

Address
Thread ID
Access class J for Java

See also

- List
- List.Asm
- List.auto
- List.Hll
- List.Mix

Format:

List.Mix

[<address>] [/ <option>]

Data.ListMix

(as an alternative)

<option>:

Mark

<break>

Flag

<flag>

(EF)

COVERAGE

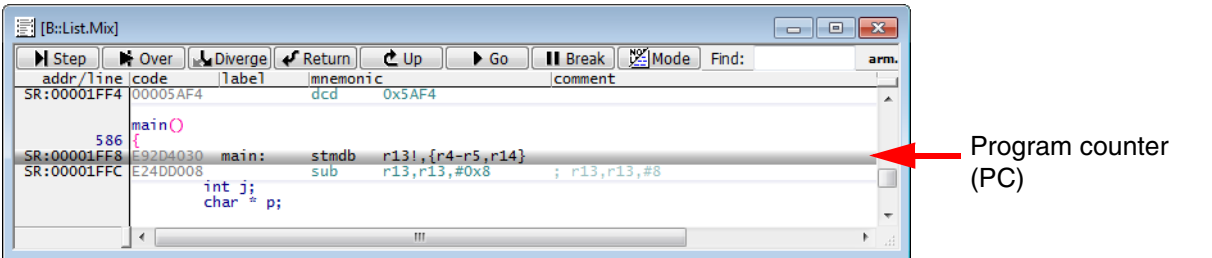
CACHE

Track

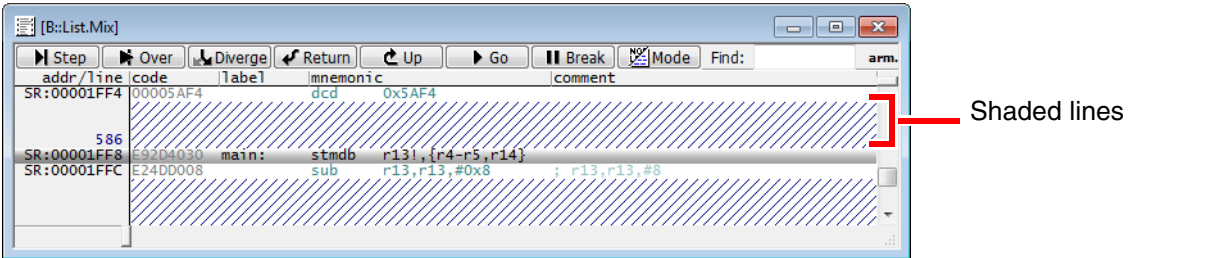
MarkPC

TOrder | SOrder

The code is displayed in HLL (High Level Language) and additionally disassembled from the memory. The functionality is the same as the [Data.List](#) command.



Shaded lines indicate that the source information is not on disk or in the wrong directory. See the command [sYmbol.SPATh](#).



<option>	For a description of the options, see List.auto command.
----------	--

See also

- [List](#)
- [List.Asm](#)
- [List.auto](#)
- [List.Hll](#)
- [List.Java](#)
- [Mode](#)
- [DEBUGMODE\(\)](#)

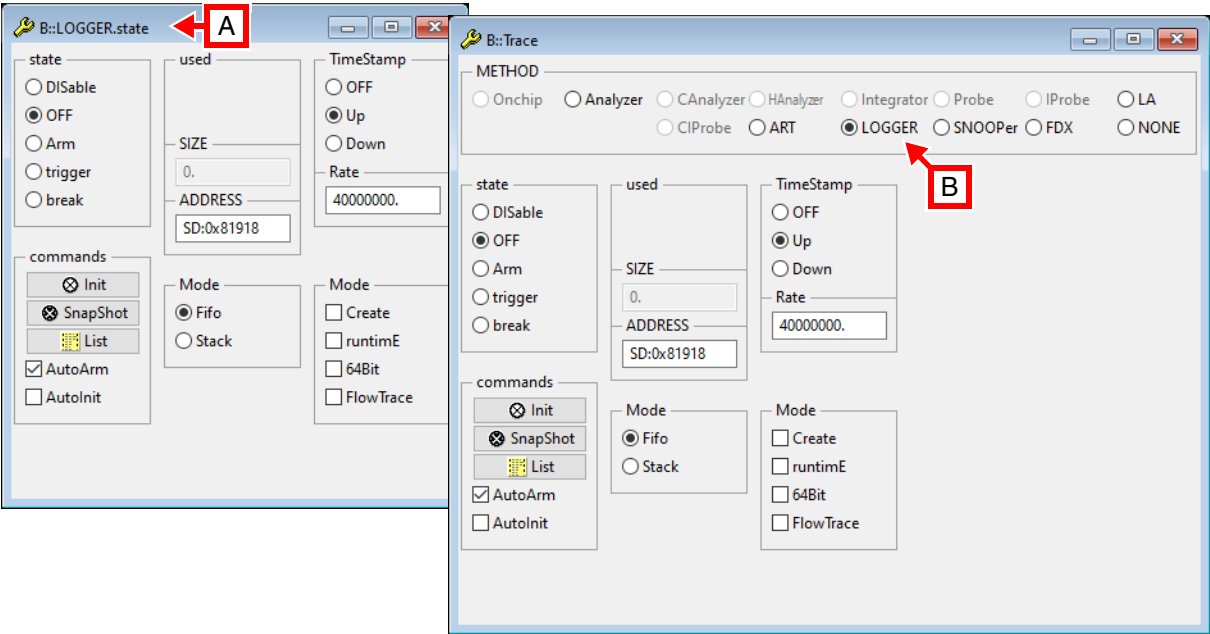
LOGGER Trace method LOGGER, recording and analysis commands

LOGGER is a software trace method where the target application writes the required trace information to the a reserved buffer on the target RAM. TRACE32 loads then the trace information from the target RAM for display and processing.

Please refer to the “[Application Note for the LOGGER Trace](#)” (app_logger.pdf) for more information.

For selecting and configuring the trace method LOGGER, use the TRACE32 command line or a PRACTICE script (*.cmm) or the **LOGGER.state** window [A].

Alternatively, use the **Trace.state** window: click the option **LOGGER** or execute the command **Trace.METHOD Analyzer** in order to select the trace method **LOGGER** [B].



The chapter “[LOGGER-specific Trace Commands](#)”, page 46 describes the LOGGER-specific configuration commands. While the chapter “[Generic LOGGER Trace Commands](#)”, page 48 lists the LOGGER trace analysis and display commands, which are shared with other TRACE32 trace methods.

See also

- [FDX](#)
- [Trace.METHOD](#)

LOGGER-specific Trace Commands

LOGGER.ADDRESS

Software trace address

Format: **LOGGER.ADDRESS** [*<address>*]

Defines the address of the logger trace control block in target memory.

LOGGER.Mode

Set LOGGER operation mode

Format: **LOGGER.Mode** [*<mode>*]

<mode>:

Fifo | Stack

**Create
runtimeE
64bit**

FlowTrace

Selects the trace operation mode.

- Fifo**

If the trace is full, new records will overwrite older records. The trace records always the last cycles before the break.
- Stack**

If the trace is full recording will be stopped. The trace always records the first cycles after starting the trace.
- Create**

Create LOGGER software trace control block by debugger.
- runtimeE**

Dualport access.
- 64Bit**

LOGGER mode for 64-bit traces.
- FlowTrace**

Special mode where the LOGGEr is used to sample the program flow. The LOGGER trace listing reconstructs the program flow based on the sampled information. Only supported for PowerPC and SH4.

See also

■ [<trace>.Mode](#)

Format: **LOGGER.TimeStamp OFF | Up | Down | Rate <rate>**

Configure timestamps for the **LOGGER** trace. The **LOGGER** trace record format includes a timestamp field for up to 48 bit timestamps. The direction and rate information passed by this command is required to convert the timestamp into the time in seconds.

OFF (default)	Disable timestamps. Use this setting if the LOGGER target code does not store timestamps in the LOGGER trace records. When this setting is used, the x-direction in chart views is the record number axis instead of the time axis.
Up	Enable timestamp counter, counting upwards. Use this setting if the LOGGER target code stores timestamps in the LOGGER trace records and if the timestamp increments with each timer tick.
Down	Enable timestamp counter, counting downwards. Use this setting if the LOGGER target code stores timestamps in the LOGGER trace records and if the timestamp decrements with each timer tick.
Rate <rate>	Frequency of the timestamp in ticks per second.
AllCycles [ON OFF] SH only	Set timestamp generation frequency. OFF (default): Generate a single timestamp for 6 trace cycles. ON: Generate dedicated timestamps for all trace cycles.

Example: The timestamp used by the **LOGGER** target code increments at a rate of 16 million per second (16 MHz):

```
LOGGER.TimeStamp Up
LOGGER.TimeStamp Rate 16000000.
```

Generic LOGGER Trace Commands

LOGGER.ACCESS Define access path to program code for trace decoding

See command [`<trace>.ACCESS`](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 131).

LOGGER.Arm Arm the trace

See command [`<trace>.Arm`](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 134).

LOGGER.AutoArm Arm automatically

See command [`<trace>.AutoArm`](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 135).

LOGGER.AutoInit Automatic initialization

See command [`<trace>.AutoInit`](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 140).

LOGGER.BookMark Set a bookmark in trace listing

See command [`<trace>.BookMark`](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 140).

LOGGER.BookMarkToggle Toggles a single trace bookmark

See command [`<trace>.BookMarkToggle`](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 143).

See command **<trace>.Chart** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 144).

LOGGER.ComPare

Compare trace contents

See command **<trace>.ComPare** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 192).

LOGGER.DISable

Disable the trace

See command **<trace>.DISable** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 197).

LOGGER.DRAW

Plot trace data against time

See command **<trace>.DRAW** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 201).

LOGGER.EXPORT

Export trace data for processing in other applications

See command **<trace>.EXPORT** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 212).

LOGGER.FILE

Load a file into the file trace buffer

See command **<trace>.FILE** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 233).

LOGGER.Find

Find specified entry in trace

See command **<trace>.Find** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 235).

LOGGER.FindAll

Find all specified entries in trace

See command **<trace>.FindAll** in 'General Commands Reference Guide T' (general_ref_t.pdf, page 237).

See command [**<trace>.FindChange**](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 238).

LOGGER.FLOWPROCESSProcess flowtrace

See command [**<trace>.FLOWPROCESS**](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 241).

LOGGER.FLOWSTARTRestart flowtrace processing

See command [**<trace>.FLOWSTART**](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 241).

LOGGER.GOTOMove cursor to specified trace record

See command [**<trace>.GOTO**](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 244).

LOGGER.InitInitialize trace

See command [**<trace>.Init**](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 246).

LOGGER.ListList trace contents

See command [**<trace>.List**](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 248).

LOGGER.ListNestingAnalyze function nesting

See command [**<trace>.ListNesting**](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 263).

See command [**<trace>.ListVar**](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 266).

See command [**<trace>.LOAD**](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 270).

See command [**<trace>.OFF**](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 278).

See command [**<trace>.PROfileChart**](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 283).

See command [**<trace>.PROfileSTATistic**](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 322).

See command [**<trace>.PROTOcol**](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 339).

See command [**<trace>.PROTOcol.Chart**](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 339).

See command [<trace>.PROTOcol.Draw](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 341).

See command [<trace>.PROTOcol.EXPORT](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 342).

See command [<trace>.PROTOcol.Find](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 343).

See command [<trace>.PROTOcol.list](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 344).

See command [<trace>.PROTOcol.PROfileChart](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 347).

See command [<trace>.PROTOcol.PROfileSTATistic](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 348).

See command [<trace>.PROTOcol.STATistic](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 350).

LOGGER.REF

Set reference point for time measurement

See command [<trace>.REF](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 357).

LOGGER.RESet

Reset command

See command [<trace>.RESet](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 357).

LOGGER.SAVE

Save trace for postprocessing in TRACE32

See command [<trace>.SAVE](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 358).

LOGGER.SelfArm

Automatic restart of trace recording

See command [<trace>.SelfArm](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 362).

LOGGER.SIZE

Define buffer size

See command [<trace>.SIZE](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 373).

LOGGER.SnapShot

Restart trace capturing once

See command [<trace>.SnapShot](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 373).

LOGGER.state

Display trace configuration window

See command [<trace>.state](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 376).

LOGGER.STATistic

Statistic analysis

See command [<trace>.STATistic](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 378).

See command [**<trace>.Timing**](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 499).

See command [**<trace>.TRACK**](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 502).

See command [**<trace>.View**](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 504).

See command [**<trace>.ZERO**](#) in 'General Commands Reference Guide T' (general_ref_t.pdf, page 505).

The TRACE32 Lua API allows the user to load and execute Lua scripts directly in the debugger. This feature can be used to accelerate execution of certain debug commands by avoiding the interaction between the TRACE32 host software and the debug driver. A Lua interpreter is built into the debug box, supporting the complete Lua language. Please refer to the official website of Lua www.lua.org for documentation.

In addition to the standard language elements, Lauterbach extended Lua with a set of TRACE32 specific libraries. This allows you to, for example, use the JTAG shift functions directly in the Lua script. For a description of the TRACE32 specific libraries, refer to **“TRACE32 Lua Library”** (lua_library.pdf).

The TRACE32 host software maintains an input buffer with size 0x1000 bytes to store input parameters for the Lua script. While sending the command to execute a certain Lua script, TRACE32 also packs and sends the input buffer to the debugger. Use **LUA.Data.SET** to manipulate the input buffer and **LUA.Data.ShowInput** to view the current content in the input buffer. Loading the input buffer from a binary file is also supported. From within the Lua scripts, the data in the input buffer can be retrieved using functions from the **“TRACE32 Lua Library”** (lua_library.pdf).

When executing a Lua script, the TRACE32 host software blocks until it returns. Inside the Lua script, functions from the **“TRACE32 Lua Library”** (lua_library.pdf) can be used to store output data into an output buffer (again 0x1000 bytes). The output buffer will be automatically sent back to the TRACE32. Use **LUA.Data.ShowOutput** to view the output data. It is also supported to save the output buffer into a binary file.

The Lua API can be used both as TRACE32 commands and through the remote C API. The **LUA** command group is described in the following sections and the C API is introduced in chapter **Remote Lua API**.

LUA.Data.Loadinput

Load content from a file into the input buffer

Format: **LUA.Data.Loadinput** <file> [/<load_option>]

Load content from a binary file into the input buffer. Use **LUA.Data.ShowInput** to check the current content of the input buffer.

Offset <offset>	Starting position in the binary file to be loaded into the input buffer
Length <length>	Number of bytes to be loaded into the input buffer

Format:

LUA.Data.Saveoutput <file> [/<save_option>]

Save content of the output buffer into a file. Use [LUA.Data.ShowOutput](#) to check the current content of the output buffer.

Append

Append data to the end of the output file.

LUA.Data.SET

Modify the Lua input buffer

Format:

LUA.Data.SET <index> %<format> <value>

<format>:

Byte | Word | Long | Quad | TByte | HByte | PByte | SByte
BE | LE

Writes byte-wise data to the input buffer. Use [LUA.Data.ShowInput](#) to view the current content of the input buffer. The input buffer will be sent to the debugger upon executing a [LUA.Program.RUN](#) command.

<index>	The byte position of the input buffer to be written.
Byte, Word, ...	Data size. • Byte (8-bit accesses) Word (16-bit accesses) • TByte (24-bit accesses) Long (32-bit accesses) • PByte (40-bit accesses) HByte (48-bit accesses) • SByte (56-bit accesses) Quad (64-bit accesses) •
BE, LE	Define byte endianness: big endian or little endian.
<value>	The value to be written to the buffer.

Format:	LUA.Data.ShowInput
---------	--------------------

Displays the current content of the input buffer in the [AREA](#) window.

Format:	LUA.Data.ShowOutput
---------	---------------------

Displays the current content of the output buffer in the [AREA](#) window. The output buffer contains the return values from the Lua script. Inside the Lua script, functions from the “[TRACE32 Lua Library](#)” ([lua_library.pdf](#)) can be used to write to the output buffer.

Format:	LUA.Program.List
---------	------------------

Lists the Lua scripts that have been loaded into the debugger. The output is redirected to the [AREA](#) window and has the following format: `<id> : <label>, <attribute>, <file>`. See [example](#).

ID	An auto-generated sequential index of the current Lua scripts. Note that the ID of a certain script may change after loading/unloading another script.
label	A unique string to identify a Lua script.
attribute	EXE: an executable script. LIB: a library script. See LUA.Program.LOAD for more details.
filename	Path to the Lua file.

Format:	LUA.Program.LOAD <file> [<label>] [/<options>]
<option>:	Program Library

Loads the Lua script to the debugger without executing it. The script is uniquely identified by its label. See [example](#) here.

<file>	Path and file name of the Lua script to be loaded.
<label>	A unique string to identify the Lua script. If not specified, a default label will be generated.
Program	The Lua script is loaded as an executable script. This is the default option.
Library	The Lua script is loaded as a library script. The Lua functions in a library script will be made available to all other Lua scripts. Although marked as a library, the script itself can still be executed normally. However, we recommend to develop separate Lua scripts for libraries and executables.

LUA.Program.RESet

Reset the Lua context

Format:	LUA.Program.RESet
---------	--------------------------

Resets the Lua context, unloads all scripts from the debugger and clears all input and output buffers.

LUA.Program.RUN

Execute a Lua script

Format:	LUA.Program.RUN <id> <label>
---------	---------------------------------------

Executes the Lua script with given index or label. Use [LUA.Program.List](#) to check IDs and labels of the Lua scripts that are currently available in the debugger.

See [example](#).

Format: **LUA.Program.UNLOAD** <id> | <label>

Removes a Lua script from the debugger. Use [LUA.Program.List](#) to see the scripts currently loaded.

```
; load the Lua script lib.lua as a library
LUA.Program.LOAD c:\lua\lib.lua "mylib" /Library
; load the Lua script jtag.lua as an executable
; use default option "/Program" and default label
LUA.Program.LOAD c:\lua\jtag.lua

; see the current list of Lua scripts
LUA.Program.List
; you should see the following:
; 0 : mylib, LIB, C:\lua\lib.lua
; 1 : jtag.lua, EXE, C:\lua\jtag.lua

; set input parameter
LUA.Data.SET 0x0 %1 0x12345678
LUA.Data.ShowInput

; execute the Lua script jtag.lua using its index
LUA.Program.RUN 1
; execute the Lua script jtag.lua using its label
LUA.Program.RUN "jtag.lua"

; view the output buffer
LUA.Data.ShowOutput

; remove a Lua script
LUA.Program.UNLOAD 0
; Note that now the Lua script with index 0 (lib.lua) is removed
; and the indexing has changed
LUA.Program.List
; now you should see the following
; 0 : jtag.lua, EXE, C:\lua\jtag.lua
; the Lua script jtag.lua now has the index 0
LUA.Program.UNLOAD "jtag.lua"

; clear the context
LUA.Program.RESet
```