

CoreSight™ Access Tool (CSAT)

User Guide



Proprietary Notice

Words and logos marked with ® or ™ are registered trademarks or trademarks owned by ARM Limited, except as otherwise stated below in this proprietary notice. Other brands and names mentioned herein may be the trademarks of their respective owners.

Neither the whole nor any part of the information contained in, or the product described in, this document may be adapted or reproduced in any material form except with the prior written permission of the copyright holder.

The product described in this document is subject to continuous developments and improvements. All particulars of the product and its use contained in this document are given by ARM in good faith. However, all warranties implied or expressed, including but not limited to implied warranties of merchantability, or fitness for purpose, are excluded.

This document is intended only to assist the reader in the use of the product. ARM Limited shall not be liable for any loss or damage arising from the use of any information in this document, or any error or omission in such information, or any incorrect use of the product.

Confidentiality Status

This document is Open Access. This document has no restriction on distribution.

Web Address

<http://www.arm.com>

Contents

1	ABOUT THIS DOCUMENT	7
1.1	Change control	7
1.1.1	Current status and anticipated changes	7
1.1.2	Change history	7
1.2	References	7
1.3	Terms and abbreviations	7
2	SCOPE	8
2.1	RealView ICE and Trace compatibility	8
3	INTRODUCTION	9
3.1	Getting Started	9
3.1.1	CSAT Exit Codes	10
3.2	Memory Based Commands	10
3.3	Configuration Commands	11
3.4	CoreSight Component Commands	11
4	CSAT COMMAND REFERENCE	13
4.1	Scan for Debug Control Units	13
4.2	Connect to Hardware Debug unit	13
4.3	Set JTAG Scan Chain	14
4.3.1	Valid Processor Names for use in the Chain command	14
4.4	Target Device open	14
4.5	Target Device close	15
4.6	Options Command	15
4.6.1	memfile option	15
4.6.2	tracefile and tracebinfile options	15
4.6.3	memprot option	16
4.6.4	autopowerup option	16
4.6.5	test_* options	16
4.6.6	batch_on_error Option	16
4.6.7	no_async_trace_msgs and no_async_rvi_msgs Options	17
4.6.8	retry_conn Option	17
4.6.9	use_dl_fill Option	17

4.7	Enumerate DAP	17
4.8	DAP File load	17
4.9	DAP File save	18
4.10	Disconnect	18
4.11	Exit	18
4.12	Logging	18
4.13	DAP memory read	19
4.14	DAP memory write	19
4.15	DAP Memory Fill	20
4.16	V7 Debug Command	20
4.17	DAP register read and write	21
4.18	System Command	21
4.19	Batch Commands	22
4.19.1	Batch File as a start-up command line parameter	22
4.19.2	Execution Errors when running Batch File Commands	22
4.20	Trace Commands	23
4.21	Abort Commands	23
4.22	Configure ARMCS-DP Template	23
4.22.1	Configuration Items for the ARMCS-DP template	23
4.23	Configure Debug Control Unit	25
4.24	Runtest	25
4.25	Reset Target	25
4.25.1	Control Register Reset	26
4.26	Echo String	26
5	CORESIGHT COMPONENT COMMAND REFERENCE	27
5.1	Common Features	27
5.1.1	Component Command Format	27
5.1.2	Supported CoreSight Components	27
5.2	Standard Register Sub-Commands	27
5.2.1	Set Base Address (baseaddr)	27
5.2.2	List Defined Registers (rlist)	28
5.2.3	Set Instance as Default (def)	28
5.2.4	Register Read (rr)	28

5.2.5	Register Write (rw)	29
5.2.6	Register Read-Modify-Write (rmw)	29
5.2.7	Set AP Index for Component (apidx)	29
5.2.8	Command Help	29
5.3	Extended Commands : ETB component	30
5.3.1	ETB Current Status (status)	30
5.3.2	ETB Start Trace Collection (start)	30
5.3.3	ETB Stop Trace Collection (stop)	31
5.3.4	ETB Read Trace Buffer (read)	31
5.3.5	ETB Dump Trace Buffer to file (dumpbin)	31
5.4	Extended Commands : v7dbg component	32
5.4.1	v7dbg – active	32
5.4.2	v7dbg fastwordaccess	32
5.5	cscomp Command.	33
5.6	Alias Command	34
6	TRACE COMMAND REFERENCE	35
6.1	Trace Support in CSAT	35
6.2	Trace Open	36
6.3	Trace Close	36
6.4	Trace Identify	36
6.5	Trace Getstate	37
6.6	Trace Reset	37
6.7	Trace Start	37
6.8	Trace Stop	37
6.9	Trace Read	38
6.9.1	RVT2 Read Commands	38
6.9.2	DSTREAM Read Commands	38
6.10	Trace Dumpbuffer	39
6.11	Trace Dumpbin	39
6.12	Trace Unit Data Capture and Storage Modes	39
6.13	Trace Configuration Item Commands	39
6.13.1	PORT_WIDTH	41
6.13.2	PORT_MODE	41
6.13.3	CLOCK_MODE	41
6.13.4	ETM_PROTOCOL	41
6.13.5	PACKING	41
6.13.6	SET_TIMESTAMPS	42

6.13.7	TRIGGER_POSITION	42
6.13.8	TRACE_DEPTH	42
6.13.9	SYNC_STRIP	42
6.13.10	CLOCK_EDGE	42
6.13.11	POST_TRIGGER_FILL	42
6.13.12	The DELAY_TRACE_xxx set	43
6.13.13	Diagnostic Config Items	43
6.13.14	DSTREAM Trace Capture Parameters	43
6.13.14.1	BUFFER_FULL_MARKER.	43
6.13.14.2	TRIGGER_RUN_LENGTH	43
6.13.14.3	STOP_AFTER_TRIGGER	43
6.13.14.4	WRAP_ON_FULL	43
6.14	Trace Async	43
7	EXAMPLES	44
7.1	Example session log	44
7.2	Example Batch Command File	45
7.3	Example Setup Script Using Coresight Component Commands	45
7.4	Example Script - Setting PTM to Trace All	46

1 ABOUT THIS DOCUMENT

1.1 Change control

1.1.1 Current status and anticipated changes

Initial Draft – external document from original internal guide. Applicable to version 1.1.5 of CSAT.

Update to coincide with release of CSAT 1.2.0 alongside RVI 3.3.

Update for release of CSAT 2.0.7 with trace command set 2.0.1.1; use with DSTREAM / RVI firmware 4.3.0 and later, and VSTREAM client 3.3c and later.

1.1.2 Change history

See change history in Domino.doc

1.2 References

This document refers to the following documents.

Ref	Doc No	Author(s)	Title
1	ARM DUI 0481E	ARM	DSTREAM, Setting Up the Hardware
2	ARM DUI 0515E	ARM	RVI and RVT, Setting Up the Hardware
3	ARM DUI 0498E	ARM	DSTREAM and RVI, Using the Debug Hardware Configuration Utility
4	DS165-PRDC-011261	ARM	VSTREAM, Client User Guide

1.3 Terms and abbreviations

This document uses the following terms and abbreviations.

Term	Meaning
Debug Control Unit	Hardware (or software virtual equivalent) that creates a connection between the debugger and the target system. Drives the target connection – JTAG, SWD, 'virtual' as required.
Trace Capture Unit	Hardware (or software virtual equivalent) that captures the trace output from the target via a trace connector.
RVI	RealView ICE Debug control unit
CSAT	CoreSight™ Access Tool
RVT2	RealView Trace Capture unit for RVI.
DSTREAM	DSTREAM Debug control and Trace capture Unit
VSTREAM (client)	Virtual Debug control unit.

2 SCOPE

This document describes the commands and usage of the CoreSight Access Tool. This tool is designed for the testing of CoreSight systems using the RealView ICE or DSTREAM hardware, the VSTREAM virtual connection, attached to the test system.

The RealView Trace or DSTREAM hardware is also required if testing involves evaluation of trace output via a trace connector. Trace captured on the target system such as in the CoreSight ETB does not require a trace capture unit.

It is assumed that the reader is familiar with the use of the RVI / DSTREAM products (see Ref 1, 2), the debug hardware configuration utilities (Ref 3), and the RVT2 trace capture unit (Ref 2)

VSTREAM user information can be obtained from the VSTREAM client user guide (Ref 4).

CSAT trace commands can also be used to investigate trace capture issues on none CoreSight systems.

2.1 RealView ICE and Trace compatibility

The CSAT tool is designed to work closely with the RVI/RVT/DSTREAM hardware, and installed firmware versions.

CSAT version 2.0.0 (and later) is delivered as part of RVI/DSTREAM version 4.0 (or later) host side utilities. Functionality has been added, which requires the use of version 4.0 (or later) of RVI/DSTREAM firmware, and version 4.0 of the RVI/DSTREAM host side utilities.

CSAT also connects with the VSTREAM client software, version 3.3c and later.

See the `csat_changeLog.txt` that is supplied with CSAT for updated compatibility, enhancement and bug-fix information.

3 INTRODUCTION

The CoreSight™ Access Tool (CSAT) program is a console based test program for validating CoreSight systems. The tool contains a number of commands to allow access to a CoreSight system, using the RealView ICE emulator hardware, and if required the RealView Trace hardware. CSAT has sufficient commands to control both the RVI and RVT units, in addition to accessing CoreSight components.

The summary of the available commands uses the following syntax. Most commands have a long and short format. These are shown as follows:

```
COMMAND (CMD) opt1 [opt2] opt3 | opt4 opt5=param1{,param2}*
```

The options use '[']' for optional, and '|' for exclusive alternatives, '{ }' to bracket alternative sets. '*' means repeat as required.

3.1 Getting Started

CSAT has versions for both Linux (`csat`) and Windows (`csat.exe`).

First ensure that the debug control unit (RVI/DSTREAM) has the JTAG cable connected you intend to use during the session and power up the unit. In the case of VSTREAM, ensure that the client is running and connected to the virtual target.

Run **CSAT** to see the command prompt `%>`

Commands may have long names plus short versions. For example, the DAP memory read command uses `dpmemread` and `dmr`.

First you need to connect to the debug control unit. The command `connect` (or `con` is the short version) can be used to connect either via TCP or USB (if connected by USB). With TCP use the debug control unit IP address, or the host name (this depends on your local networks DNS server – if the hostname connection fails try IP address). e.g.

```
con TCP:Koyaanisqatsi
con TCP:192.168.23.240
con USB
```

The same command is used to connect to the VSTREAM client, although if the VSTREAM client and CSAT are running on the same PC then the following localhost command works.

```
con TCP:localhost
```

Next the debug control unit needs to get information about devices on the scan chain. This can be done in one of two ways. The debug control unit can autodetect the chain, which succeeds only if all the devices on the chain are recognised. Otherwise the scan chain can be directly specified. Also the JTAG clock frequency must be set at this time. The chain command is used to set this information.

The most common command used is to connect to a CoreSight System with a single DAP on the scanchain.

```
chain dev=ARMCS-DP clk=10000000
```

This example autodetects an adaptively clocked system.

```
chain dev=auto clk=A
```

A typical response to this is:

```
Jtag clock set to 50000000A
```

```
ID:0 ARMCS-DP
```

```
ID:1 ARM1136JF-S
```

This shows two target devices on the scan chain, a DAP (ARMCS-DP) at scan chain ID 0, and an ARM1136 core at scan chain ID 1.

The next example sets the chain for a system that cannot be auto detected, and set the JTAG clock to 10MHz. This sets two target devices, an ARMCS-DP device at scan chain position (ID) 0, and an unknown device at position 1.

```
chain dev=ARMCS-DP,UNKNOWN_4 clk=10000000
```

For the VSTREAM product, the clock parameter is still supplied as part of the command, though it is not actually used.

Now we need to open the required target device on the chain – this makes a connection to the single device, using a device template on the debug control unit, and allows the unit to route the commands correctly. The chain command responds with a list of devices and scan chain IDs. Use the scan chain ID to open the device.

```
dvo 0
```

All commands are now available for use – some would not have worked until the target device is open.

This sequence could be written to a command file and run at program start. See section 7.2 for an example script. Assuming that the script is in a file call start.txt, start CSAT as '`csat start.txt`'. See also () for batch processing operation.

The trace capture unit (RVT2/DSTREAM) can be controlled using the `trace` command. This command has a number of functions, such as `open`, `start`, `stop`, `read` and `close`, which are used to initiate tracing and extract trace data from the unit.

3.1.1 CSAT Exit Codes

CSAT can return a number of codes on exit.

Code	Description
-1	Internal C error. Memory allocation error or some other system issue.
0	Normal Program termination
1	Batch command execution error forced exit. Occurs if a command from batch file is executed and results in an error, and the <code>batch_on_error</code> option is set to <code>exit</code> . See (4.19)
2	A start-up batch file could not be found, or contained a syntax error on parsing. No commands run, CSAT exits with this code.

3.2 Memory Based Commands

All memory based commands require the selection of the appropriate Access Port (AP) on the DAP. A typical set up has an AHB-AP connected to the system memory, and an APB-AP connected to the debug-APB. A DAP can have up to 256 APs connected, which are accessed by an index number in CSAT. This is the first parameter on all memory based commands.

The memory based commands are:

GENC-008687 v5.0

Copyright ©2011 ARM Limited. All rights reserved.

Command	Short name	Description
dpmemread	dmr	Read memory.
dpmemwrite	dmw	Write memory.
dpfileload	dfl	Load a file into memory.
dpfilesave	dfs	Save a block of memory to file.

```
dmr 0 0x13000000 4          Read from AP 0, 4 words at address 0x13000000.
dmw 1 0x12500000 0x7f      Write to AP 1, 1 word value 0x7f to address 0x12500000.
dfl 0 bin 0x8000 myfile.bin Load a binary file into 0x8000 on AP 0.
```

See the `dapenum` command for details on how to discover the AP indexes for your DAP system.

The memory commands can also use the `v7dbg` option in place of the AP index. This routes the memory command via the V7 debug support built into the device template. This allows memory to be accessed using the core debug features rather than the APs. This is useful for systems where the an AP is not attached to some or all of the system memory. See the `v7dbg` command for setting up this support.

3.3 Configuration Commands

The CoreSight Access Tool consists of three elements. The `csat` executable on the host is the command line interface for the user. The debug control unit (RVI/DSTREAM/VSTREAM) makes the physical connection to the target, and the Core template, hosted on the debug hardware unit translates commands into actions on the target. CSAT, RVI and the Template all have items which are configurable. There are three configuration commands:

```
options      - configures CSAT.
cfgbox       - configures the debug hardware unit.
cfigtplate   - configures the core template.
```

3.4 CoreSight Component Commands

CSAT supports register based access to CoreSight components on the debug bus. The register access permits easier access to memory mapped components.

The default AP for the CoreSight system may be set, then each component in the system is set to have a memory address. Multiple instances of each component may be set up, up to 10 instances per component type.

Once components are set up then the registers can be accessed by name, rather than having to remember the address and offset for memory based registers.

e.g. Script to set up and access to a multi core CoreSight system

```
# Set default AP index for the debug bus on the APB-AP
cscomp def_apidx 1

# set component base addresses
v7dbg.0 baseaddr 0x80001000
v7dbg.1 baseaddr 0x80003000
v7dbg.2 baseaddr 0x80005000
```

```
ptm.0 baseaddr 0x80002000
ptm.1 baseaddr 0x80004000
etm.0 baseaddr 0x80006000
ctf baseaddr 0x80007000
etb baseaddr 0x80008000
tpiu baseaddr 0x80009000
```

```
# alias for cores
alias v7dbg.0 a9_0
alias v7dbg.1 a9_1
alias v7dbg.2 a5
```

```
# register access - stop the A5
a5 rr prsr
a5 rr prsr
a5 rr dscr
a5 rmw dscr 0x00004000 0x00004000
a5 rw prcr 0x1
a5 rr dscr
```

See section 5 for a reference for these commands.

4 CSAT COMMAND REFERENCE

4.1 Scan for Debug Control Units

rviscan (rvs)

Scans for RVI, DSTREAM and VSTREAM debug control units on the network and connected via USB. Scans for 10 seconds.

```
%>rviscan
RVIScan started...
>> TCP:arm03965xp          10.33.0.118      VSTREAM ; 0 active connections
>> TCP:DS-noisyfan        10.33.0.108      DSTREAM ; 0 active connections
>> TCP:DS0000000556       10.33.0.211      DSTREAM ; 0 active connections
>> TCP:ds-manaslu         10.33.0.152      DSTREAM ; 0 active connections
>> TCP:DS-Noodle          10.33.0.212      DSTREAM ; 0 active connections
>> TCP:DS-Haystacks       10.33.0.161      DSTREAM ; 0 active connections
>> TCP:rvi-dave           10.33.0.226      RVI      ; 0 active connections
>> TCP:rvi-splodge        10.33.0.224      RVI      ; 1 active connections
>> TCP:admiral-ackbar     10.33.0.231      RVI      ; 0 active connections
>> TCP:SevenZark7         10.33.0.169      RVI      ; 0 active connections
>> TCP:Koyaanisqatsi      10.33.0.221      RVI      ; 0 active connections
>> USB:109330168          10.33.0.221      RVI      ; 0 active connections
>> TCP:DS-Daedalus        10.33.0.150      DSTREAM ; 2 active connections
>> TCP:DS-Enterprise      10.33.0.179      DSTREAM ; 0 active connections
RVIScan : Scan complete.
%>
```

Note: The scan tool only searches for run control units that are connected to your local network, so units that are connected to separate subnets will not be found. Consequently, if you want to connect to an RVI unit on a separate network, you must ensure that you know the IP address of that network.

4.2 Connect to Hardware Debug unit

```
connect (con) TCP:<hostname> | TCP:<ip address> | USB | USB:<serial_no>
```

Connect to a debug control unit. Specify TCP or USB. USB attaches to the first USB RVI/DSTREAM found on the local system, USB:<serial_no> attaches to a USB unit with the given number. The TCP option can specify either the hostname or the IP address of your debug control unit. VSTREAM cannot use USB, but can use localhost if VSTREAM client and CSAT are running on same system.

Examples from the list above:

```
con TCP:MIKED
con TCP:192.168.23.172
con USB
con USB:109330168
```

```
con TCP:localhost
```

4.3 Set JTAG Scan Chain

```
chain (chn) [dev=auto | dev=? | dev=DEVICE_NAME{,DEVICE_NAME}*] [clk=<FreqHz> |
clk=A]
```

Scan chain setup. Some setup is usually required before connecting to a target, though if another client such as a debugger is connected to the target system then the chain may already be set. Dev=auto autodetects the scan chain. Dev=? displays the current setup of the chain. Clk sets the JTAG clock speed, either a frequency in Hz, for example `clk=10000000` for 10MHz, or A for adaptive clocking.

Autodetect and adaptively clocked device:

```
e.g. %> chain dev=auto clk=A
```

Set a known JTAG chain, at 20MHz JTAG clock, with a CoreSight DAP and an unknown device, IR length 4.

```
e.g. %> chain dev=ARMCS-DP,UNKNOWN_4 clk=2000000
```

4.3.1 Valid Processor Names for use in the Chain command

Below is a set of names for processors which can be used in the chain command. These names are associated in RVI with processor templates. Not all names are available in all builds of the debug control unit firmware. Where items exist on the JTAG scan chain which are not included in the names below, the designation UNKNOWN_N may be used, where N is the length of the IR register. This allows RVI to put the unknown item into bypass, while accessing the known items normally.

CoreSight : ARMCS-DP

This template is the principle template used by the CSAT tool. The `dvo` command (see section 4.4) should refer to this template.

The following templates may appear on a scan chain that is already set, or can be used when defining a scan chain, but cannot be opened using the CSAT `dvo` command.

Arm 7 Family : ARM7EJ-S, ARM710T, ARM720T, ARM720T_r4, ARM740T, ARM7TDMI, ARM7TDMI-S, ARM7TDMI_r4, ARM7TDMI-S_r4.

Arm 9 Family : ARM966E-S, ARM940T, ARM946E-S, ARM926EJ-S, ARM9SMP-Eval, ARM925T, ARM922T, ARM920T, ARM9E-S, ARM9EJ-S, ARM9TDMI.

Arm 10 Family : ARM1020E, ARM1022E, ARM10200E, ARM10220E, ARM1026EJ-S

Arm 11 Family : ARM1136J-S, ARM1136JF-S, ARM1156-S, ARM1156F-S, ARM1176JZF-S

Secure Core : SC100D, SC200D

Arm Misc : ETMBUF, MPCore,

Note: Cortex cores, or devices accessed through the DAP never appear on the scan chain as part of the chain command.

4.4 Target Device open

```
devopen (dvo) <devid>
```

Open a device on the scan chain. The <devid> is the ID displayed by the chain command. Only one device can be open at any one time.

e.g. %> dvo 0

4.5 Target Device close

devclose (dvc)

Close current device connection.

4.6 Options Command

options (opt) [<option_name> <option_value>]

The option command allows default parameters to be changed from the initial program defaults to user defined defaults. The <option_name> parameter is the name of the option to set. The <option_value> is dependent on the option being set. Running the options command with no name displays all the current options values.

For example:

```
%>opt
```

```
Current Default Option Settings:-
```

```
memfile = MEMDMP.BIN
```

```
tracefile = TRACEDMP.TXT
```

```
tracebinfile = TRCDMPBIN.TRB
```

```
memprot = 0x43
```

```
autopowerup = true
```

```
test_timeout = 0x000493E0
```

```
test_baseaddr = 0x12004000
```

```
test_termval = 0x00000004
```

```
test_emptyval = 0xF1F00000
```

```
test_apidx = 0x00
```

```
test_resetlo_ondone = false
```

```
batch_on_error = halt
```

```
no_async_trace_msgs = 0x00
```

```
no_async_rvi_msgs = 0x00
```

```
retry_conn = 1
```

```
use_dl_fill = true
```

4.6.1 memfile option

The `memfile` option defines the default file used when loading and saving binary memory files. See `dpfload` and `dpfsave` commands.

4.6.2 tracefile and tracebinfile options

The `tracefile` option defines the default file used when dumping trace data to a text file on the host system. See the `trace dumpbuffer` command. The `tracebinfile` option defines the default filename used when dumping a binary trace data file with the `trace dumpbin` command.

4.6.3 memprot option

The `memprot` option, is used when accessing memory through a bus AP – such as the AHB-AP or APB-AP. These AP types have access control bits as the most significant byte of the AP CSW register. The `memprot` option value is an 8 bit value which is used by memory commands using an AP to set the MS-byte of the AP CSW register. Dependent on the AP in use, only certain bits in this value are significant. See the CoreSight documentation for further details.

The `memprot` value defaults to 0x43, which is the default value for the MS byte in the AHB-AP CSW register. All memory commands (`df1`, `dfs`, `dmr`, `dmw`) use the `memprot` value when accessing memory.

4.6.4 autopowerup option

The `autopowerup` option controls if the DAP attempts to use the power control bits in the DPACC.CSW register to power up the system on device open (`dv0`) and power down the system on device close (`dv1`). Defaults to true.

4.6.5 test_* options

The `test_*` options control the default values and operation of the ‘runtest’ command. See this command for more information.

`test_timeout` - the timeout before the runtest command aborts.

```
%> options test_timeout 300000
```

`test_baseaddr` - the address of the Tube FIFO register to monitor

```
%> options test_baseaddr 0x12004000
```

`test_apidx` - the AP index of the DAP AP that the Tube FIFO is attached to.

```
%> options test_apidx 0
```

`test_termval` - FIFO read value runtest uses to determine test is complete

```
%> options test_termval 0x00000004
```

`test_emptyval` - FIFO read value runtest uses to determine FIFO is empty

```
%> options test_emptyval 0xF1F00000
```

`test_resetlo_ondone` - controls if reset is asserted when test completes.

```
%> options test_resetlo_ondone false
```

4.6.6 batch_on_error Option

This option controls the action of CSAT when an error occurs executing a set of commands from a batch file, either the start-up batch file, or as a result of a batch command. There are three possible values for the `batch_on_error` option

`halt` - this is the default. Results in all remaining commands from the batch file being cancelled and control returns to the command line.

`cont` - The error is ignored. The remaining commands are executed.

`exit` - All remaining commands are cancelled and CSAT terminates as though an exit command had been executed.

```
%> options batch_on_error exit
```

4.6.7 no_async_trace_msgs and no_async_rvi_msgs Options

These options suppress the asynchronous messages that appear from the RVT and RVI units respectively.

Useful during test runs, as the timing of the appearance of these messages can be unpredictable, and result in a different text log.

4.6.8 retry_conn Option

Determines how many times CSAT attempts to connect to the requested target.

4.6.9 use_dl_fill Option

The file load operations can use template download and fill commands to increase efficiency of download operations – on version 3.3.x firmware and above. This option available on **CSAT 1.2.0** and above.

If connected to an older firmware version, then this option automatically sets itself to false as soon as an invalid download or fill command is attempted and revert to the older file load mechanism.

4.7 Enumerate DAP

```
dapenum (dpe)
```

Scans all AP locations and build list of available APs on the DAP. Numbers shown are the index numbers to use in commands which require an AP index.

```
%>dapenum
```

```
Enumerated 3 APs
```

```
0 : AHB-AP
```

```
1 : APB-AP
```

```
2 : JTAG-AP
```

4.8 DAP File load

```
dpfload (dfl) <ap_no> [memlog] { elf [execload] } | { bin <addr> } [filename]
```

Load either a binary or an elf file into the target memory.

`<ap_no>` is the AP index (0 – 255) used to access the memory, or can be `v7dbg` to route the memory access via v7 debug.

The `elf` or `bin` parameters control the type of file loaded. The elf file contains information on memory locations to load.

When loading a binary file, an `<addr>` parameter is supplied to define the start address for the load. If no filename is supplied then the default `MEMDMP.BIN` is used. This default filename can be altered using the `options memfile` command.

The `memlog` option is used to force CSAT to log the addresses that the load process used.

The `execload` option is used in the elf files to force the execute time addresses to be used for load, rather than the default load time addresses. Sometimes used if executable does not contain scatter-loading code.

`v7dbg` routes the memory command via the v7 debug support on the core. See the `v7dbg` command for options to configure this support.

e.g. `%> df1 0 bin 0x8000 mydata.bin`

Load `mydata.bin` into the memory accessed by AP0, at location 0x8000.

e.g. `%> df1 v7dbg elf myprog.axf`

Load elf program `myprog.axf` into the memory accessed via v7 debug.

The memory access privileges may be affected by the `memprot` option. See the `options memprot` command.

4.9 DAP File save

`dpfsave <ap_no> <addr> <nWords> [filename]`

Save a block of memory to a file. The memory and length are defined by the `<ap_no>` (AP index / `v7dbg`), `<addr>` (memory address) and `<nWords>` (number of 32bit words) parameters. If no filename is supplied then the default `MEMDMP.BIN` is used. This default filename can be altered using the `options memfile` command.

The memory access privileges may be affected by the `memprot` option. See the `options memprot` command.

`v7dbg` routes the memory command via the v7 debug support on the core. See the `v7dbg` command for options to configure this support.

4.10 Disconnect

`disconnect (dcn)`

Disconnect from RVI. Executes `devclose` if currently attached to a device.

4.11 Exit

`exit`

Close CSAT program. Executes `disconnect` if still connected to a RVI unit.

4.12 Logging

`log on | off | <filename>`

Logging is on by default, and all commands are logged to CSAT.LOG. Logging can be turned on or off, or a new logfile name for the session may be specified. CSAT.LOG is overwritten each time CSAT is started.

4.13 DAP memory read

```
dpmemread (dmr) <ap_no> <address>[.{b|B|h|H}] <no. items>
```

Read some memory. <ap_no> is the AP index (0 – 255), or the **v7dbg** option. <address> is the address to access. Access size defaults to word access, but a suffix of .b, .B, .h or .H chooses byte or half word access. <no. items> defines the number of elements of the given size. Addresses must be aligned to element access size. **v7dbg** routes the memory command via the v7 debug support on the core. See the **v7dbg** command for options to configure this support.

Example reads 4 words from address location 0x8100.

```
e.g. %> dmr 0 0x8100 4
0x00008100 : 0x12601E30 0x88E0A9CA 0x1AC23260 0x34A6FB79
```

Example reads 7 bytes from address location 0x9101.

Note: CSAT always displays from word aligned addresses, so the location not read is shown as '----'.

```
e.g. %> dmr 0 0x9101.B 7
0x00009100 : ---- 0x1E 0xEF 0xB2 0x52 0x87 0x4E 0x49
```

Example reads 3 half words from address location 0x8042 using the v7 debug.

```
e.g. %> dmr v7dbg 0x8042.h 3
0x00008040 : ----- 0xF8A5 0xAE3D 0xA251
```

The memory access privileges may be affected by the `memprot` option. See the `options memprot` command.

4.14 DAP memory write

```
dpmemwrite (dmw) <ap_no> <address>[.{b|B|h|H}] <data> [<data>]*
```

Write some memory. <ap_no> and <address> as above. <data> in decimal or 0x hex format. Access size defaults to 32 bit words, but can be set to byte or halfword accesses using the suffixes on the address value as above. Data is interpreted according to access size. 0x00 is a single byte, half word or word. The value 0x123456 is interpreted as 1 word 0x00123456, 2 half words 0x3456, 0x0012 or 3 bytes, 0x56, 0x34, 0x12.

v7dbg routes the memory command via the v7 debug support on the core. See the **v7dbg** command for options to configure this support.

Example writes 4 32 bit words, 0x00000001, 0x00003444, 0x00000004 and 0x00000007 to address location 0x8000.

```
e.g. %> dmw 0 0x8000 1 0x3444 4 7
```

The same data as bytes writes 5 bytes to 0x8000, 0x01, 0x44, 0x34, 0x04, 0x07.

```
e.g. %> dmw 0 0x8000.B 1 0x3444 4 7
```

Memory write via v7 debug:

e.g. %> `dmw v7dbg 0x8000 0x12345678 0x87654321`

The memory access privileges may be affected by the `memprot` option. See the `options memprot` command.

4.15 DAP Memory Fill

`dpmemfill (dmf) <ap_no> <address>[.{b|B|h|H}] <pattern> <iterations>`

This is a new command available with CSAT 1.2.x and RVI firmware 3.3.x. It is not available if the `use_dl_fill` option is set to false.

The command fills memory according to `<ap_no>` and `<address>`, defined as per the `dpmemwrite` command.

Pattern is a decimal or hex value of up to 32 bits, used according to the size parameter on the address. Iterations determines how many times the pattern is repeated. The number of bytes actually filled depends on both the size and number of iterations.

e.g. %> `df1 v7dbg 0x0000 0xEAFFFFFEE 8`

fills the first 8 words at address 0x0000 with the pattern 0xEAFFFFFEE.

e.g. %> `df1 v7dbg 0x8000.b 0x123 7`

fills the first 7 bytes at address 0x8000 with the pattern 0x23.

4.16 V7 Debug Command

`v7dbg (v7d) [apidx=<N>] [baseaddr=<0xNNNN>] [fastwordaccess={true|false}]`

This command allows the user to define the parameters of the v7 debug module in the CoreSight system to be used when the `v7dbg` option is used with relevant memory commands. The `apidx` is the AP index the v7 debug is connected to, the `baseaddr` is its address on that AP, and the `fastwordaccess` parameter can be true or false to enable fast access method of DTR read write (see CoreSight documentation for further information).

Run without parameters, the current values are displayed.

e.g. %> `v7dbg`

`v7dbg : apidx=0, baseaddr=0x00000000, fastwordaccess=false`

Any or all of the parameters can be set at one time:

e.g. %> `v7dbg apidx=1 baseaddr=0x80001000 fastwordaccess=true`

sets all parameters,

e.g. %> `v7dbg fastwordaccess=false`

sets just one.

This command must be run once before any memory commands which use the `v7dbg` option are used. Once the command is run, then the v7 debug parameters are used in all subsequent memory commands using the `v7dbg` option.

Note: *this version of the v7dbg command has been superceded by the v7dbg component in the CoreSight components command group. The above syntax still works, and is applied to the default v7dbg object instance. Memory commands such as dmr and dmw that use v7dbg as a parameter use the currently **active** v7dbg instance.*

See section 5.4 for more information.

4.17 DAP register read and write

```
dpregread (drr) <base name>.<index/name>
dpregwrite (drw) <base name>.<index/name> <data>
```

Read or write a DAP register. Base name can be **DPACC**, **APACC** or **APn** (where n is 0 – 255 – e.g. AP3). **DP** and **AP** can be used in place of **DPACC** and **APACC** respectively.

Index/name is dependent on the base name.

For **DPACC** (or **DP**) the following are valid index/names: **CSW**, **ADDR**, **RDBUFF**, and indexes 1 – 3. Any of the DPACC registers can be accessed.

For **APACC** (or **AP**) the following are valid index/names: 0 – 3. Any of the 4 AP registers for the current AP address can be accessed.

For **APn** the following are valid index/names: **CSW**, **TAR**, **DRW**, **BD0**, **BD1**, **BD2**, **BD3**, 0 – 7.

The APn.index/name combination sets the appropriate AP address before accessing the register. Thus the APn register name can access the bottom 8 registers of any AP. These are the most often used in the current APs (JTAG, AHB, APB).

Register names can be lower case.

e.g.

Write a couple of words to memory using register combinations...

```
drr AP0.CSW
drw ap0.csw 0x43800022
drr AP0.TAR 0x8100
drw AP0.DRW 0x12345678
drw ap0.drw 0x87654321
```

Combinations of DPACC and APACC registers can access any register on the DP or any register on any AP.

e.g. Access the ROM address and ID registers for AP 1.

```
drw DP.ADDR 0x010000F0
drr AP.2
drr APACC.3
```

4.18 System Command

CSAT can execute an operating system command using the **system** or **:** commands.

e.g. 'dir' command of local file system

```
%>: dir
Volume in drive C is WINDOWS XP
Volume Serial Number is A457-1CD4

Directory of C:\ProgFiles\CSAT

20/12/2007  13:37    <DIR>          .
20/12/2007  13:37    <DIR>          ..
11/11/2005  11:57                40,960 boost_thread-vc71-mt-1_31.dll
20/12/2007  13:45                229,376 csat.exe
```

```

20/12/2007  14:12          4,221 CSAT.LOG
20/12/2007  11:25          53,248 rvicomms.dll
20/12/2007  14:06          217,088 rvt_cli_cmds.dll
25/07/2007  15:20           2,604 startup.txt
           6 File(s)          564,106 bytes
           3 Dir(s)  15,380,971,520 bytes free

```

```
%>
```

Note: results of these external commands do not appear in the CSAT log file.

4.19 Batch Commands

```
batch <filename>
```

Run a batch of commands from a file. File format is a text file with lines which can contain the following:

- Any valid command. Batch files cannot call the `batch` command.
- A blank line.
- A line with a `#` character. The `#` and anything after it on the line are treated as a comment. Comments can appear after commands, or on an otherwise blank line.

e.g. `%> batch c:\startup.txt`

Batch files are parsed completely to create a list of executable commands, which are only executed after the file processing is complete. If a file contains a syntax error, then none of the commands in the file are executed.

Once the batch file has been parsed and a command list created then the commands are executed. Action on errors during batch command execution are described below (see section 4.19.2).

4.19.1 Batch File as a start-up command line parameter

`csat` can be started with a batch command file as a parameter. This is a start-up batch file.

e.g. `%> csat c:\startup.txt`

This has the same effect as entering `'batch c:\startup.txt'` as the first command at the command prompt. If the `startup.txt` file cannot be found, or contains syntax errors then no commands are run, and CSAT exits with an error code.

Once the start-up batch file has been parsed and a command list created then the commands are executed. Action on errors during batch command execution are described below (see section 4.19.2).

4.19.2 Execution Errors when running Batch File Commands

The operation of CSAT when a command in a batch file fails with an error is controlled by the `batch_on_error` option. This is set as follows:

```
%> options batch_on_error exit
```

Possible values are:

`halt` - this is the default. Results in all remaining commands from the batch file being cancelled and control returns to the command line.

`cont` - The error is ignored. The remaining commands are executed.

`exit` - All remaining commands are cancelled and CSAT terminates as though an exit command had been executed with return code 1. (See 3.1.1 for CSAT return codes.)

4.20 Trace Commands

`trace (trc) <function> [options]`

This command has a set of functions used to control the RVT2 trace capture unit when attached to the RVI debug control unit, or the trace capture functionality in the DSTREAM unit. The functions available are described in the trace command reference below.

The command line

```
%> trace help
```

lists possible trace functions.

4.21 Abort Commands

`abort`

`abortall`

The `abort` and `abortall` commands cancel the current and current plus all remaining commands respectively. The current command is aborted only if it operates asynchronously. This currently applies only to the `runtest` command.

4.22 Configure ARMCS-DP Template

`cfgtplate (cfg) <item_name> [<item_value>]`

Some templates require configuration of certain template specific items before they can be used. This command enables that configuration. `<item_name>` is the name of the configuration item, `<item_value>` is the string value. If `<item_value>` is omitted then the current value for the item is displayed. The `cfg` command operates on the template connected using the `dvo` command.

A full list of available configuration items for the template can be obtained by reading the special "CONFIG_ITEMS" configuration item:-

```
%> cfg CONFIG_ITEMS
```

lists all the template configuration items. See the configuration tool documentation (Ref 3) for further information.

4.22.1 Configuration Items for the ARMCS-DP template

There are a number of significant configuration items for the DAP template when using CSAT:-

RESET_REG_DATA

This sets the parameters for the reset register. The value is a string of comma separated values of the format:

```
<ap_no>,<base_addr>,<assert_value>,<deassert_value>,<mask>,<hprot_rst_regs>
```

All values are in hex.

`<ap_no>` = AP number for the memory mapped reset register logic.

<base_addr> = 32 bit base address value for register set containing the reset register.
 <assert_value> = 32 bit data value when asserting reset.
 <deassert_value> = 32 bit value when de-asserting reset.
 <mask> = 32 bit mask of bits not to be altered in the register. (set bit to 1 = do not allow change to bit value.).
 <hprot_rst_regs> = 8 bit value for the HProt value used when accessing the reset register values.

e.g. read then set the parameters:

```
%>cfg RESET_REG_DATA 1,80003094,4,0,4,80
Configuration item : RESET_REG_DATA set.
%>cfg RESET_REG_DATA
Configuration item : RESET_REG_DATA is : 01,80003094,00000004,00000000,00000004,80
```

If only the first n parameters are supplied when setting, the rest remain the same:

```
%>cfg RESET_REG_DATA 1,80003098
Configuration item : RESET_REG_DATA set.
%>cfg RESET_REG_DATA
Configuration item : RESET_REG_DATA is : 01,80003098,00000004,00000000,00000004,80
```

USE_TB_RST_OFFSETS

These are parameters for the Validation Trickbox reset scheduler (only applicable if the trickbox IP has been built into the system).

```
e.g. %>cfg USE_TB_RST_OFFSETS
Configuration item : USE_TB_RST_OFFSETS is : 01,00000520,00000018
```

<use_tb_sched> = 1 if trickbox reset scheduling in use, 0 if not.
 <reset_reg_offset> = 0x520, offset from <base_addr> for the reset register.
 <reset_sched_offset> = 0x018, offset from < base_addr> for the schedule register.

By default, the DAP template implements a `ctr1` register reset (see 4.25) assuming a validation configuration with a reset control register at 0x520 offset from the base address of the register set, and a reset request register 0x018 offset from the base. This is the configuration of the current validation trickbox.

If reset without the trickbox is required, or offset values are different then this config item should be changed. To disable the trickbox reset and use a normal register reset:

```
%>cfg USE_TB_RST_OFFSETS 0
Configuration item : USE_TB_RST_OFFSETS set.
```

IGNORE_BUS_BUSY

The template checks the AP status register bits (6 & 7) to establish if a transaction is in progress, and that the bus is enabled, before undertaking a memory transaction across the bus connected to the AP. If a transaction is in progress (bit 7 = 0) or the AP is disabled (bit 6 = 0) then the operation is rejected. This is seen in CSAT as an error in a memory command. This configuration item when set to '1' forces the memory operations to ignore the status of these bits. The example sets the configuration item.

e.g. `%> cfg IGNORE_BUS_BUSY 1`

The item is normally set to '0'

The remaining config items are used as part of other CSAT commands or are not relevant when connected with CSAT and must not be altered:-

DCOMMS_CHANNEL_ID, MEMORY_ACCESS_AP, DAP_CONFIG_INFO, V7DBG_CONFIG, DAP_POWER_UP and IGNORE_POWERUP_FAIL.

These items exist at the time of writing, but may vary with versions of debug control unit.

4.23 Configure Debug Control Unit

```
cfgbox (cfb) <item_name> [<item_value>]
```

The debug control unit has a number of configurable items which affect the operation of the unit and certain functions such as reset. Most of these do not affect the operation of CSAT, other than the reset timing parameters which are used when system reset is requested.

A list of some of the configurable items which could be used in CSAT is shown below:

Name	Description
ResetOperation	Overridden by the CSAT reset command
ResetHoldTime	Time in milliseconds system reset is asserted (if complete cycle)
PostResetDelay	Time in milliseconds before further operations on template after reset.

A full list of available configuration items for the debug control unit can be obtained by reading the special "CONFIG_ITEMS" configuration item:-

```
%> cfb CONFIG_ITEMS
```

lists all the box items. See the configuration tools documentation (Ref 3) for further information.

4.24 Runtest

```
runtest (rnt) [timeout=<milliseconds>] [termval=<val>] [apidx=<ap_no>]
[emptyval=<val>] [baseaddr=<addr>]
```

Runtest has a specific set of operations, designed for validation of core on a test FPGA with appropriate additional validation hardware. It first de-asserts system reset, and then start monitoring the supplied address as the Tube FIFO. If a timeout occurs or the termval is detected, then the test command terminates, asserting system reset.

All parameters are optional, default values are used for any not supplied as follows:

```
apidx      = 0
termval    = 4
emptyval   = 0xF1F00000
timeout    = 300000
baseAddr   = 0x12500000
```

4.25 Reset Target

```
reset (rst) {sys &| ctrl [lo | hi] } | { tap &| dapabort }
```

Reset the target system. Options for system, control register, tap or dapabort resets.

<code>sys</code>	Drive nSRST line on the JTAG connector. Can be used with hi, lo and ctrl.
<code>ctrl</code>	Use the configured reset control register. Can be used with hi, lo and ctrl.
<code>tap</code>	Drive the nTRST line on the JTAG connector. Can be used with dapabort.
<code>dapabort</code>	Set the DAP abort command. Can be used with tap.
<code>hi</code>	De-assert reset level. Can be used with sys and ctrl.
<code>lo</code>	Assert reset level. Can be used with sys and ctrl.

System reset can be timed, according to box settings. Alternatively, a system or control register reset level can be set. `reset sys lo` asserts system reset, `reset sys hi` de-asserts system reset.

If the DAP template is connected then the parameters supplied in the `cfgtplat` command described above is used to implement the reset if a control register reset is used. `reset dapabort` does not cause the nTRST line to be used on the tap controller, but forces the ABORT instruction code into the JTAG-DP IR register.

It is not possible to use the sys, ctrl options with the tap, dapabort options.

4.25.1 Control Register Reset

The following algorithm is used for the control register reset. <name> represent configuration parameter details. (see 4.22.1 for config parameter details).

Reset Assert():

```
rstregaddr = <base_addr>
if ( <use_tb_sched> = 1)
    rstregaddr += < reset_reg_offset>
rstregval = read ( rstregaddr )
rstregval &= <mask>
rstregval |= (<assert_value> & ~<mask>)
write (rstregaddr, rstregval)
if ( <use_tb_sched> = 1)
    write (<base_addr> + <reset_sched_offset>, 0)
```

Reset De-assert():

```
rstregaddr = <base_addr>
if ( <use_tb_sched> = 1)
    rstregaddr += < reset_reg_offset>
rstregval = read ( rstregaddr )
rstregval &= <mask>
rstregval |= (<deassert_value> & ~<mask>)
write (rstregaddr, rstregval)
```

4.26 Echo String

```
echo <some string>
```

Prints <some string> to the display. Useful in batch files.

5 CORESIGHT COMPONENT COMMAND REFERENCE

5.1 Common Features

CSAT version 2.0.0 introduces a set of commands based on CoreSight components available on the debug bus. These commands simplify accessing and controlling CoreSight systems.

All CoreSight components supported have a set of standard register access sub-commands including those to read and write component registers, or set the component base address. Multiple instances of a component type are supported.

Component commands which have support for only the standard register sub-commands are referred to as register components. Some CoreSight component types have additional sub-commands, and are referred to as extended components.

5.1.1 Component Command Format

Component commands take the form:-

```
<name>.<n> <sub_command> [<options>*]
```

<name> - The name of the component.

<n> - the component instance in the range 0 – 9. Components instances allow the user to support multiple components of the same type on more complex CoreSight systems. Using the command name without an instance suffix references instance .0 initially. This can be changed using the 'def' sub command.

<sub_command> - either one of the standard register sub-commands, or an extended sub-command in component commands that support this feature.

<options> - 0 or more sub command options.

5.1.2 Supported CoreSight Components

Currently the following components support the standard register sub-commands:-

```
cti, etb, etm, itm, pmua9, ptm, tfun, tpiu, v7dbg.
```

The list of supported components can be found using the `cscomp` command.

The following components support additional extended sub-commands:-

```
etb, v7dbg.
```

The component data file `cscompdata.dat` defines all the supported components which have the standard set of register sub-commands. This file defines the register names for the component, the offset from the component base address, and the access permissions for the register. Users can add their own CoreSight component definitions to this file if desired. This file is read into CSAT when the program starts.

Extended commands for components are not defined in this file, but are built into the CSAT product.

5.2 Standard Register Sub-Commands

5.2.1 Set Base Address (baseaddr)

```
<component> baseaddr <address_value>
```

This command must be used first to 'activate' the component instance. No other sub-commands can be used until the component instance has had its base address on the debug bus defined. Base address value is supplied as a 32 bit hex number.

e.g. %> `ptm.3 baseaddr 0x80104000`

5.2.2 List Defined Registers (rlist)

`<component> rlist`

This command lists the registers and register groups defined for the component in the component data file. Registers are grouped in the component data file broadly by function.

e.g %> `tfun rlist`

List of valid register groups and names:-

group name : `csmgmt`

`imcr,ctagsr,ctagcr,lar,lsr,authsr,`

group name : `devid`

`devid,devtypeid,peripid4,peripid5,peripid6,peripid7,peripid0,peripid1,
peripid2,peripid3,compid0,compid1,compid2,compid3,`

group name : `integ`

`itatbdata0,itatbctr2,itatbctr1,itatbctr0,`

group name : `prog`

`ctrl,pcr,`

5.2.3 Set Instance as Default (def)

`<component> def`

This sets the defined instance as the one used for the component command when an instance suffix is not used.

e.g %> `etm.5 def`

Instance `etm.5` is used whenever the `etm` command is used. At startup, the default for each component is set to instance `.0`.

5.2.4 Register Read (rr)

`<component> rr <offset>|<reg_name>|<group_name>| all`

Component registers can be read by offset from the base address of the component, by register name, as a group of registers, or all registers in a component may be read. Registers are grouped in the component data file broadly by function.

`<offset>` - referencing the register by offset means that the read is always attempted, even if the register is write only. The offset reference can also be used to access locations that are not defined as register names. Offset in range `0x000 – 0xFFC`.

`<reg_name>` - referencing the register by name reads the register at the offset defined for the name in the component data file. If the access for the register name is defined as write only then the read is not attempted.

`<group_name>` - all the registers in the group are read and displayed. If any register in the group is defined as write-only access then the read for that register is not attempted, and value marked as `!!!!!!!`. The operation then continues with the next register in the group.

`<all>` - displays all defined registers with same semantics as using a group name.

```
e.g %> etm.1 rr 0x120
%> ptm rr cr
%> v7dbg rr devid
```

5.2.5 Register Write (rw)

```
<component> rw <offset>|<reg_name> <value>
```

Writes value to the register referenced by <offset> or <reg_name>.

<offset> - referencing the register by offset means that the write is always attempted, even if the register is read only. The offset reference can also be used to access locations that are not defined as register names. Offset in range 0x000 – 0xFFC.

<reg_name> - referencing the register by name writes the register at the offset defined for the name in the component data file. If the access for the register name is defined as read only then the write is not attempted.

```
e.g %> etm.1 rw 0x020 0x012403404
%> ptm rw cr 0x0D400
```

5.2.6 Register Read-Modify-Write (rmw)

```
<component> rmw <offset>|<reg_name> <reg_write_value> <mask>
```

This command allows a register to be modified on a bitfield basis. The register is read, the mask and value used to determine which bits are altered. A '1' in the mask value indicates a bit that is allowed to be changed, '0' indicates a bit to remain unaltered. The following logic is used to write the register

```
reg_final_value := (reg_read_value & ~mask) | (reg_write_value & mask)
```

<offset> - referencing the register by offset means that the write is always attempted, even if the register is read only. The offset reference can also be used to access locations that are not defined as register names. Offset in range 0x000 – 0xFFC.

<reg_name> - referencing the register by name writes the register at the offset defined for the name in the component data file. If the access for the register name is defined as read only then the write is not attempted.

```
e.g %> etm.1 rmw 0x020 0x002 0x0F
%> ptm rmw cr 0x400 0x400
```

5.2.7 Set AP Index for Component (apidx)

```
<component> apidx <val>
```

<val> in range 0 – 255.

Normally it is assumed that all CoreSight components are on a single debug bus, attached to an AP in the DAP, the index of which is defined in the `cscomp` command using the `def_apidx` command. If a design has components connected to a different AP, then this command can be used to override the global value.

5.2.8 Command Help

```
<component> help | ?
```

The component commands can all print a summary of the available sub-commands and options. For component commands with extended sub-commands these are also listed.

```
%>etb help
etb <subcommand> [<options>]*
subcommands:-
    rr <offset>|<reg_name>|<group_name>|all - read register or group of
registers
    rw <offset>|<reg_name> <value> - write register by offset or name
    rmw <offset>|<reg_name> <value> <mask> - read-modify-write register by
offset or name
    baseaddr <addr_val> - set base address for this instance
    apidx <val> - override default AP idx for this instance
    rlist - list valid group and register names
    def - make this instance the default for unqualified name
    dumpbin [forceall] [<filename>] - dump trace buffer to data file -
optionally supply filename, force all buffer download.
    read [<start_rec>] [<num_recs>] - display trace data from the etb buffer
    status - current state of etb trace buffer.
    start [<mode>] - start the etb tracing, optionally define mode.
    stop - stop the etb tracing.
```

5.3 Extended Commands : ETB component

5.3.1 ETB Current Status (status)

etb status

Displays information on the state of the trace buffer, and if the ETB is collecting trace or stopped. Displays details of status and FFSR regs, along with the number of records captured if stopped.

```
e.g. %>etb status
etb.0 Status : Trace Disabled
    Buffer Size = 2048 records, 8192 Bytes
    STS Reg 0x0000000C:- FtEmpty:AcqComp:Not Triggered:Not Full
    FFSR Reg 0x2:- FtStopped:Not FlInProgress

    1172 Trace Records Captured
%>
```

5.3.2 ETB Start Trace Collection (start)

etb start [<mode>]

This starts the ETB capturing trace. The default is to use normal mode if the <mode> option is omitted. At start of trace, the RAM write pointer (RWP) is set to 0 – the start of the ETB RAM buffer, and the mode bits in the formatter and flush control register (FFCR) are set to 'normal', or the supplied user mode. Other bits in this register are unaffected.

<mode> can be omitted or one of:-

normal	– normal mode for formatter.
cont	– continuous mode for formatter.
bypass	– formatter off.
asis	– leave mode as-is, do not alter the ffcf.

Note: if the mode bits in the FFCR are altered by this command, they are not restored on stopping trace collection.

5.3.3 ETB Stop Trace Collection (stop)

etb stop

Halts trace collection by the ETB.

5.3.4 ETB Read Trace Buffer (read)

etb read <start_rec> <num_recs>

Read and display on screen <num_recs> records from the trace buffer starting at <start_rec> offset from the oldest/first record in the buffer.

```
%>etb read 0 64
Rec(00000000) : 0x80000813 0xEC600000 0x7C8892D8 0x38223344
Rec(00000004) : 0x05F8A810 0x04220BCE 0x08C808E0 0x09C008E0
Rec(00000008) : 0xD008C808 0xC808C008 0xC008D008 0x0008C808
Rec(00000012) : 0x08C008D0 0x08D008C8 0x08C808C0 0x00C008D0
Rec(00000016) : 0xD008C808 0xC808C008 0xC008D008 0x0008C808
Rec(00000020) : 0x08C008D0 0x08D008C8 0x08C808C0 0x00C008D0
Rec(00000024) : 0xD008C808 0xC808C008 0xC008D008 0x0008C808
Rec(00000028) : 0x08C008D0 0x08D008C8 0x08C808C0 0x00C008D0
Rec(00000032) : 0x08C80813 0x08C008D0 0x08D008C8 0x01C808C0
Rec(00000036) : 0xC008D008 0xD008C808 0xC808C008 0x0008D008
Rec(00000040) : 0x08C808C0 0x08C008D0 0x08D008C8 0x00C808C0
Rec(00000044) : 0xC008D008 0xD008C808 0xC808C008 0x0008D008
Rec(00000048) : 0x08C808C0 0x08C008D0 0x08D008C8 0x00C808C0
Rec(00000052) : 0xC008D008 0xD008C808 0xC808C008 0x0008D008
Rec(00000056) : 0x08C808C0 0x08C008D0 0x08D008C8 0x00C808C0
Rec(00000060) : 0xC008D008 0xD008C808 0xC808C008 0x0008D008
%>
```

5.3.5 ETB Dump Trace Buffer to file (dumpbin)

etb dumpbin [fsync] [forceall] [<file_name>]

This command dumps all the trace samples from the ETB buffer into a disk file, for offline analysis by a suitable tool.

fsync – pre-pend a tpiu frame sync packet to the data – this allows the trace analysis tool to auto-sync with the data in the buffer.

forceall – download the entire buffer, irrespective of how many records have been captured.

<file_name> - name of file for download. Default is **etb_buffer.trb**.

```
%>etb dumpbin fsync mya9_dump.trb
```

```
Uploading 1172 records from ETB memory buffer
Writing 1172 trace records to file mya9_dump.trb
```

5.4 Extended Commands : v7dbg component

The v7dbg component inherits the standard register sub commands and defines has 2 additional sub-commands that control the settings and operation of the component, and dap memory commands `dmr` and `dmw`. The sub-commands are `active` and `fastwordaccess`.

Typing the command v7dbg command without any sub commands causes the current settings to be shown.

```
%>v7dbg
v7dbg.0 Settings:-
  baseaddr 0x80110000
  apidx     0x01
  apctrl    0x80
  fastwordaccess true
  ACTIVE
```

By default the v7dbg command without a suffix refers to the v7dbg.0 instance. The apctrl parameter is set by the cscomp command `def_apctrl`. (See section 5.5).

5.4.1 v7dbg – active

The ‘active’ setting determines which instance of v7dbg is used when memory accesses with a v7dbg option are executed. For example in the 3 core system described below:-

```
%>a9_0
v7dbg.0 Settings:-
  baseaddr 0x80110000
  apidx     0x01
  apctrl    0x80
  fastwordaccess true
  ACTIVE

%>a9_1
v7dbg.1 Settings:-
  baseaddr 0x80112000
  apidx     0x01
  apctrl    0x80
  fastwordaccess true
  Inactive (active instance : v7dbg.0)

%>a9_2
v7dbg.2 Settings:-
  baseaddr 0x80114000
  apidx     0x01
  apctrl    0x80
  fastwordaccess true
  Inactive (active instance : v7dbg.0)
```

a9_0, which is the aliased v7dbg.0 instance is marked as the active instance. Thus if a memory read command such as ‘`dmr v7dbg 0x1000 128`’ is issued, then the v7dbg.0 instance is used.

Note: the `dmr` command **cannot** use the `v7dbg.<n>` syntax to select an instance.

5.4.2 v7dbg fastwordaccess

‘`fastwordaccess`’ can be true or false. By default this is set to true for using the maximum efficiency during memory operations. Only set this to false if a fault is suspected in the core debug hardware. `fastwordaccess` causes the DSCR DCC transfer mode bits to be set to ‘fast’ for the duration of the memory operation.

e.g. `%> v7dbg.5 active`

```
%> v7dbg.1 fastwordaccess false
```

5.5 cscomp Command.

cscomp

cscomp def_apidx <val>

cscomp def_apctrl <val>

The **cscomp** command controls the default settings for all the supported CoreSight component commands. Running the command without any options displays the current settings.

e.g. %>cscomp

```
cscomp : Settings:-
```

```
    def_apidx    0x01
```

```
    def_apctrl   0x82
```

```
cscomp : Available components:-
```

```
    cti
```

```
    etb
```

```
    etm
```

```
    itm
```

```
    pmua9
```

```
    ptm
```

```
    tfun
```

```
    tpiu
```

```
    v7dbg
```

The **def_apidx** option allows the setting of the default apidx for all the components, unless this has been overridden by the individual component setting. <val> should be in the range 0 – 255, but is typically 0 for a system with a DAPLite, or 1 for a system with a DAP.

The **def_apctrl** option sets the value used in bits 31:24 of the control-status register in the AP used to address the debug bus. For an APB-AP this should normally be set to 0x80.

5.6 Alias Command

```
alias <command_name> <alias_name>
```

The alias command creates a new command in the command tree with the name <alias_name>. When this command is executed, the <alias_name> is substituted in the command line with <command_name>.

The alias command is not specifically a CoreSight component command, but is useful in the CoreSight context in describing multi-core systems, making scripts easier to read for the user. For example, if two v7dbg cores are in the system one A9 and one A5, the following commands could be used:-

```
v7dbg.0 baseaddr 0x80001000
v7dbg.1 baseaddr 0x80007000
alias v7dbg.0 a9
alias v7dbg.1 a5
```

Now,

```
A5 rr dscr
```

is interpreted as

```
v7dbg.1 rr dscr
```

6 TRACE COMMAND REFERENCE

6.1 Trace Support in CSAT

CSAT has a set of trace commands which allow the control of trace and extraction of trace data from the RealView Trace 2 (RVT2) trace capture unit and the DSTREAM debug and trace unit.

VSTREAM does not currently have a virtual trace capture capability.

'trace help' lists the available trace functions:

```
%>trace help
Help for RVT CLI
=====

Command Set version : 2.0.1.0
List of Commands:-

async    Switch async monitoring on or off.
close    Close connection to RVT unit
commitconfig  Commit current configuration.
describemode  Describe trace mode.
dumpbin   Dump trace buffer to a binary file.
dumpbuffer  Dump trace buffer to a file.
getconfigitem  Get the value of a config item.
getconfigitems  Get supported configuration items.
getmode    Get current trace operational mode.
getmodes   Get available trace operational modes.
getstate   Get Current RVT state.
help       Print a list of available commands.
identify   Identify versions on attached RVT unit.
open       Open connection to RVT unit
read       Read data
readusb    Read data via usb
reset      Reset RVT.
setconfigitem  Set value of a configuration item.
setmode    Set current trace operational mode.
signals    Get trace signal settings from RVC file and store in configuration.
start      Start tracing.
stop       Stop tracing.
For individual command help type <cmd> help

%>
```

The 'help' command may be used at any time.

6.2 Trace Open

```
trace open [TCP:<hostname> | TCP:<ip address> | USB]
```

Open a connection to the RVT2 trace capture unit or the trace capture component within DSTREAM, using the same connection syntax as connecting to the debug control unit. The connection option can be omitted, in which case the same option as used to connect to the debug control unit is used. The open function must be used before any other of the trace functions except 'trace async'.

e.g. %> `trc open`

Connect to the RVT2 unit attached to the currently connected RVI unit, or the trace components for the currently connected DSTREAM unit.

e.g. %> `trace open TCP:192.168.23.123`

Connect to RVT unit by TCP address.

Opening trace normally results in asynchronous notification messages being sent and displayed.

```
%>
```

```
Async Notification of RVT internal state change...
```

```
  TriggerState = Not Triggered
```

```
  BufferState = Trace Buffer Empty, Stopped
```

```
%>
```

```
Async Notification of RVT internal configuration change...
```

To suppress these messages both on open and later in the trace session, use the 'trace async' function.

6.3 Trace Close

```
trace close
```

This function closes the connection to the trace capture unit.

6.4 Trace Identify

```
trace identify
```

This obtains hardware and software version information from the attached trace capture unit.

e.g.

```
%>trace identify
```

```
RVT Product Name: RealView Trace
```

```
RVT Serial Number: 1064
```

```
Firmware revision: 0x03010000
```

```
Hardware revision: 0x02000201
```

```
PRB Serial Number: 0
```

```
PRB HW Rev: 0x00000000
```

```
FPGA Image Name: rvt2x.bin
```

```
FPGA Image Rev: 0x000004AD
```

```
%>
```

The MS byte of 'hardware revision' indicates the version of the trace capture unit – in the above example RVT2 (0x02.....).

6.5 Trace Getstate

```
trace getstate
```

This function gets the current trace state from the trace capture unit. This indicates if the unit is tracing or stopped, and the amount of records captured if stopped, along with the trigger position if triggered. The format differs between RVT2 and DSTREAM.

e.g. for RVT2

```
%>trace getstate
```

```
GetState Command Response
```

```
Trigger State : Triggered
Buffer State : Trace Buffer Empty, Stopped
Current Trace Write Pointer : 74565
Trigger Pointer : 0
```

e.g. for DSTREAM

```
Hardware is DSTREAM
```

```
Current DSTREAM state...
```

```
lastFrameWritten = not set
triggerFrame = not set
status = 0x00006800
```

BUFFER_FULL(0)	CLOCK_AT_EDGE(0)	USB_PATH_ERR(0)	INVAL_PTR_UPDATE(0)
VALID_TPKT(0)	ALIGNED(0)	FIFO_OVERFLOW(0)	T_CLOCK_ERR(1)
READY(1)	FIFO_UNDERFLOW(0)	FIFO_EMPTY(1)	RESYNC_ERR(0)
TRIGGER_ERR(0)	DATA_IN_BUFFER(0)	TARGET_ERR(0)	USB_ACTIVITY(0)
BUFFER_RD_ERR(0)	DONE_DELAY(0)	ERR(0)	BUFFER_WRAPPED(0)
TRACE_STOPPED(0)	TRIGGER_FOUND(0)		

6.6 Trace Reset

```
trace reset
```

This function resets the trace capture unit.

6.7 Trace Start

```
trace start
```

This function starts the trace unit capturing trace.

6.8 Trace Stop

```
trace stop
```

This function stops the trace unit capturing trace.

6.9 Trace Read

```
trace read <start index> <record count> [{extract [withts]} | raw]
```

This function allows the display of a block of trace data from the trace capture unit. The command line above is the complete set of options, but differing options and combinations are valid for differing versions of trace capture unit.

Read commands are useful for checking for the presence of trace data, and examining small amounts, but the 'dumpbuffer' and 'dumpbin' trace commands are more useful for collecting all the data in a buffer for analysis by other tools.

6.9.1 RVT2 Read Commands

```
trace read <start index> <record count> raw
```

Only the read raw command is valid for RVT2 hardware.

With RVT2, when timestamps are not needed, some of the timestamp area is used to store trace data.

e.g.

Data generated on short run without timestamps

```
%>trace read 0 6 raw
TD00000000: 00080a4c 41000808 080a000a 00000000
TD00000001: 00080a00 0a00080a 080a0008 00000000
TD00000002: 00080a00 0a00080a 080a0008 00000000
TD00000003: 00080a00 0a00080a 080a0008 00000000
TD00000004: 00080a00 0a00080a 080a0008 00000000
TD00000005: 00080a00 0a00080a 080a0008 00000000
```

Data generated on same run with timestamps

```
%>trace read 0 14 raw
TD00000000: 00080a4c 00000808 007ae92e 00000000
TD00000001: 00080100 00000808 007ae960 00000000
TD00000002: 00080800 00000808 007ae992 00000000
TD00000003: 00080800 00000808 007ae9c4 00000000
TD00000004: 00080800 00000808 007ae9f6 00000000
TD00000005: 00080800 00000808 007aea28 00000000
TD00000006: 00080800 00000808 007aea5a 00000000
TD00000007: 00080800 00000808 007aea8c 00000000
TD00000008: 00080800 00000808 007aeabe 00000000
TD00000009: 00080800 00000808 007aeaf0 00000000
TD0000000a: 00080800 00000808 007aeb22 00000000
TD0000000b: 00080800 00000808 007aeb54 00000000
TD0000000c: 00080800 00000808 007aeb86 00000000
TD0000000d: 07080800 00000808 00dca8a1 00010000
```

6.9.2 DSTREAM Read Commands

DSTREAM can also read using the USB port to speed up data transfer.

```
trace readusb <start index> <record count> raw
```

6.10 Trace Dumpbuffer

```
trace dumpbuffer <filename>
```

This function dumps the entire contents of the trace buffer on the trace capture unit into the named file <filename> supplied. The default filename name 'TRACEDMP.TXT' is used if <filename> is omitted. This default can be altered using the `options tracefile` command.

6.11 Trace Dumpbin

```
trace dumpbin <filename>
```

Dumps the entire contents of the trace buffer into a binary format file.

The default filename name 'TRCDMPBIN.TRB' is used if <filename> is omitted. This default can be altered using the `options tracebinfile` command.

These files may be input by other tools to analyse the trace data collected.

6.12 Trace Unit Data Capture and Storage Modes

```
trace describemode <mode name> : get more info about mode.
```

```
trace getmode : get current mode for unit
```

```
trace getmodes : get list of available modes for the unit
```

```
trace setmode <mode name> : set mode on unit
```

Data capture modes may be described as 'classic' or 'continuous'. Classic mode uses the trace TRACECTL pin, along with other pins to determine if a valid trace sample is being presented to the trace probe inputs. This mode may be used with single trace source targets, or CoreSight trace systems that use a TPIU either in Normal or Bypass modes. Continuous mode is designed exclusively to capture trace from a TPIU in Continuous mode.

Data storage modes describe what happens to the data once it has been captured. 'Store and forward' storage modes place trace data into RAM on the capture device until the trace operation is complete. A client can then access the data from the storage RAM.

CSAT can extract trace data from the store and forward modes using either the 'trace read' or 'trace dumpbuffer' and 'trace dumpbin' commands.

The table below shows the names of the modes and the RVT/DSTREAM versions that support them.

Name	Description	RVT2	DSTREAM
classic	Classic store and forward mode	Y	Y
continuous	TPIU continuous mode, store and forward storage.	Y	Y

6.13 Trace Configuration Item Commands

```
trace getconfigitems
```

```
trace getconfigitem [-v] <config item name>
```

```
trace setconfigitem <config item name> <config item value>
trace commitconfig
```

The trace configuration items set up the trace capture unit to capture trace. There are a number of options, which depend on the capture and storage modes, and also the capabilities of the trace capture unit.

The 'getconfigitems' command lists valid configuration items for the current mode and capture device.

The 'getconfigitem' command gets the value, and if the '-v' option is used, additional information about a config item.

e.g.

```
%>trace getconfigitem -v ETM_PROTOCOL
```

```
Name      : ETM_PROTOCOL
```

```
Type      : TPA_TYPE_ENUM
```

```
Size      : 4
```

```
Max       : 0x00000003
```

```
Min       : 0x00000000
```

```
EnumStr:-
```

```
ETMv1
```

```
ETMv3
```

```
ETMv1tov3
```

```
DispStr: ETM Protocol
```

```
HelpStr: The ETM Protocol version
```

```
Avail    : true
```

```
RdOnly   : false
```

```
Value: 0x00000001 {ETMv3}
```

This config item is an enum type, so has string and integer values that are equivalent.

It may be set using either value, using the 'setconfigitem' command:

```
%>trace setconfigitem ETM_PROTOCOL 2
```

```
%>trace commitconfig
```

```
%>trace getconfigitem ETM_PROTOCOL
```

```
ETM_PROTOCOL=0x00000002 {ETMv1tov3}
```

```
%>trace setconfigitem ETM_PROTOCOL ETMv3
```

```
%>trace commitconfig
```

```
%>
```

Note: a 'commitconfig' command is required to force the trace capture unit to validate the configuration. This is to allow a number of values to be set, before doing a 'commitconfig' to validate the set. When setting certain config items, it is possible to temporarily have an illegal combination due to the one at a time nature of setting the items, therefore the combination is only validated once 'commitconfig' is sent, which may be after a number of individual items.

If an illegal combination is detected by the 'commitconfig' command, then an error is returned and none of the config items are set.

Note: before a config item is committed, attempting to read the item results in the original value being read back, not the value about to be committed.

The following sub-sections describe the configuration items, which modes and trace capture unit versions they appear in, and possible values where appropriate.

The modes and capture unit versions are abbreviated as follows:

RVT2X	-	RVT2 classic store and forward mode.
RVT2C	-	RVT2 continuous store and forward mode.
DST	-	DSTREAM – all modes

6.13.1 PORT_WIDTH

Item Name : PORT_WIDTH

Valid In : All versions and modes.

Description: The bit width of the connected ETM or TPIU port. Classic modes have discrete sets of values, continuous is variable from 1 to max port width for capture device. Port widths greater than 16 bit, require RVT2 using a 32 bit trace probe.

Type : Integer

Values : RVT2X, RVT2XS – 4,8,16,32 bits; RVT2C, RVT2CS – 1 to 32 bits; DST 1-16 bits

6.13.2 PORT_MODE

Item Name : PORT_MODE

Valid In : Any classic mode.

Description : The current setting on the ETM if the data is normal or multiplexed across the output pins.

Type : Enum

Values : Mode_Normal, Mode_Mux;

6.13.3 CLOCK_MODE

Item Name : CLOCK_MODE

Valid In : Any classic mode.

Description : The clock mode used for data capture – either single edged – SDR, or double edged – DDR.

Type : Enum

Values : Clock_SDR, Clock_DDR

6.13.4 ETM_PROTOCOL

Item Name : ETM_PROTOCOL

Valid In : Any classic mode

Description : information for the capture device on the ETM protocol being captured. Allows rejection of disabled trace cycles

Type : Enum

Values : RVT2X, RVT2C - ETMv1, ETMv3, ETMv1Tov3; DST - ETMv1, ETMv3

6.13.5 PACKING

Item Name : PACKING

Valid In : All modes

Description : How the data is packed into the trace data ram. RVT2 uses a maximum or minimal packing scheme. DSTREAM does not vary packing

Type : boolean

Values : RVT2 - true – maximum packing, false – minimum packing.

6.13.6 SET_TIMESTAMPS

Item Name : SET_TIMESTAMPS

Valid In : All modes in RVT2

Description : Determines if timestamps are generated and recorded by the RVT unit.

Type : boolean

Values : true – timestamps in use; false – no timestamps being used.

6.13.7 TRIGGER_POSITION

Item Name : TRIGGER_POSITION

Valid In : All modes – RVT2 only.

Description : determines the trigger position in the capture buffer, in terms of a percentage of the buffer.

Type : integer

Values : 1 – 100

6.13.8 TRACE_DEPTH

Item Name : TRACE_DEPTH

Valid In : Any mode. Read only in DSTREAM

Description : How large a buffer to used when tracing data. May be used to artificially reduce the captured data size in RVT2.

Type : integer.

Values : depends on size of buffer.

6.13.9 SYNC_STRIP

Item Name : SYNC_STRIP

Valid In : Any continuous mode

Description : Determines if the contiguous TPIU frame syncs, or half frame syncs, are stripped from the data before it is stored.

Type : Enum – RVT2C, RVT2CS;

Values : RVT2 – KeepAll, StripHalf; StripFull.

Unused in DSTREAM.

6.13.10 CLOCK_EDGE

Item Name : CLOCK_EDGE

Valid In : Any classic mode. (visible but unused in RVT2 continuous modes)

Description : Determines the clock edge to capture data if using SDR clocking.

Type : enum

Values : SDR_Rising – capture on rising edge; SDR_Falling – capture on falling edge.

6.13.11 POST_TRIGGER_FILL

Item Name : POST_TRIGGER_FILL

Valid In : RVT2 none streaming modes.

Description : continue to fill buffer to the end after a trigger operation, even if this exceeds buffer depth.

Type : boolean

Values : true / false

6.13.12 The DELAY_TRACE_xxx set

The config items DELAY_TRACE_CLOCK, and DELAY_TRACE_SIGNAL_1 to DELAY_TRACE_SIGNAL_36, allow the user to tune the line delays in the trace data capture system in RVT2. See ref 1 for further information.

6.13.13 Diagnostic Config Items

The following RVT2 items are for diagnostic and test use only – do not use.

TPIU_DEBUG.

FPGA_DIAGNOSTICS.

The following DSTREAM item is also for diagnostic use only

USB_ENUM_ID.

6.13.14 DSTREAM Trace Capture Parameters

The amount of trace captured, and trigger position is controlled by a group of config items in DSTREAM.

6.13.14.1 BUFFER_FULL_MARKER.

Integer number of 512 byte trace lines to capture before buffer is full. Limits the buffer size

6.13.14.2 TRIGGER_RUN_LENGTH

Integer number of trace lines to capture after a trigger is detected.

6.13.14.3 STOP_AFTER_TRIGGER

Boolean controlling if the trace capture stops after a trigger is detected. See the TRIGGER_RUN_LENGTH element for the number of frames captured after the trigger, and before the capture is halted.

6.13.14.4 WRAP_ON_FULL

Boolean controlling if the trace capture stops when full, or wraps and starts overwriting oldest data until a stop condition is met.

6.14 Trace Async

```
trace async {on | off }
```

This command enables or disables the display of the asynchronous status messages from RVT within CSAT. These occur if the state of trace capture changes, or if configuration changes. The information given in the async state change message is the same as that printed by the 'getstate' function.

7 EXAMPLES

7.1 Example session log

```
%%con TCP:Koyaanisqatsi
Connected to:ARM RealView ICE
Base H/W: V1 Rev C-01
TurboTAP Rev: 1.10
Firmware: 7.2.0, Build 12
%%chain dev=auto clk=A
Jtag clock set to 50000000A
ID:0 ARMCS-DP
ID:1 ETMBUF
ID:2 ARM1136JF-S
%%dmr DPACC.CSW
dpmemread : Insufficient command parameters.
%%drr DPACC.CSW
Command failed - no connection.
%%dvo 0
Open connection to device ID : 0x00000F0F, version 0x00000006
Msg returned with RVMOpenConn: CoreSight DAP template
%%drr DPACC.CSW
DPACC.CSW [2081] => 0xF0000000
%%dmr 0 0x8000 8
0x00008000 : 0x86A942AF 0x6A41717B 0x1BF3500C 0xE1AC9832
0x00008010 : 0xD07FBD57 0xD50D030F 0xFA736D84 0x2409FF10
%%dmw 0 0x8000 1 2 3 4 5 6 7 8
Wrote 8 words to memory via AP0
%%dmr 0 0x8000 8
0x00008000 : 0x00000001 0x00000002 0x00000003 0x00000004
0x00008010 : 0x00000005 0x00000006 0x00000007 0x00000008
%%drr AP0.CSW
AP0.CSW [1800] => 0x43800062
%%drr AP0.0
AP0.0 [1800] => 0x43800062
%%drr APACC.0
APACC.0 [2000] => 0x43800062
%%exit
```

7.2 Example Batch Command File

```
# a startup script

# A single line comment
echo Running my startup script

con TCP:Koyaanisqatsi # an inline comment
chain dev=auto clk=A
dvo 0
```

7.3 Example Setup Script Using Coresight Component Commands

This setup script is used with a 3 core A9 on a PBXA9 board.

```
# CSAT 2.0 - PB-A9 component setup

# components all on APb-AP - apidx 1
cscomp def_apidx 1

#### 3 a9 cores

# core 0
#
v7dbg.0 baseaddr 0x80110000
pmua9.0 baseaddr 0x80111000
cti.0 baseaddr 0x80118000
ptm.0 baseaddr 0x8011c000

alias v7dbg.0 a9_0
alias pmua9.0 pmu_0
alias cti.0 cti_0
alias ptm.0 ptm_0

# core 1
#
v7dbg.1 baseaddr 0x80112000
pmua9.1 baseaddr 0x80113000
cti.1 baseaddr 0x80119000
ptm.1 baseaddr 0x8011d000

alias v7dbg.1 a9_1
alias pmua9.1 pmu_1
alias cti.1 cti_1
alias ptm.1 ptm_1

# core 2
#
v7dbg.2 baseaddr 0x80114000
```

```
pmua9.2 baseaddr 0x80115000
cti.2   baseaddr 0x8011a000
ptm.2   baseaddr 0x8011e000
```

```
alias v7dbg.2 a9_2
alias pmua9.2 pmu_2
alias cti.2   cti_2
alias ptm.2   ptm_2
```

```
# common cs comps
etb.0 baseaddr 0x80001000
cti.4 baseaddr 0x80002000
tpiu.0 baseaddr 0x80003000
tfun.0 baseaddr 0x80004000
itm.0 baseaddr 0x80005000
```

```
alias cti.4 cscti
alias etb.0 csetb
alias tpiu.0 cstpiu
alias tfun.0 cstfun
alias itm.0 csitm
```

Once the script is run then the aliases can be used to control components – e.g. halt core 0

```
a9_0 rr prsr
a9_0 rr prsr # read prsr to clear down sticky power down bit
a9_0 rr dscr # read dscr
a9_0 rmw dscr 0x00004000 0x00004000 # write dscr halt mode debugging bit
a9_0 rw prcr 0x1 # request core to halt
a9_0 rr dscr # read dscr - check it has halted
```

7.4 Example Script - Setting PTM to Trace All

```
# setup ptm to trace all - default ptm instance

# print ptm settings
ptm

# print current control register
ptm rr cr

# write power up and programming bit on
ptm rw cr 0x400

# check programming bit is on
ptm rr sr
ptm rr cr

# write the trace control regs for trace all.
ptm rw tecr1 0x01000000
ptm rw teevr 0x6f

# write the trace id to 9
ptm rw traceidr 9

# done - now need to clear the program bit and trace will start.
```