RFcreations
...a different perspective

Moreph Channel Sounding User Manual V3.01

Tim Newton

Last Updated: 17th July 2025

1 Contents

1 CONTENTS 2

2 HARDWARE CONNECTIVITY 7

2.1 Power and control 7

2.2 DUT HCI 7

2.3 Single antenna port 8

2.4 Multiple antenna ports 8

2.5 Logic analyser 10

2.6 Multiple DUTs 12

2.7 External reference clock 13

3 LAUNCHING THE CHANNEL SOUNDING APPLICATION 14

3.1 Launching UI 14

3.2 Launching from the API 15

4 ANATOMY OF UI 16

5 MODES OF OPERATION 17

6 TOOLBAR 18

6.1 Open 18

6.2 Save 18

6.3 Export IQ data 18

6.4 Export summary 19

6.5 Export test results 19

6.6 PCT 20

6.7 Flash 20

6.8 Capture plot 20

6.9 Screen capture 21

6.10 Zoom in 21
6.11 Zoom out 21
6.12 Pan 21
6.13 Pan horizontally 21
6.14 Pan vertically 22
6.15 Connect DUT 22
6.16 Query DUT 22
6.17 Reset 23
6.18 Run/Stop 23
6.19 Discard 23
6.20 About 24
6.21 Help 24
6.22 DUT selection 24

7 STATUS BAR 25

7.1 Overview 25
7.2 Friendly name 25
7.3 Unit serial number 25
7.4 Overload indicator 25
7.5 Number of DUTs 25
7.6 Switch box detection 25
7.7 Error message 26
7.8 Data and time 26

8 CONNECTING TO A DUT 27

8.1 DUT control menu 27
8.2 Mandatory items 28
8.3 Optional items 28
8.4 Establishing a connection 30

8.5 Setting supported features **30**
8.6 Sending additional HCI commands **34**
8.7 Disconnecting **35**
9 MANUAL MODE **36**
9.1 Overview **36**
9.2 Mode **37**
9.3 Distortions **40**
9.4 Timings **43**
9.5 Addresses **44**
9.6 Channels **45**
9.6.1 Custom channel map 45
9.6.2 Frequency hopping algorithms #3b and #3c 46
9.7 Power **49**
9.8 Antennas **50**
9.9 Switch/GPIO **52**
9.10 Cable **55**
9.11 FAE **56**
9.12 Running a channel sounding procedure **58**
9.13 Analysing a channel sounding procedure **59**
9.13.1 Upper graph 59
9.13.2 Lower graph 62
9.13.3 Delta cursors 63
9.13.4 Summary window 63
9.13.5 Linear phase regression API 70
10 SCRIPTED MODE **72**
10.1 Overview **72**
10.2 Creating a script **72**

10.3 Gain control test 77
10.4 Running a script 78
10.5 Examining results 79
10.6 Debugging test scripts 81
10.7 Running test scripts from the API 81

11 AM ATTACK MODE 87
11.1 Overview 87
11.2 Generating an AM attack slice 87
11.3 AM attack API 88
11.4 Custom AM attack pulses 88

12 LOG WINDOW 90
12.1 Overview 90
12.2 Raw octets 90
12.3 HCI decode 91
12.4 Channels, addresses and modes 91
12.5 Informational messages 91
12.6 Error messages 91
12.7 Test script messages 92
12.8 Navigating within the log window 92
12.9 Exporting log window contents 92

13 CALIBRATION 93
13.1 Overview 93
13.2 Calibration procedure 94
13.3 Save and restoring calibration data 94

14 HEALTH 95

15 C API 96
15.1 Overview 96
15.2 API server 96
15.3 Error handling 96
15.4 Connecting to a moreph 97
15.5 Making the UI visible 97
15.6 Connecting to a DUT 98
15.7 Setting the operating mode 99
15.8 Set cable parameters 99
15.9 Setting up a channel sounding procedure 100
15.10 Running a channel sounding procedure 100
15.11 Reading results of a channel sounding procedure 101
15.12 Loading a test script 101
15.13 Creating a test script 101
15.14 Running a test script 103
15.15 Reading the results of a test script 103
15.16 Disconnecting 105
15.17 Resetting moreph hardware 105
16 PYTHON API 106
16.1 Overview 106
16.2 API server 106
16.3 Reseting moreph hardware 106
17 STANDALONE MODE 107
18 SUPPORTED GRAPHICS FORMATS 108

2 Hardware connectivity

2.1 Power and control

The moreph is powered by a 12V mains adapter. This should be connected to the barrel jack on the rear panel.

The moreph can be controlled by a host computer over either USB or Ethernet. These connectors are on the rear panel of the moreph.



Power and control connections on rear of moreph

2.2 DUT HCI

Although manual mode channel sounding procedures can be executed without the moreph having control of the DUT, most operations require an HCI interface to the DUT.

The channel sounding software expects the DUT to expose a comport to which HCI commands can be sent and from which HCI responses and events can be received using the H4 protocol. The channel sounding software provides a number of controls to limit the rate at which HCI commands are sent. All HCI traffic can be recorded in the “Log” window.

2.3 Single antenna port

For DUTs with a single antenna port a connection should be made direct to the Tx/Rx port on the moreph. It is advisable to minimise the loss in this connection.



Single DUT connected to Tx/Rx port of moreph

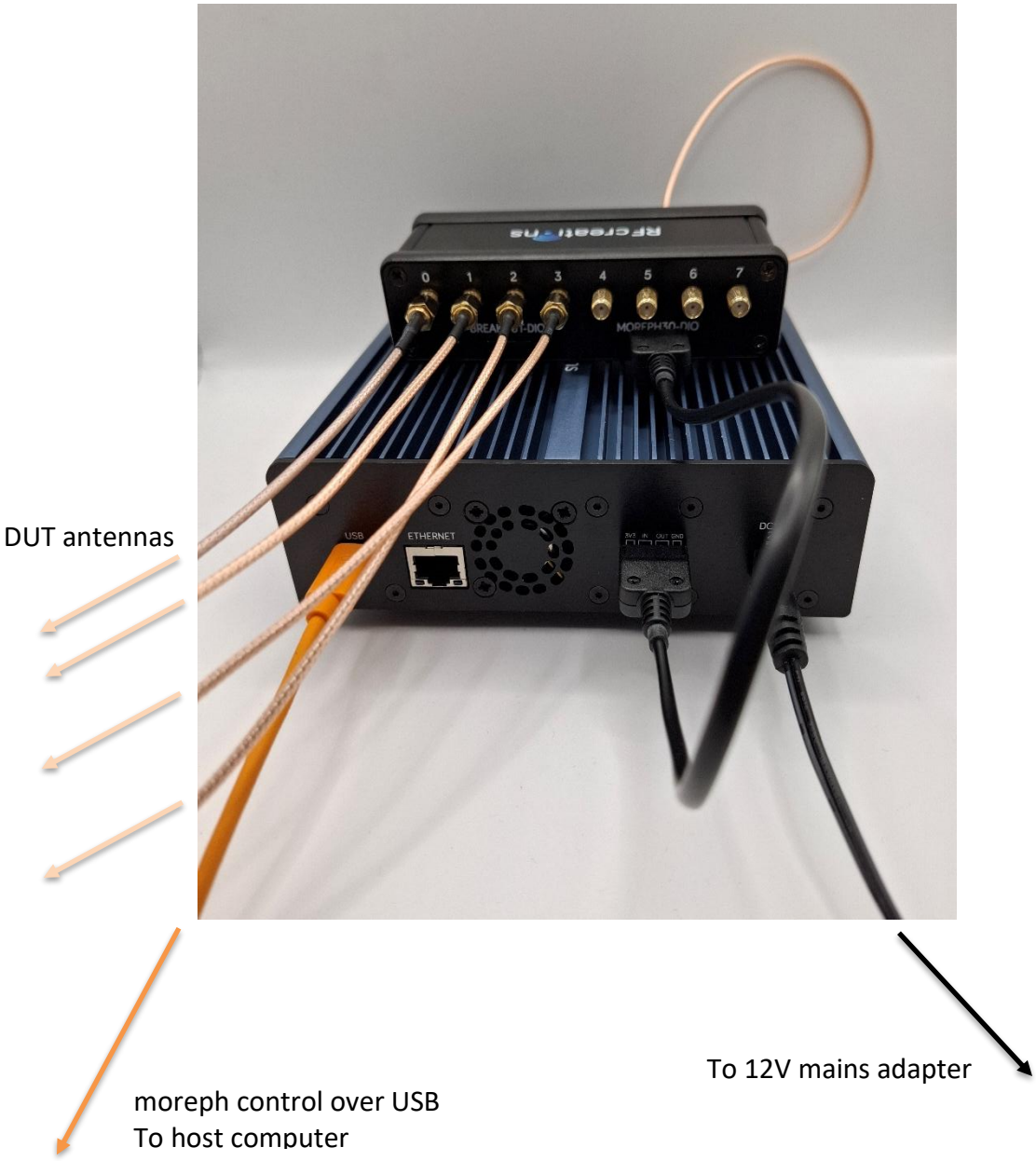
2.4 Multiple antenna ports

For DUTs with multiple antenna ports connections should be made to the 8-way switch box. The lowest numbered antenna port on the DUT should be connected to port “0” on the 8-way switch box, the next to port “1”, etc. The “COMMON” port on the 8-way switch box should be connected to the Tx/Rx port on the moreph. It is advisable to minimise the loss in these connections.

The ”MOREPH-30 DIO” port on the 8-way switch box should be connected to the digital IO port on the rear panel of the moreph.

The 8-way switch box consists of a 1:8 power splitter. On each of the 8 branches are two non-reflective switches. When a branch is disabled, these switches terminate both the internal branch of the power splitter and the associated SMA connector in 50Ω.

The majority of the Bluetooth defined test cases only access one antenna port on the DUT. With the currently defined tests, only the antenna switching integrity tests and the phase measurement accuracy tests access more than one antenna port on the DUT. Hence the antenna switch box is currently only required for these test cases.



8-way RF switch box with cabling to 4 DUT antennas and control to moreph



Common RF connection between 8-way RF switch box and moreph

2.5 Logic analyser

A logic probe can be used to monitor signals generated by the DUT and plot these alongside IQ data when operating in manual mode. The moreph will also output its timing signals via the logic probe. The logic probe has a bandwidth of 22MHz.

The logic probe should be connected to the digital IO port on the rear panel of the moreph.

The IO voltage for the logic signals can either be provided by the moreph or applied externally. See section Switch/GPIO. If the moreph provides the IO voltage, then it is fixed at 3v3. If the IO voltage is applied externally, then it must be in the range 0v8 to 3v3.

The red cable on the logic probe either provides the 3v3 IO voltage from the moreph or accepts the IO voltage applied by the DUT. The black cable on the logic probe should be connected to the DUT ground.

The signals output by the moreph are:

CS_SYNC transmit active	GPIO 11
CS tone transmit active	GPIO 12
Transmitter active	GPIO 13
Ready	GPIO 14



moreph control over USB
To host computer

To 12V mains adapter

Logic analyser connected to rear of moreph via control cable



Logic analyser daisy chained via 8-way RF switch box to moreph

2.6 Multiple DUTs

It is possible to have multiple DUTs connected to the moreph simultaneously. Only one DUT can be tested at any one time. All the DUTs must operate with the same UART configuration and have the same supported channel sounding features.

The multiple DUTs must be connected via the 8 way RF switch box. The first DUT has its antenna ports connected to ports 0 to (N-1) on the 8 way RF switch box, where N is the number of antenna ports supported by the DUT. The second DUT has its antenna ports connected to ports N to 2N-1 on the 8 way RF switch box.

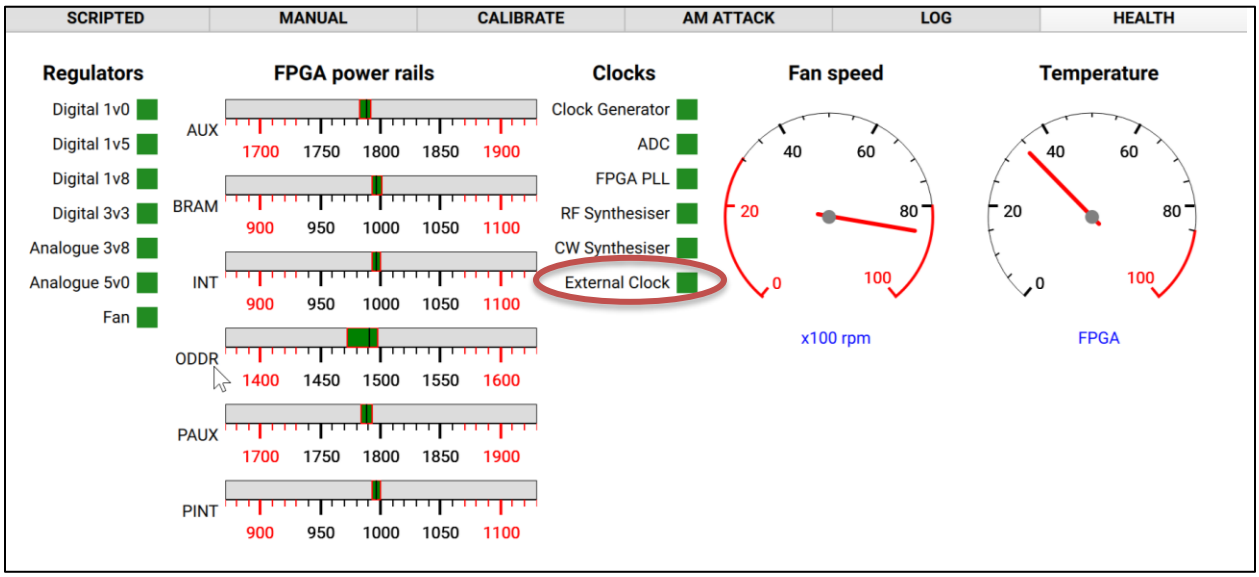
2.7 External reference clock

The moreph can accept an external 10MHz reference clock on the “EXT CLK IN” port on the front panel. If this clock is present prior to launching an application on the moreph, then the moreph will automatically lock its internal clocks to the external reference.

The specification for the reference clock input port is:

Connector type	SMA
Minimum input signal	-10dBm
Maximum input signal	+20dBm
Frequency	10 MHz
Pullability	30ppm
Impedance	50Ω
Coupling	AC
Maximum DC voltage	50V

If the moreph successfully locks to an external reference clock, then this will be indicated under the “Health” tab.



3 Launching the channel sounding application

3.1 Launching UI

In order to communicate with the moreph, it is necessary to attach it to a host computer via USB or Ethernet (or both). An Ethernet connection is only possible if the host computer and moreph reside on the same subnet. The moreph IP address can be changed by connecting it to a host computer via USB and using the Application Loader.

To launch the channel sounding UI it is first necessary to run Application Loader. This should result in the following screen being displayed:

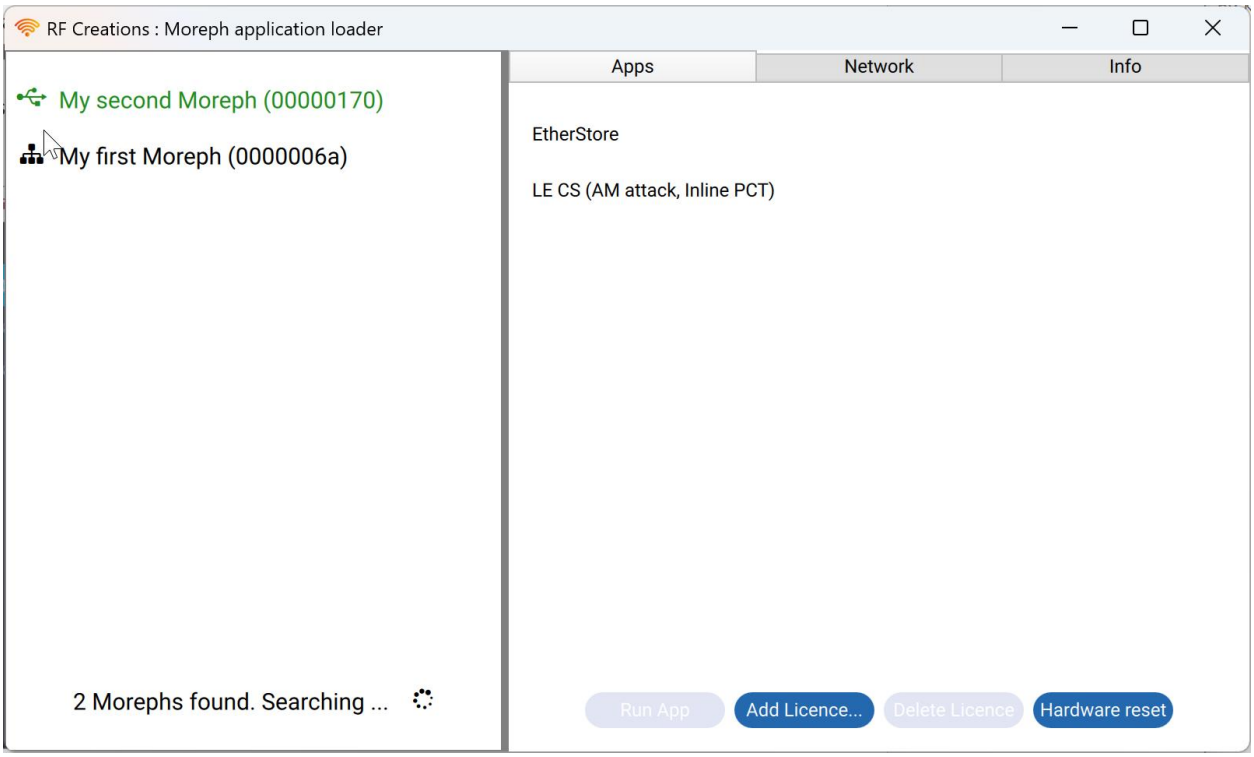


Figure 1: Application Loader main screen.

This screen indicates that the following moreph devices have been discovered:

1. Serial number 00000170 connected via USB (highlighted)
2. Serial number 0000006A connected via Ethernet

The right-hand side of the window has three tabs:

1. *Apps*. Shows which applications are licensed to run on this unit. It also permits the loading of new licence keys.
2. *Network*. Shows the current network settings and permits these to be modified.
3. *Info*. Provides more information about the unit and permits the unit's friendly name to be modified. It also provides a means of updating the firmware on the unit.

To launch the channel sounding UI open the "*Apps"* tab and then either:

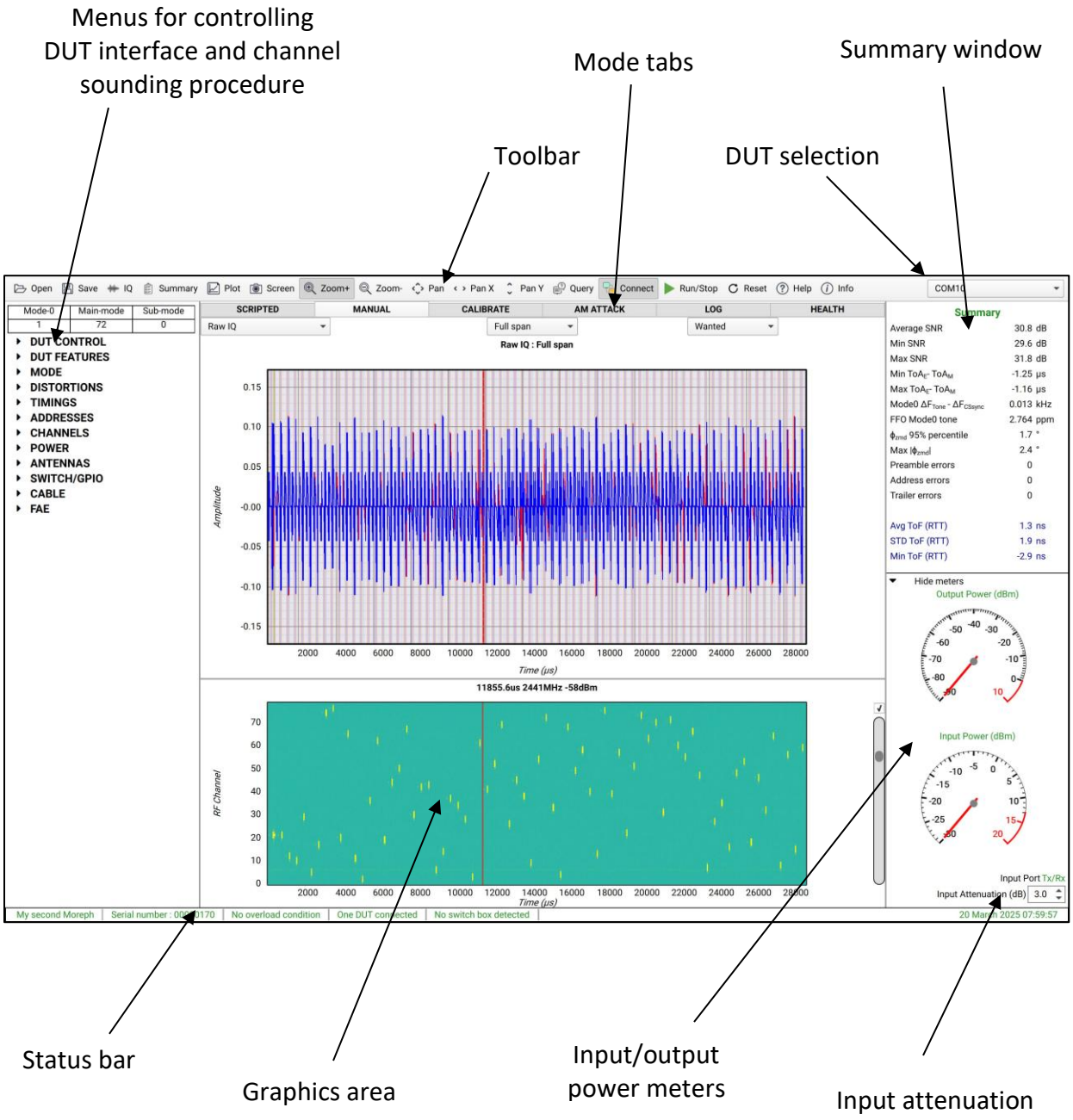
1. Double click on the *LE CS* application
2. Highlight the *LE CS* application and then click the "*Run App*" button.

On launching, the searching cursor should stall, the fan on the moreph unit will start to spin and after a few seconds the channel sounding UI will load.

3.2 Launching from the API

Details of how to launch the channel sounding application from the API are contained in section C API. When running with the API it is possible to make the UI visible.

4 Anatomy of UI



5 Modes of operation

There are four modes of operation:

1. Scripted
   In this mode of operation the user can construct a list of test cases based on the Bluetooth test specifications. The test cases in the list can be executed and the results exported. Test lists may be saved and restored.
   Scripted mode requires an HCI connection to the DUT.
2. Manual
   In this mode of operation the user can construct a specific channel sounding procedure and then execute it. Once executed, the user can examine the collected data and quantities derived from it in detail.
   Manual mode can be run either with or without an HCI connection to the DUT.
3. AM attack
   In this mode an AM attacker is deployed against the DUT. A duty cycle / offset slice showing the derived ToF is generated. The slice can be generated for different attack amplitudes and overall the full range of channel sounding procedure parameters.
   AM attack mode requires an HCI connection to the DUT.
4. Calibrate
   The calibrate mode allows the user to perform both group delay and phase response calibrations of the moreph. Amplitude calibration of the moreph requires the unit to be returned to RFcreations.
   No DUT is required when performing calibration of the unit.

To select the operating mode of the unit from the UI, use the tabs immediately below the toolbar.

To select the operating mode of the unit from the API use:

```
CS_setOperatingMode( opMode_t mode );
```

Options for `mode` are:

`CS_scripted`	Scripted mode
`CS_manual`	Manual mode
`CS_amAttack`	AM attack mode
`CS_calibrate`	Calibrate mode

6 Toolbar

6.1 Open



Mode	Action	API
Scripted	Loads a test script	`CS_loadTestScript( char* filename )`
Manual	Loads a saved capture	`CS_loadCapture( char* filename )`
AM attack	*Not available*	
Calibrate	Loads calibration data	`CS_loadCalibration( char* filename )`

6.2 Save



Mode	Action	API
Scripted	Saves current test script	`CS_saveTestScript( char* filename )`
Manual	Saves current capture	`CS_saveCapture( char* filename )`
AM attack	*Not available*	
Calibrate	Saves current calibration data	`CS_saveCalibration( char* filename )`
Log	Exports log as HTML	`CS_saveLog( char* filename )`

Note: In the UI the save icon has a different meaning when the log window is being viewed. The API can save the contents of the log window irrespective of the current operating mode.

6.3 Export IQ data



Mode	Action	API
Scripted	*Not available*	
Manual	Saves current capture	`CS_saveIQ( char* filename )`
AM attack	*Not available*	
Calibrate	*Not available*	

The IQ data consists of pairs of 16bit signed integers sampled at 20MSps. These are exported as a binary file.

This icon is only visible after a successful channel sounding procedure has been executed.

6.4 Export summary



Mode	Action	API
Scripted	*Not available*	
Manual	Saves results of scripted tests	`CS_saveSummary( char* filename )`
AM attack	*Not available*	
Calibrate	*Not available*	

The contents of the summary window for all steps in the current channel sounding procedure are exported in xml format.

This icon is only visible after a successful channel sounding procedure has been executed.

6.5 Export test results



Mode	Action	API
Scripted	Saves results of scripted tests	`CS_saveResults( char* filename )`
Manual	Saves results of scripted tests	`CS_saveResults( char* filename )`
AM attack	Saves results of scripted tests	`CS_saveResults( char* filename )`
Calibrate	Saves results of scripted tests	`CS_saveResults( char* filename )`

The results of the last executed test script are exported as html. The export includes both the tabular results as well as any graphics which were produced.

This icon is only visible once a test script has been executed. The icon can be accessed irrespective of the current operating mode.

6.6 PCT



Mode	Action	API
Scripted	*Not available*	
Manual	Save PCT results as CSV	
AM attack	*Not available*	
Calibrate	*Not available*	

Exports the PCT measurements made by the moreph and the PCT measurements made by the DUT to a comma separated variable file (csv).

6.7 Flash



Mode	Action	API
Scripted	*Not available*	
Manual	*Not available*	
AM attack	*Not available*	
Calibrate	Saves results of scripted tests	`CS_flashCalibration( void )`

Places the current calibration data into non-volatile storage on the moreph. This will become the default calibration data when the unit is powered up.

This icon is only visible once a calibration procedure has been completed or a calibration file has been loaded.

6.8 Capture plot



Mode	Action	API
Scripted	Saves current displayed graph	`CS_saveGraph( char* filename )`
Manual	Saves upper graph	`CS_saveGraph( char* filename )`
AM attack	Saves displayed slice	`CS_saveGraph( char* filename )`
Calibrate	Saves group delay graph	`CS_saveGraph( char* filename )`

The format in which the graph is saved is determined by the filename extension.

6.9 Screen capture



Mode	Action	API
Scripted	Performs a screenshot	`CS_screenshot( char* filename )`
Manual	Performs a screenshot	`CS_screenshot( char* filename )`
AM attack	Performs a screenshot	`CS_screenshot( char* filename )`
Calibrate	Performs a screenshot	`CS_screenshot( char* filename )`

The format in which the screenshot is saved is determined by the filename extension.

6.10 Zoom in



Activates the cross-hair cursor which permits zooming within in the graphics area. Depress the left mouse button whilst dragging the cursor to select the area to be displayed. Clicking the right mouse button within the graphics area will give a list of additional zoom options.

6.11 Zoom out



Zooms out within the graphics area. Clicking the right mouse button within the graphics area will give a list of additional zoom options.

6.12 Pan



Pans within the graphics area. Hold down the left mouse button and drag to pan anywhere within the graphics display.

6.13 Pan horizontally



Pans along the x-axis within the graphics area. Hold down the left mouse button and drag horizontally. This is particularly useful for examining long waveforms.

6.14 Pan vertically



Pans along the y-axis within the graphics area. Hold down the left mouse button and drag vertically.

6.15 Connect DUT



When the icon is shown in outline no DUT is connected. On clicking the icon, the moreph will try to communicate with the DUT(s) using the H4 protocol and the parameters selected in the “DUT control” menu.

Corresponding API call is `CS_connectDUT( char *comport )`.



When the icon is shown filled one or more DUT(s) are connected via H4. On clicking the icon the DUT(s) will be disconnected.

Corresponding API call is `CS_disconnectDUT( void )`.

6.16 Query DUT



Clicking the query icon will cause the moreph to query the DUT supported features over HCI. If no DUT is currently connected, then an attempt will be made to connect to the DUT(s) selected in the “DUT control” menu before querying the supported features. If more than one DUT is connected, then a single query will be made to the currently selected DUT. Once the query has been completed, the “DUT features” menu will be expanded to show the supported features.

Corresponding API call is `CS_queryDUT( void )`.

6.17 Reset



Resets all channel sounding procedure parameters to their default values. This icon is only visible in manual mode. Corresponding API call is `CS_resetCS( void )`.

6.18 Run/Stop



Mode	Action	API
Scripted	Execute the current test script	`CS_runProcedure( void )`
Manual	Run as CS procedure	`CS_runProcedure( void )`
AM attack	Generate an AM attack slice	`CS_runProcedure( void )`
Calibrate	Start the calibration procedure	`CS_runProcedure( void )`



Mode	Action	API
Scripted	Use cancel from progress window	`CS_abortProcedure( void )`
Manual	Aborts CS procedure	`CS_abortProcedure( void )`
AM attack	Use cancel from progress window	`CS_abortProcedure( void )`
Calibrate	Use cancel from progress window	`CS_abortProcedure( void )`

6.19 Discard



Mode	Action	API
Scripted	Discards test results	`CS_discardResults( void )`
Manual	Discards current capture	`CS_discardCapture( void )`
AM attack	Not available	
Calibrate	Clears calibration data	
Log	Clears log window contents	`CS_clearLog( void )`

Note that the impact of this icon is different if the log window is the currently selected tab.

6.20 About



Displays version information.

6.21 Help



Displays online documentation in the default web browser.

6.22 DUT selection



Shows the currently selected DUT. When more than one DUT is connected, a drop-down menu is provided to select which DUT to use.

There is no corresponding API call since the API is only able to connect to a single DUT at any one time.

7 Status bar

7.1 Overview

The status bar at the bottom of the window is divided into seven areas:

1. Friendly name of the moreph
2. Serial number of the moreph
3. Overload indicator
4. Number of DUTs connected
5. Switch box detection
6. Error message
7. Current data and time

7.2 Friendly name

Displays the moreph friendly name. This may be modified using the “Info” tab on the Application Loader.

7.3 Unit serial number

Displays the moreph serial number as an 8 digit hexadecimal value.

7.4 Overload indicator

The overload indicator will turn red when an overload condition occurs on either the transmitter output or the selected receiver input port. The text of the message will indicate where the overload condition is occurring.

7.5 Number of DUTs

The text indicates the number of DUTs which are currently connected to the moreph.

7.6 Switch box detection

The text indicates whether an 8-way RF switch box connected to the moreph has been automatically detected.

7.7 Error message

The error message reflects the last error detected by the channel sounding application running on the moreph. This message is cleared when either the “Run” or “Discard” buttons are pressed on the toolbar.

7.8 Data and time

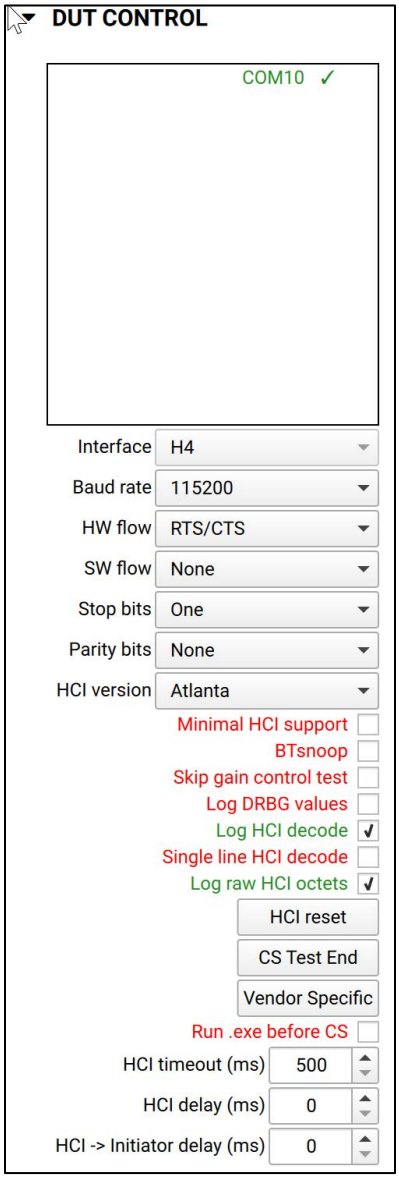
The current data and time are displayed at the bottom right of the status bar.

8 Connecting to a DUT

8.1 DUT control menu

Open the DUT control menu.

Select the comport(s) to which the DUT(s) are attached. More than one DUT can be connected simultaneously. If more the one DUT is connected, they must be connected via the 8-way RF switch box. Although more than one DUT can be connected simultaneously, only one DUT can be tested at a time. The DUT to be tested is selected from the drop-down menu on the toolbar.



8.2 Mandatory items

It is mandatory to set the following items:

Item	API	Options
Baud rate	`CS_setBaudRate( uint32_t )`	
Hardware flow control	`CS_setHWflow( hwFlow_t )`	`CS_HW_none` `CS_HW_RTS_CTS`
Software flow control	`CS_setSWflow( swFlow_t )`	`CS_SW_none` `CS_SW_XON_XOFF`
Number of stop bits	`CS_setStopBits( uint32_t )`	`1` `2`

Number of parity bits	`CS_setParity( parity_t )`	`CS_PARITY_none`
		`CS_PARITY_even`
		`CS_PARITY_odd`

HCI version	`CS_setHCIversion( char * )`	`"Budapest IOP 2023"`
		`"Paris IOP 2023"`
		`"Atlanta R04, 3 octet PCT"`
		`"Atlanta R04, 4 octet PCT"`
		`"Atlanta R06"`
		`"Atlanta"`
		`"Core 6.1"`
		`"Stockholm"`

Currently only H4 protocol is supported. Contact RFcreations if H5 or BCSP is required.

If “Minimal HCI support” is checked, then the moreph will only issue HCI CS test commands and HCI CS test end commands. No event filters will be set.

8.3 Optional items

The following optional items impact the H4 protocol:

Item	Description	API
HCI timeout	The maximum interval between the moreph sending an HCI command and the DUT responding.	`CS_setHCItimeout( uint32_t ms )`
HCI delay	The delay inserted prior to issuing an HCI command.	`CS_setHCIdelay( uint32_t ms )`
HCI -> Initiator delay	The delay between the moreph issuing an HCI test command and the moreph sending the first initiator Mode-0 CS_SYNC.	`CS_setHCIinitiator( uint32_t ms)`

The following optional items alter what is displayed in the log window:

Item	Description	API
DRBG values	When checked, the addresses and channels generated by the DRBG will be written to the log window.	`CS_logDRGB( bool )`
Log HCI decode	When checked, a full HCI decode of all HCI traffic to and from the DUT will be written to the log window.	`CS_logHCIdecode( bool )`
Single line HCI decode	When checked, the HCI decode is written to the log file on a single line. When unchecked, multiple lines are used for the HCI decode.	
Raw HCI octets	When checked, the raw octets sent and received over the H4 link will be written to the log window together with an associated timestamp.	`CS_logRawOctets( bool )`

Checking “BTsnoop” will result in all HCI traffic being written to a BTsnoop file. When this item is checked, a file dialogue will pop-up so the destination for the BTsnoop data can be specified.

The corresponding API call is `CS_setBTsnoop( char *filename )`. If the filename specified in the call is empty, then any open BTsnoop file will be closed.

Prior to each test script run a gain control test is executed. The purpose of the gain control test is to estimate the output power from the DUT. Once this has been established the gain controls within the moreph receiver chain are automatically adjusted to maximise the dynamic range. If “Skip gain control test” is checked, then this procedure will be skipped.

The corresponding API call is `CS_skipGainControl( bool )`.

Prior to each HCI LE CS Test command issued by the moreph it is possible to run a user supplied executable. The moreph will close the corresponding comport prior to executing the user supplied code and then re-open the comport once the code has completed. This provides a facility for the user to reset the DUT prior to a new HCI LE CS Test command being issued. To enable this functionality check “Run .exe before CS”. A file dialogue will pop-up so that the location of the user supplied executable can be entered.

The corresponding API call is `CS_setUserExe( char* filename )`. If the filename specified is an empty string, then the corresponding functionality will be disabled.

8.4 Establishing a connection

To establish a connection press the “Connect” button on the toolbar.

Any errors incurred during the connection process will be appear in the status bar and also appear in the log window.

The number of DUTs successfully connected will appear in the status bar. The drop-down menu at the top right of the toolbar will list the comports which have been connected.

In the “DUT control” menu, devices which are selected and connected will appear in green with a tick. Devices which are selected and have not been connected will be in red with a cross.

The corresponding API call is `CS_connectDUT( char* comport )`. In the API there is a restriction on a single DUT being connected at any one time.

It is possible to determine whether a DUT is connected by calling `CS_isDUTconnected()`. This can be called ay anytime.

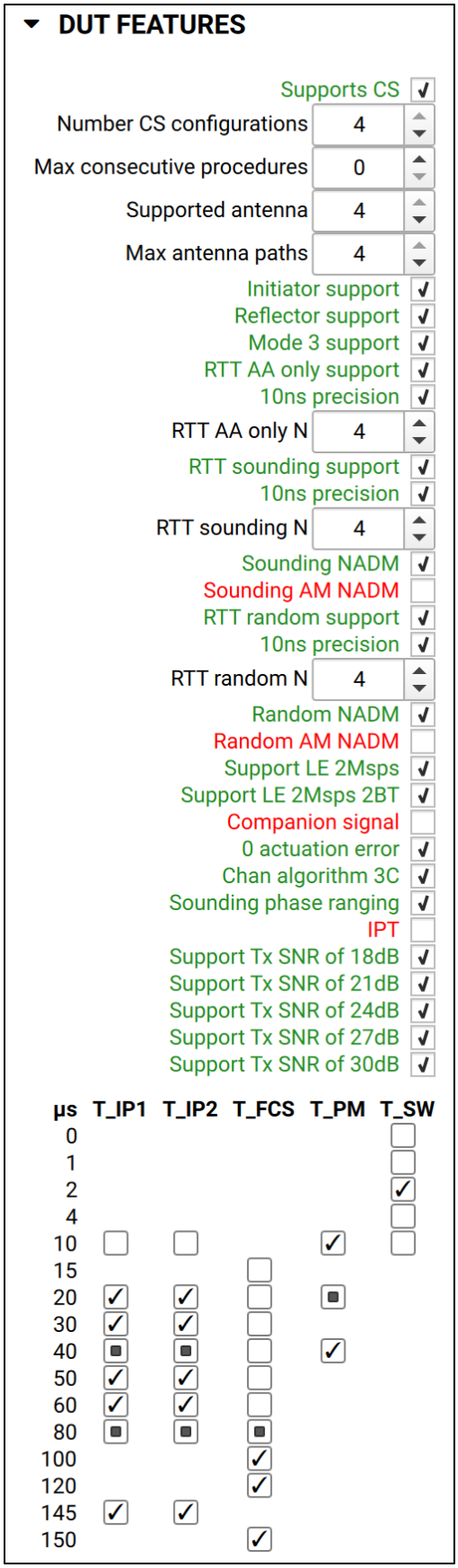
8.5 Setting supported features

Prior to any channel sounding procedure being run, it is essential that the CS features supported by the DUT are known to the application. This can be done in two ways:

1. Querying the supported features over HCI
2. Entering the supported features manually

To query the supported features over HCI, press the query button the toolbar. Once the query is complete, the “DUT features” menu will automatically be expanded so the results of the query can be viewed. If no DUT is connected when the query button is pressed, then an attempt will automatically be made to connect to any comports selected in the “DUT control” menu prior to issuing the HCI command.

The DUT supported features can be entered or altered manually at any time. The `CS_support…` commands are used to set specific features. If a query DUT supported features command has been executed, then the resulting values can be read back using the `CS_isSupported…` commands.


DUT FEATURES
Supports CS
Number CS configurations 4
Max consecutive procedures 0
Supported antenna 4
Max antenna paths 4
Initiator support
Reflector support
Mode 3 support
RTT AA only support
10ns precision
RTT AA only N 4
RTT sounding support
10ns precision
RTT sounding N 4
Sounding NADM
Sounding AM NADM
RTT random support
10ns precision
RTT random N 4
Random NADM
Random AM NADM
Support LE 2Msps
Support LE 2Msps 2BT
Companion signal
0 actuation error
Chan algorithm 3C
Sounding phase ranging
IPT
Support Tx SNR of 18dB
Support Tx SNR of 21dB
Support Tx SNR of 24dB
Support Tx SNR of 27dB
Support Tx SNR of 30dB
µs T_IP1 T_IP2 T_FCS T_PM T_SW
0
1
2
4
10
15
20
30
40
50
60
80
100
120
145
150

Control	Core specification parameter	API
Support CS	*Must be true*	
Number of CS configurations	*Ignored*	
Max consecutive configurations	*Ignored*	
Supported antenna	Num_Antennas_Supported	CS_supportNumAnt(uint32) CS_isSupportedNumAnt(uint32_t*)
Max antenna paths	Max_Antenna_Paths_Supported	CS_supportNumPath(uint32_n) CS_isSupportedNumPath(uint32_t*)
Initiator support	Roles_Supported[0]	CS_supportInitiator(bool) CS_isSupportedInitiator(bool*)
Reflector support	Roles_Supported[1]	CS_supportReflector(bool) CS_isSupportedReflector(bool*)
Mode 3 support	Modes_Supported[0]	CS_supportMode3(bool) CS_isSupportedMode3(bool*)
RTT AA only support	RTT_AA_Only_N != 0	CS_supportRTTAAonly(bool) CS_isSupportedRTTAAony(bool*)
RTT AA only precision	RTT_Capability[0]	CS_rttAAonlyPrecision(uint32_t) CS_isSupportedrttAAonlyPrecision(uint32_t*)
RTT AA only N	RTT_AA_Only_N	CS_supportRTTAAonlyN(uint32_t) CS_isSupportedRTTAAonlyN(uint32_t*)
RTT sounding support	RTT_Sounding_N != 0	CS_supportRTTsounding(bool) CS_isSuportedRTTsounding(bool*)
RTT sounding precision	RTT_Capability[0]	CS_rttSoundingPrecision(uint32_t) CS_isSupportedrttSoundingPrecision(uint32_t*)
RTT sounding N	RTT_Sounding_N	CS_supportRTTsoundingN(uint32_t) CS_isSupportedRTTsoundingN(uint32_t*)
Sounding NADM	NADM_Sounding_Capability[0]	CS_supportNADMsounding(bool) CS_isSupportedNADMsounding(bool*)
Sounding AM NADM	NADM_Sounding_Capability[1]	CS_supportAMsounding(bool) CS_isSupportedAMsounding(bool*)
RTT random support	RTT_Random_Payload_N != 0	CS_supportRTTrandom(bool) CS_isSupportedRTTrandom(bool*)
RTT random precision	RTT_Capability[0]	CS_rttRandomPrecision(uint32_t) CS_isSupportedrttRandomPrecision(uint32_t*)
RTT random N	RTT_Random_Payload_N	CS_supportRTTrandomN(uint32_t) CS_isSupportedRTTrandomN(uint32_t*)
Random NADM	NADM_Random_Capability[0]	CS_supportNADMrandom(bool) CS_isSupportedNADMrandom(*bool)
Random AM NADM	NADM_Random_Capability[1]	CS_supportAMrandom(bool) CS_isSupportedAMrandom(bool*)
Support LE 2Msps	CS_SYNC_PHYs_Supported[1]	CS_support2M(bool) CS_isSupported2M(bool*)
Support LE 2Msps 2BT	CS_SYNC_PHYs_Supported[2]	CS_support2M2BT(bool) CS_isSupported2M2BT(bool*)
Companion signal	Subfeatures_Supported[0]	CS_supportCompanion(bool) CS_isSupportedCompanion(bool*)
0 actuation error	Subfeatures_Supported[1]	CS_supportZeroActuationError(bool) CS_isSupportedZeroActuationError(bool*)
Channel algorithm 3C	Subfeatures_Supported[2]	CS_supportAlgorithm3C(bool) CS_isSupportedAlgorithm3C(bool*)
Sounding phase ranging	Subfeatures_Supported[3]	CS_supportSoundingPhaseRanging(bool) CS_isSupportedSoundingPhaseRanging(bool*)
IPT	Subfeatures_Supported[4]	CS_supportInlinePCT(bool) CS_isSupportedInlinePCT(bool*)
Support Tx SNR of 18dB	TX_SNR_Capability[0]	CS_supportTxSNR18dB(bool) CS_isSupportedTxSNR18dB(bool*)

Support Tx SNR of 21dB	TX_SNR_Capability[1]	CS_supportTxSNR21dB(bool) CS_isSupportedTxSNR21dB(bool*)
Support Tx SNR of 24dB	TX_SNR_Capability[2]	CS_supportTxSNR24dB(bool) CS_isSupportedTxSNR24dB(bool*)
Support Tx SNR of 27dB	TX_SNR_Capability[3]	CS_supportTxSNR27dB(bool) CS_isSupportedTxSNR27dB(bool*)
Support Tx SNR of 30dB	TX_SNR_Capability[4]	CS_supportTxSNR30dB(bool) CS_isSupportedTxSNR30dB(bool*)

The values of T_IP1, T_IP2, T_FCS, T_PM and T_SW can be selected using the table of checkboxes at the bottom of the "DUT features" menu. Greyed out boxes with ticks indicate mandatory settings. Checkboxes with squares rather than ticks indicate that these timings must be permitted if any shorter timings are selected.

The associated commands for setting these parameters are:

```
CS_supportT_IP1( uint32_t* T_IP1 , uint32_t N )

CS_supportT_IP2( uint32_t* T_IP2 , uint32_t N )

CS_supportT_FCS( uint32_t* T_FCS , uint32_t N )

CS_supportT_PM( uint32_t* T_PM , uint32_t N )

CS_supportT_SW( uint32_t T_SW )
```

where T_IP1 is an array of integers containing the permissible values of T_IP1 in units of µs. N is the number of elements in the array T_IP1. Similarly for the other API calls.

To check if a certain value of T_IP1, T_IP2, T_FCS or T_PM is supported by the DUT after a DUT query supported features has been executed use the functions:

```
CS_isSupportedT_IP1( uint32_t T_IP1 , bool *supported )

CS_isSupportedT_IP2( uint32_t T_IP2 , bool *supported )

CS_isSupportedT_FCS( uint32_t T_FCS , bool *supported )

CS_isSupportedT_PM( uint32_t T_PM , bool *supported )
```

Where `T_IP1, T_IP2, T_FCS, T_PM` are the values whose support is to be verified.

The T_SW supported by the DUT is obtained via:

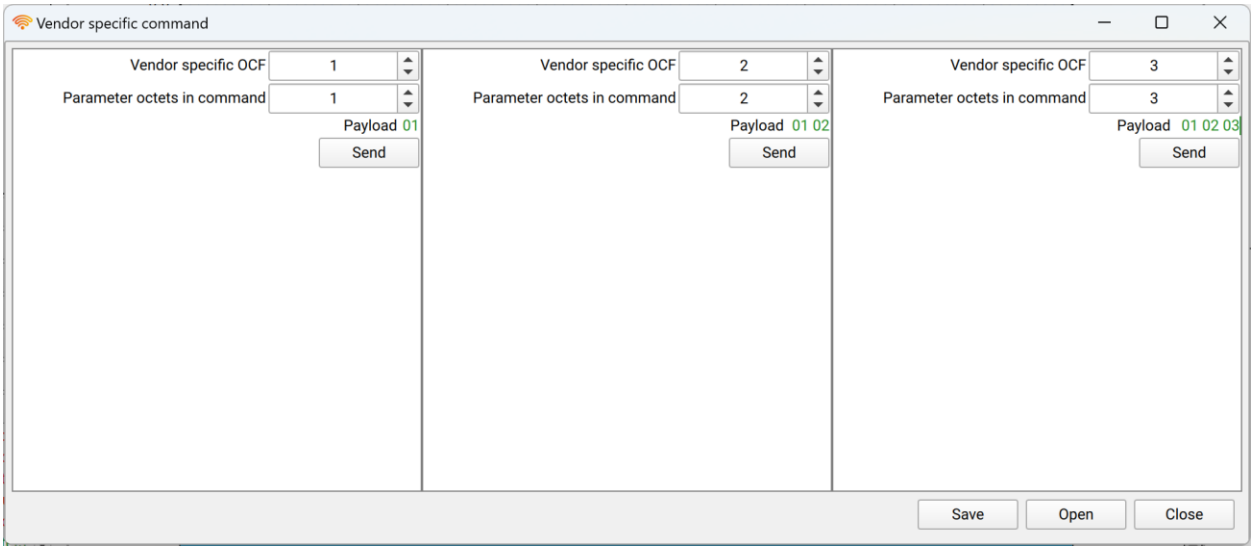
```
CS_isSupportedT_SW( uint32_t *T_SW )
```

after a DUT query supported features has been executed.

8.6 Sending additional HCI commands

Once a DUT is connected, it is possible to send additional HCI commands from the UI:

1. HCI reset
   An HCI reset command is sent to the DUT. The UART is closed after the HCI reset command is sent and then reopened. A “Read local version information” command is then issued to ensure the H4 link is still operational. The corresponding API call is `CS_HCIreset()`.
2. CS Test End
   A channel sound test end command is sent. This can be useful during initial debugging to ensure a DUT has completed the last HCI test command. The corresponding API call is `CS_HCItestEnd()`.
3. Vendor specific
   A vendor specific HCI command can be sent. A pop-up dialog allows the user to enter up to 3 vendor specific HCI commands.



For each vendor specific HCI command, it is possible to enter:

a. The vendor specific OCF
b. The number of octets in the payload
c. The payload data

The individual vendor specific HCI commands can be sent by clicking the associated “Send” button. All 3 vendor specific HCI commands can be saved/recalled from a single file by clicking the “Save” or “Open” buttons.

Individual vendor specific HCI commands can be sent via the API by calling `CS_HCI_vendor( unint32_t ocf , char * tx)`. The parameter `ocf` is the OCF which will be used in the HCI vendor specific command. The payload of the command is contained within the character string `tx`. This need only contain the payload data and not the payload count. The character string should contain the hex values of the octets to be sent, each octet separated by a space. For example, `"01 02 03"`.

8.7 Disconnecting

To disconnect the DUT(s) press the "Connect" button on the toolbar. The "Connect" button will return to its outline state. In the "DUT control" menu, all selected DUTs will now be shown in red with an associated cross. The DUT selection drop-down menu in the toolbar will now show the selected comport in red to indicate it is no longer available.

The corresponding API call is `CS_disconnectDUT( void )`.

9 Manual mode

9.1 Overview

In the manual mode of operation the user can construct a specific channel sounding procedure and then execute it. Once executed, the user can examine the collected data and quantities derived from it in detail.

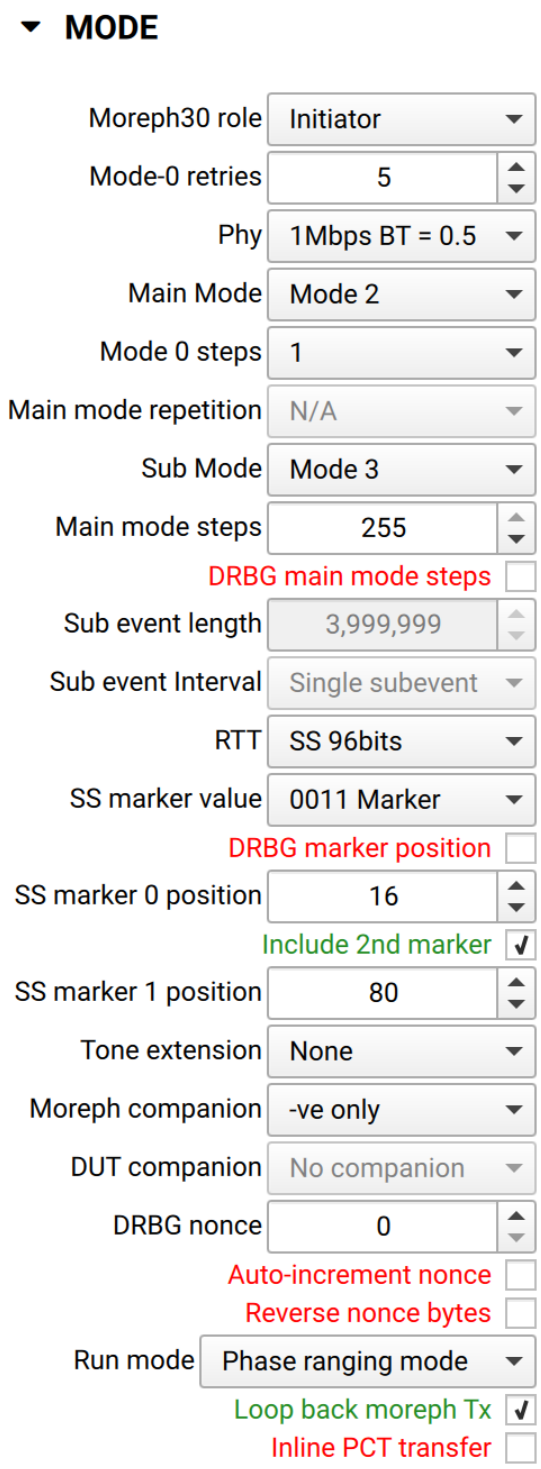
Manual mode can be run either with or without an HCI connection to the DUT.

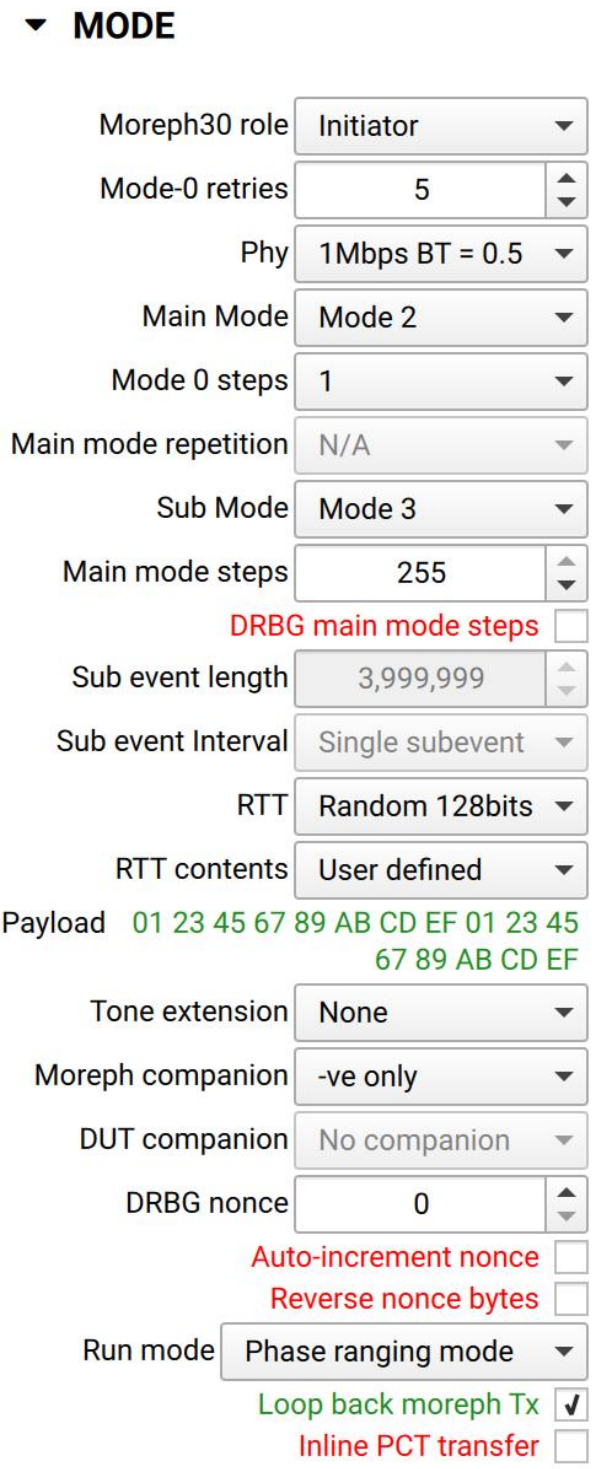
The channel sounding procedure is defined using the various menu. The parameters which can be adjusted are separated into 10 different categories:

1. Mode
   Defines the steps which are to be performed during the channel sounding procedure.
2. Distortions
   Defines attackers, frequency & phase errors, bit errors, timing jitter and AWGN contributions.
3. Timings
   Defines the timings of the steps in the channel sounding procedure as well as any clock offset to be imposed on the moreph transmissions.
4. Addresses
   Defines the addresses to be used by the initiator and reflector for each step in the channel sounding procedure.
5. Channels
   Defines the channels to be used in each step of the channel sounding procedure.
6. Power
   Defines the transmit powers of both the DUT and the moreph.
7. Antennas
   Defines which antennas and antenna paths are to be used. Also controls which tones are processed on the basis of their tone quality.
8. Switch/GPIO
   Provides manual control of the switch box and logic pod.
9. Cable
   Defines the group delay and loss between the moreph Tx/Rx port and the antenna connection for the DUT.
10. FAE
    The FAE table for the DUT.

9.2 Mode

The contents of the “Mode” menu will vary depending on the parameters selected. The range of parameter values which can be selected will be limited by the features supported by the DUT.





UI Item	Description	API call	API parameters
Moreph30 role	Selects between initiator and reflector.	`CS_setRole(role_t)`	`CS_Initiator` `CS_Reflector`
Mode-0 retries	Number of retransmissions of first mode-0 step when moreph is initiator if no response seen from reflector	`CS_setNumRetry(uint32_t)`	-

Phy	Selects phy to be used for CS procedure.	`CS_setPhy(phy_t)`	`CS_Phy1M`
			`CS_Phy2M`
			`CS_Phy2M2BT`

Main Mode	Selects main mode step type.	`CS_setMainMode(csmode_t)`	`CS_Mode1`
			`CS_Mode2`
			`CS_Mode3`
			`CS_StablePhi`

Mode 0 steps	The number of Mode 0 steps.	`CS_setMode0Steps(uint32_t)`	`1 to 3`
Main mode repetition	Ignored. Sub-events not currently supported.	-	-

Sub Mode	Mode to be used for sub-mode steps.	`CS_setSubMode(csmode_t)`	`CS_ModeNone`
			`CS_Mode1`
			`CS_Mode2`
			`CS_Mode3`

Main mode steps	The number of main mode steps inserted between sub-mode steps.	`CS_setMainModeSteps(int32_t)`	`1 to 255`
DRBG main mode steps	The number of main mode steps inserted between sub-modes is determined by the DRBG.	`CS_setMainModeSteps(-1)`	`-1`
Sub event length	Automatically calculated from the number of steps specified in the “Channels” menu.	-	-
Sub event interval	Currently only a single sub-event is supported.	-	-

RTT	Payload to be used for Mode-1 & Mode-3 steps.	`CS_setRTT(rtt_t)`	`CS_AAonly`
			`CS_Sounding32`
			`CS_Sounding96`
			`CS_Random32`
			`CS_Random64`
			`CS_Random96`
			`CS_Random128`

SS marker value	Sounding sequence marker value.	`CS_setSSmarkerValue(marker_t)`	`CS_marker0011`
			`CS_marker1100`
			`CS_markerAlt`
			`CS_markerDRBG`

SS marker 0 position	Position of first sounding sequence marker.	`CS_setSSmarker0pos(int32_t)`	`0 to 28`
SS marker 1 position	Position of second sounding sequence marker.	`CS_setSSmarker1pos(int32_t)`	`67 to 92`

UI Item	Description	API call	API parameters
DRBG marker position	The sounding sequence marker positions are determined by the DRBG.	CS_setSSmarker0pos(-1)	-1
Include 2nd marker	Determines whether a 2nd marker should be inserted in the sounding sequence.	CS_setSSmarker1pos(-1)	-1 will disable 2nd marker
RTT contents	Determines payload for Mode-1 & Mode-3 steps.	CS_setRTTpayload(payload_t)	CS_prbs9 CS_11110000 CS_10101010 CS_prbs15 CS_11111111 CS_00000000 CS_00001111 CS_01010101 CS_DRBG CS_user
Payload	User defined payload for Mode-1 & Mode-3 steps.	CS_setUser(uint8_t*,uint32_t)	First argument is an array of uint8_t containing the payload. The LSB of the first octet is transmitted first. The second argument is the number of octets in the array.
Tone extension	Determines when tone extensions will be inserted in Mode-2 & Mode-3 steps.	CS_setToneExtension(tone_t)	CS_toneExtNone CS_toneExtInitiator CS_toneExtReflector CS_toneExtBoth CS_toneExtLoop CS_toneExtDRBG
Moreph companion	Determines whether the moreph transmits companion signals.	CS_setMCompanion(companion_t)	CS_compNone CS_compNegOnly CS_compPosOnly CS_compBoth
DUT companion	Determines whether the DUT transmits companion signals.	CS_setDCompanion(companion_t)	CS_compNone CS_compNegOnly CS_compPosOnly CS_compBoth
DRBG nonce	Octets 14 and 15 of the initial DRBG nonce.	CS_setNonce(uint32_t)	0 to 255
Auto-increment nonce	The initial value of the DRBG nonce will be incremented by one every time a manual CS procedure is executed.	CS_setAutoIncrNonce(bool)	True False
Reverse nonce bytes	The two octets entered for the initial DRBG nonce will be reversed in the HCI test command.	CS_setRevNonce(bool)	True False
Run mode	The “Phase ranging mode” should be selected if phase measurements are to be made. This mode is	CS_setRunMode(run_t)	CS_runRTT CS_runPhase

UI Item	Description	API call	API parameters
	automatically selected if either the main mode or sub mode is set to mode-2 or mode-3. The “RTT mode” should be selected if RTT measurements are to be made. This mode is automatically selected if either the main mode or sub mode are set to mode-2 or mode-3.		
Loopback moreph Tx	When checked, the DAC output of the moreph is looped back to the ADC when the moreph is transmitting.		
Inline PCT transfer	When checked: If the moreph is an initiator, then the PCT values returned by the DUT will not be used. If the moreph is an reflector, then it will minimise the phase difference between its transmission and the signal received from the DUT.	`CS_setInlinePCT(bool)`	`True` `False`

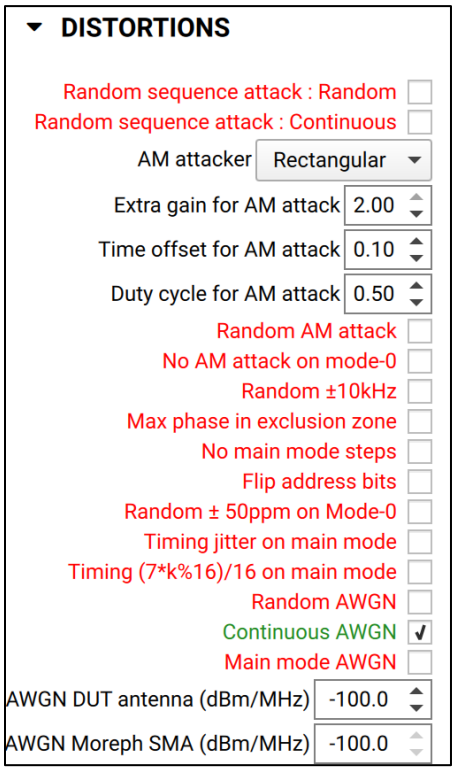
The option “Loopback moreph Tx” determines what data is captured when the moreph is transmitting. If this option is checked, then the DAC output of the moreph is looped back to the ADC input when the moreph is transmitting. A coarse delay is applied so that the transmission appears at approximately the correct time relative to the DUT transmission. No phase compensation is applied to the looped back signal. This option gives a good overview of the entire CS procedure, but the displayed moreph signal cannot be used for accurate timing or phase measurements. If this option is not checked, then the data displayed during the moreph transmissions is simply the energy reflected back down the moreph receiver chain. The reflection is uncontrolled, so the amplitude and phase of the signal are not predictable. The reflected signal also cannot be used for accurate timing or phase measurements.

9.3 Distortions

The “Distortions” menu determines what types of distortion will be imposed on the moreph transmissions and when these distortions are going to be added during the channel sounding procedure.

If the selected distortion is going to be added randomly, the presence of the distortion in a given step can be determined from the title of the upper graph. Change the time span of the graphics to be either “Mode-0 step” or “Main mode step”. If a distortion was added during the

displayed step, then the title of the upper graph will be modified to include “- random distortion added”.



UI Item	Description	API call	API parameters
Random sequence attack: Random	Implements Vol 6, Part H, 3.5.3.2 (Random sequence attack signal definition)to randomly selected steps.	`CS_setNADM(nadm_t)`	`CS_NADMnone` `CS_NADMrandom`
Random sequence attack: Continuous	Implements Vol 6, Part H, 3.5.3.2 (Random sequence attack signal definition) to all steps.	`CS_setNADM(nadm_t)`	`CS_NADMnone` `CS_NADMcontinuous`
Sound sequence attack	Implements Vol 6, Part H, 3.5.3.1 (Sounding sequence attack signal definition) to all steps.	`CS_setSSattack(bool)`	`True` `False`

Attacker guess	When implementing Vol 6, Part H, 3.5.3.1 (Sounding sequence attack signal definition) forces the attacker guess of the marker to be random,	`CS_setSSguess(guess_t)`	`CS_GuessRandom`
			`CS_GuessBad`
			`CS_GuessGood`

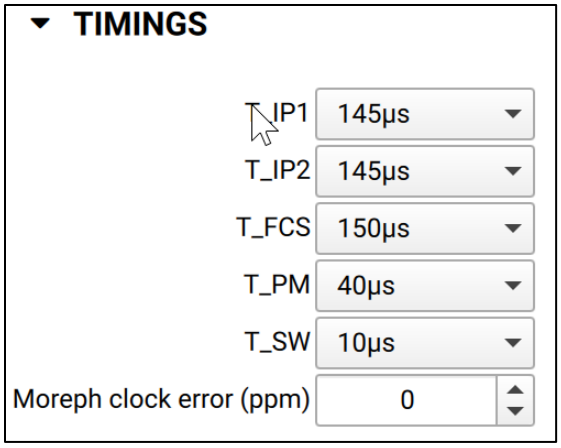
UI Item	Description	API call	API parameters
	always bad or always good.		
AM attacker	Implements proposed BT6.1 AM attacker. The AM pulse can be selected to be rectangular, sinusoidal or custom.	CS_setAMattack(am_t) CS_setAMcustom(char *)	CS_AMnone CS_AMrect CS_AMsine CS_AMcustom File name
Extra gain for AM attack	The increase in gain during the pulse of the AM attacker.	CS_setAMgain(double)	1.0 to 2.0
Time offset for AM attack	Time offset of the AM attacker pulse from start of symbol as a fraction of a symbol period. Resolution is 0.01 symbols.	CS_setAMoffset(double)	0.0 to 0.99
Duty cycle for AM attack	The duty cycle of the AM attacker pulse as a fraction of a symbol period. Resolution is 0.01 symbols.	CS_setAMduty(double)	0.01 to 0.99
Random AM attack	The steps within the CS procedure which are attacked are chosen at random.	CS_setAMrandom(bool)	True False
No AM attack on mode-0	The mode-0 steps of the CS procedure are never attacked.	CS_setAMmode0(bool)	True False
Random ±10kHz	A ±10kHz offset is applied randomly to each main mode step.	CS_set10KHz(bool)	True False
Max phase in exclusion zone	When transmitting CS tones, the phase is modified by π during the exclusion zones.	CS_setMaxPhi(bool)	True False
No main mode steps	Only the Mode-0 steps are transmitted.	CS_setNoMainMode(bool)	True False
Flip address bits	Main mode steps are transmitted with the kth bit in the address flipped, where k is the number of the main mode step.	CS_setFlipBits(bool)	True False
Random invalid markers	Sounding sequences are transmitted with invalid marker signals as per CS/PAC tests,	CS_setInvalidMarker(bool)	True False
Random ±50ppm on Mode-0	Mode-0 steps 1 & 2 have a random ±50ppm offset added to the carrier frequency.	CS_set50ppmMode0(bool)	True False
Timing jitter on main mode	Each main mode step is transmitted with a jitter randomly	CS_setJitter(jitter_t)	CS_JitterNone CS_JitterRandom

UI Item	Description	API call	API parameters
	sampled uniformly in the range ±1 symbol.		
Timing (7*k%16)/16 on main mode	Each main mode step is transmitted with a timing delay of (7*k%16)/16 of a symbol, where k is the main mode step number.	`CS_setJitter(jitter_t)`	`CS_JitterNone` `CS_JitterPeriodic`
Random AWGN	The moreph transmits each step with AWGN at the level specified by "AWGN at DUT antenna". On randomly selected steps the AWGN is reduced by 18dB.	`CS_setAWGN(awgn_t)`	`CS_AWGNnone` `CS_AWGNrandom`
Continuous AWGN	The moreph transmits each step with AWGN at the level specified by "AWGN at DUT antenna".	`CS_setAWGN(awgn_t)`	`CS_AWGNnone` `CS_AWGNcontinuous`
Main mode AWGN	The moreph transmits each main mode step with AWGN at the level specified by "AWGN at DUT antenna". No AWGN is added to the Mode-0 steps.	`CS_setAWGN(awgn_t)`	`CS_AWGNnone` `CS_AWGNmainMode`
AWGN at DUT antenna	The level of the AWGN at the DUT antenna port in units of dBm/MHz.	`CS_setAWGNlevel(double)`	`dBm/MHz`
AWGN at Moreph SMA	The level of the AWGN at the moreph Tx/Rx port in units of dBm/MHz. The displayed value is for information only.	-	-

9.4 Timings

The timings menu controls the timing of the individual channel sounding procedure steps. The range of parameters which can be selected is limited by the values entered in the "DUT features" menu.

The timings menu allows a clock error to be imposed on the moreph. This affects both the RF transmit frequency as well as the baseband symbol timing.

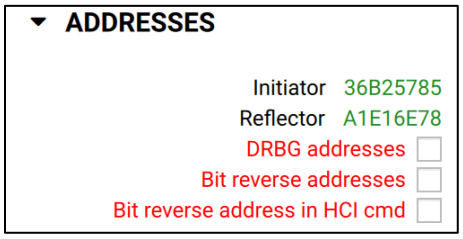


UI Item	Description	API call	API parameters
T_IP1	Idle time for Mode-1 steps	`CS_setT_IP1(uint32_t)`	`T_IP1 in µs`
T_IP2	Idle time for Mode-2 & Mode-3 steps	`CS_setT_IP2(uint32_t)`	`T_IP2 in µs`
T_FCS	Time allowed for frequency hop	`CS_setT_FCS(uint32_t)`	`T_FCS in µs`
T_PM	Phase measurement period for Mode-2 & Mode-3 steps	`CS_setT_PM(uint32_t)`	`T_PM in µs`
T_SW	Antenna switching period	`CS_set_SW(uint32_t)`	`T_SW in µs`
moreph clock error	The ppm error imposed on the moreph transmissions. Impacts both RF frequency and baseband timing.	`CS_setPPM(int32_t)`	`ppm error`

9.5 Addresses

The addresses menu defines the initiator and reflector address to be used for each step. These can either be fixed or derived from the DRBG.

If the addresses are fixed, then the initiator and reflector addresses are displayed in green. These can be edited by simply placing the cursor over the displayed address and typing over the characters.



UI Item	Description	API call	API parameters
Initiator	Fixed initiator address to be used in every step of the CS procedure.	`CS_setInitatorAddr(uint32_t)`	`Address`
Reflector	Fixed reflector address to be used in every step of the CS procedure.	`CS_setReflectorAddr(uint32_t)`	`Address`

UI Item	Description	API call	API parameters
DRBG addresses	Initiator and reflector addresses will be derived from the DRBG for each step in the CS procedure.	`CS_setDRBGAddr(bool)`	`True` `False`
Bit reverse addresses	Bit reverses the displayed initiator and reflector addresses.	-	-
Bit reverse address in HCI cmd	The initiator and reflector address are bit reversed in the HCI test command,	`CS_setBitRevAddr(bool)`	`True` `False`

9.6 Channels

The “Channels“ menu defines the frequency hopping sequence for the channel sounding procedure. The method of entering the frequency hopping sequence is different if a custom channel map is used compared to the use of frequency hopping algorithms #3b and #3c.

Use the radio buttons to select whether a custom map is to be used or whether frequency hopping algorithm #3b or #3c is to be used.

A graph is used to display the channels selected as a function of channel sounding procedure step.

9.6.1 Custom channel map

If a custom channel map has been selected then there are two options for defining the frequencies in the map:

1. Select one of the predefined sequences using the radio buttons.
2. Enter a list of channels to be used in the text box.

To enter a channel map into the text box separate the required channels by commas, eg:

2,3,5,50

It is invalid to enter channels 0,1,23,24,25,77,78 unless the “Restrict channels” box is unchecked.

A range of channels can be entered using *start channel : step : end channel*, eg:

2:4:76

Invalid channels in the entered range will be ignored. The step value can be negative to generate a ramp of decreasing channels.

Single channels and ranges of channels can be combined:

2,3,5:2:71,50,70:-1:20

On entering a new custom channel map, the channel map length field will automatically be reset to the length of the new channel map.

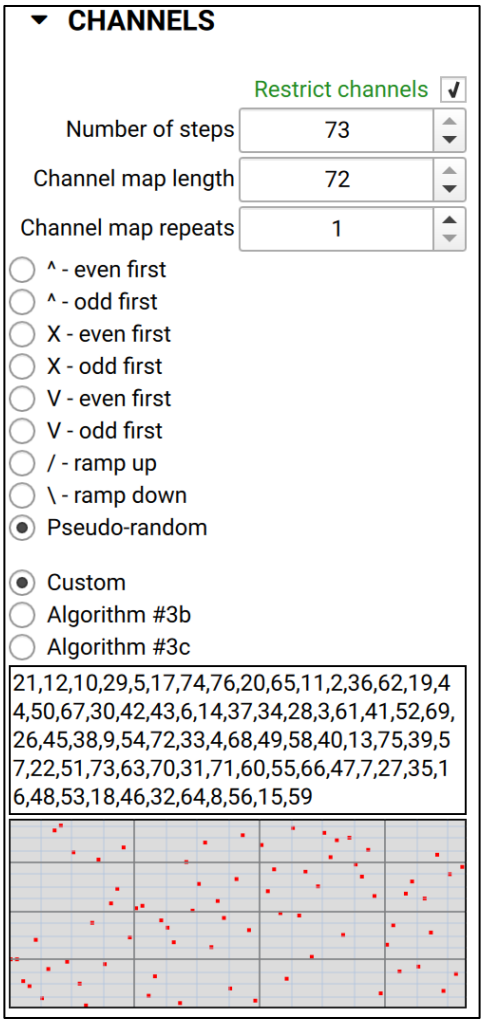
For a custom channel map, it is possible to set the channel map length independently of the text which has been entered. This allows for truncation of the channel map.

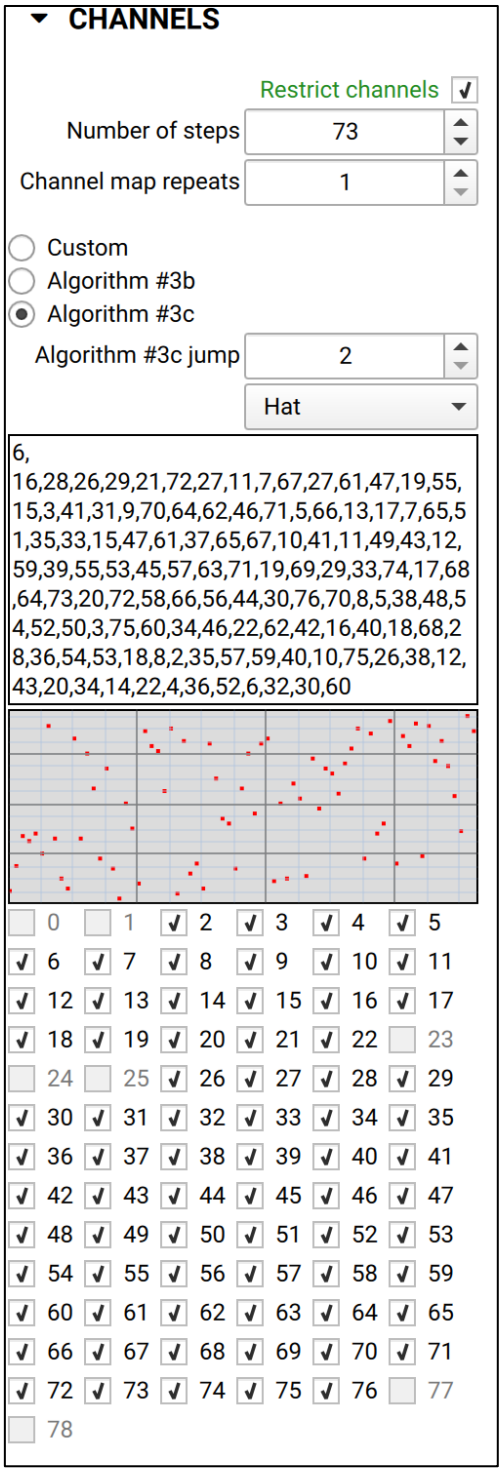
For a custom channel map, the index into the channel map is reset after the transmission of the Mode-0 steps.

9.6.2 Frequency hopping algorithms #3b and #3c

For frequency hopping algorithms #3b and #3c it is necessary to specify the channels which will be included in the map. This is done through a series of check boxes below the graph.

For frequency hopping algorithms #3b and #3c, the index into the channel map is **not** reset after the transmission of the Mode-0 steps.


CHANNELS
Restrict channels
Number of steps
73
Channel map length
72
Channel map repeats
1
^ - even first
^ - odd first
X - even first
X - odd first
V - even first
V - odd first
/ - ramp up
\ - ramp down
Pseudo-random
Custom
Algorithm #3b
Algorithm #3c
21,12,10,29,5,17,74,76,20,65,11,2,36,62,19,44,50,67,30,42,43,6,14,37,34,28,3,61,41,52,69,26,45,38,9,54,72,33,4,68,49,58,40,13,75,39,57,22,51,73,63,70,31,71,60,55,66,47,7,27,35,16,48,53,18,46,32,64,8,56,15,59


CHANNELS
Restrict channels
Number of steps
73
Channel map repeats
1
Custom
Algorithm #3b
Algorithm #3c
Algorithm #3c jump
2
Hat
6,
16,28,26,29,21,72,27,11,7,67,27,61,47,19,55,15,3,41,31,9,70,64,62,46,71,5,66,13,17,7,65,51,35,33,15,47,61,37,65,67,10,41,11,49,43,12,59,39,55,53,45,57,63,71,19,69,29,33,74,17,68,64,73,20,72,58,66,56,44,30,76,70,8,5,38,48,54,52,50,3,75,60,34,46,22,62,42,16,40,18,68,28,36,54,53,18,8,2,35,57,59,40,10,75,26,38,12,43,20,34,14,22,4,36,52,6,32,30,60
0 1 2 3 4 5
6 7 8 9 10 11
12 13 14 15 16 17
18 19 20 21 22 23
24 25 26 27 28 29
30 31 32 33 34 35
36 37 38 39 40 41
42 43 44 45 46 47
48 49 50 51 52 53
54 55 56 57 58 59
60 61 62 63 64 65
66 67 68 69 70 71
72 73 74 75 76 77
78

UI Item	Description	API call	API parameters
Restrict channels	Prohibits the use of channels 0,1,23,24,25,77 & 78.	CS_restrictChannels(bool)	True False
Number of steps	The total number of CS procedure steps to be executed. This must be less than or equal to the number of channels defined by the channel map length and the channel map repeats.	CS_setNumSteps(uint32_t)	2 to 160
Channel map length	The length of the channel map. This can be less than or equal to the number of channels described by the contents of the text box.	CS_setChanMapLength(uint32_t)	1 to 72
Channel map repeats	The number of times the channel map can be repeated.	CS_setChanMapRepeats(uint32_t)	
^ - even first	Ramp up using even channels and then ramp down using odd channels.	CS_setChanMapCustom(custom_t)	CS_mapUDeven
^ - odd first	Ramp up using odd channels and then ramp down using even channels.	CS_setChanMapCustom(custom_t)	CS_mapUDodd
X – even first	Starting with channel 2, alternate between high and low even channels. Once even channels are exhausted, swap to odd channels.	CS_setChanMapCustom(custom_t)	CS_mapXeven
X – odd first	Starting with channel 3, alternate between high and low odd channels. Once odd channels are exhausted, swap to even channels.	CS_setChanMapCustom(custom_t)	CS_mapXodd
V - even first	Ramp down using even channels and then ramp up using odd channels.	CS_setChanMapCustom(custom_t)	CS_mapDUeven
V - odd first	Ramp down using odd channels and then ramp up using even channels.	CS_setChanMapCustom(custom_t)	CS_mapDUodd
/ - ramp up	Ramp up through all channels.	CS_setChanMapCustom(custom_t)	CS_mapU
\ - ramp down	Ramp down through all channels.	CS_setChanMapCustom(custom_t)	CS_mapD
Pseudo-random	Use the pseudo-random channel map defined in the RF-PHY test specification.	CS_setChanMapCustom(custom_t)	CS_mapPseudo
Text box	Textual representation of channel map.	CS_setChanMapText(char *)	Channel map
Custom	Select a custom channel map.	CS_setChanMap(map_t)	CS_custom
Algorithm #3b	Select frequency hopping algorithm #3b to generate the channel map.	CS_setChanMap(map_t)	CS_3b
Algorithm #3c	Select frequency hopping algorithm #3c to generate the channel map.	CS_setChanMap(map_t)	CS_3c
Algorithm #3c jump	Jump parameter for frequency hopping algorithm #3c.	CS_set3cJump(uint32_t)	2 to 8

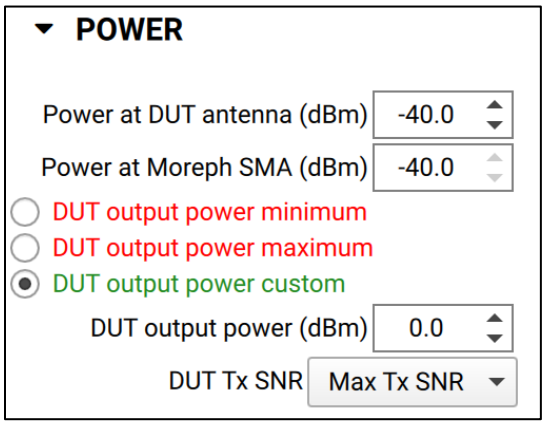
UI Item	Description	API call	API parameters
Algorithm #3c shape	Shape parameter for frequency hopping algorithm #3b.	CS_set3cShape(shape_t)	CS_hat CS_X
Channel checkboxes	Channels to be included in frequency hopping algorithms #3b and #3c.	CS_setChanSelection(uint32_t*)	An array of 3 uint32_t containing a bit map of the selected channels. Bit 0 of the first uint32_t corresponds to channel 0

9.7 Power

The “Power“ menu controls the output power of the moreph and also the transmit power of the DUT.

The output power of the moreph is always set by specifying the power required at the DUT antenna port. The external losses are then used to calculate the power at the moreph Tx/Rx port. This power is also displayed in the UI.

If the DUT supports SNR control, then the SNR of the DUTs transmissions can also be configured.

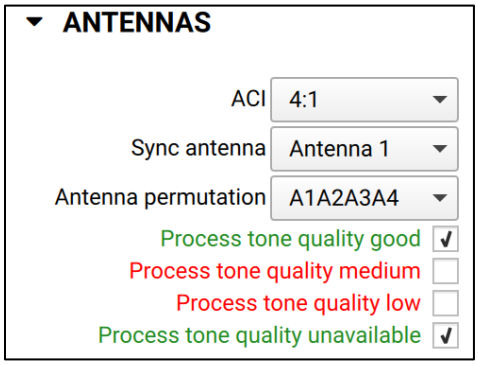


UI Item	Description	API call	API parameters
Power at DUT antenna	moreph transmit power measured at DUT antenna.	CS_setMorephPwr(double)	dBm
Power at Moreph SMA	moreph transmit power measured at moreph Tx/Rx port.	-	-
DUT output power minimum	Set the DUT to transmit at minimum output power.	CS_setDUTpwr(int32_t)	< -127
DUT output power maximum	Set the DUT to transmit at maximum output power.	CS_setDUTpwr(int32_t)	➢ 20

UI Item	Description	API call	API parameters
DUT output power custom	Set the DUT to transmit at a custom power level.	CS_setDUTpwr(int32_t)	dBm
DUT output power	Requested custom power level for DUT transmissions.	CS_setDUTpwr(int32_t)	dBm
DUT Tx SNR	The SNR value of the DUT transmissions.	CS_setDUTsnr(snr_t)	CS_snrMax CS_snr18 CS_snr21 CS_snr24 CS_snr27 CS_snr30

9.8 Antennas

The “Antennas“ menu defines which antenna paths are used throughout the channel sounding procedure. In addition, it also determines which CS tones are processed in linear regression calculations on the basis of their tone quality.



UI Item	Description	API call	API parameters
ACI	Antenna configuration index	CS_setACI(aci_t)	CS_aci_1_1 CS_aci_2_1 CS_aci_3_1 CS_aci_4_1 CS_aci_1_2 CS_aci_1_3 CS_aci_1_4 CS_aci_2_2 CS_aci_max_1 CS_aci_1_max Where max is the minimum of number of supported antennas and number of supported antenna paths
Sync antenna	Antenna used for CS_SYNC transmissions	CS_setSyncAnt(uint32_t)	1 to 4

UI Item	Description	API call	API parameters
Antenna permutation	Antenna permutation	CS_setPermAnt(perm_t)	CS_perm1234 CS_perm2134 CS_perm1324 CS_perm3124 CS_perm3214 CS_perm2314 CS_perm1243 CS_perm2143 CS_perm1423 CS_perm4123 CS_perm4213 CS_perm2413 CS_perm1432 CS_perm4132 CS_perm1342 CS_perm3142 CS_perm3412 CS_perm4312 CS_perm4231 CS_perm2431 CS_perm4321 CS_perm3421 CS_perm3241 CS_perm2341 CS_permLoop CS_permDRBG
Process tone quality good	Tones with quality good are included in linear phase regression calculations.	CS_setToneQualityGood(bool)	True False
Process tone quality medium	Tones with quality medium are included in linear phase regression calculations.	CS_setToneQualityMedium(bool)	True False
Process tone quality low	Tones with quality low are included in linear phase regression calculations.	CS_setToneQualityLow(bool)	True False
Process tone quality unavailable	Tones with quality set to unavailable are included in linear phase regression calculations.	CS_setToneQualityUnavailable(bool)	True False

9.9 Switch/GPIO

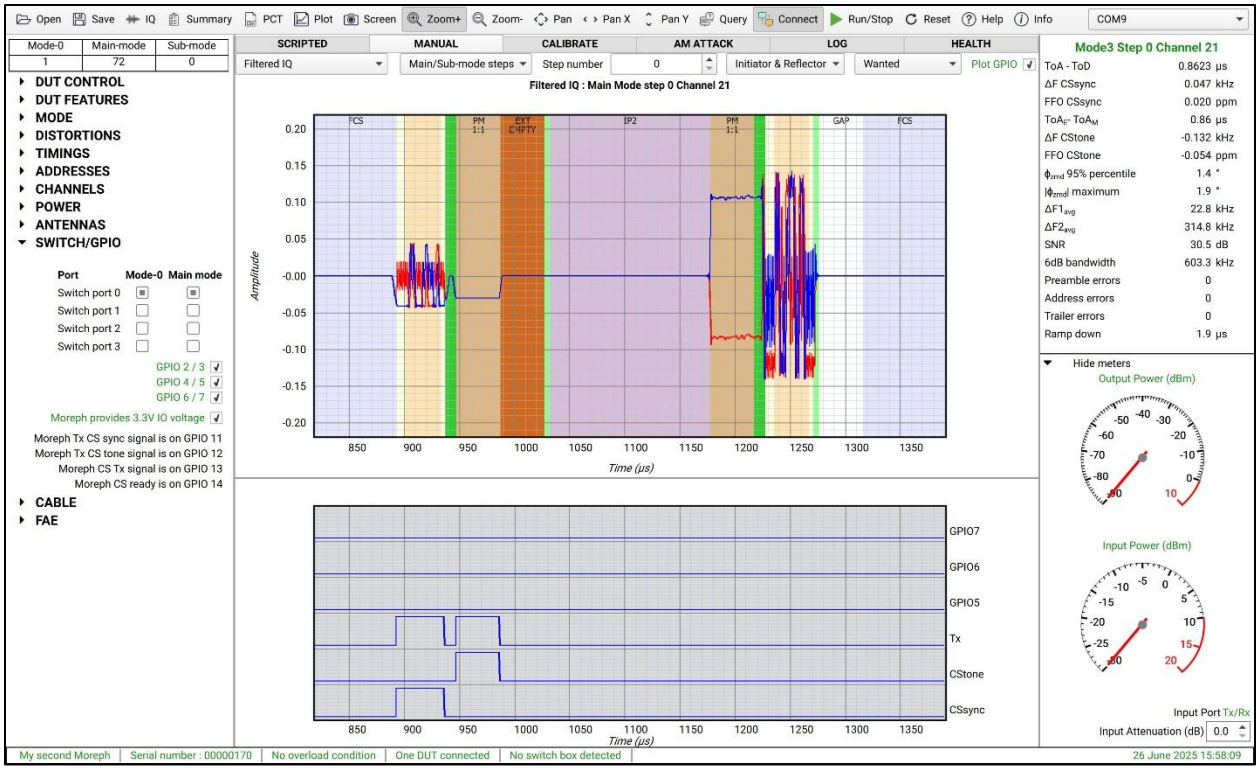
The “Switch/GPIO“ menu controls the operation of the external 8-way RF switch box and the external logic pod, if attached.

The external 8-way RF switch box controls which combination of DUT antennas are combined and connected to the moreph Tx/Rx port. Unused DUT antennas are terminated in 50Ω. Different combinations of DUT antennas can be specified for Mode-0 steps and for main mode steps. There is no switching of antennas within main mode steps.

If more than one DUT is attached to the external 8-way RF switch box, then “Switch port N” refers to the Nth antenna port on the DUT.

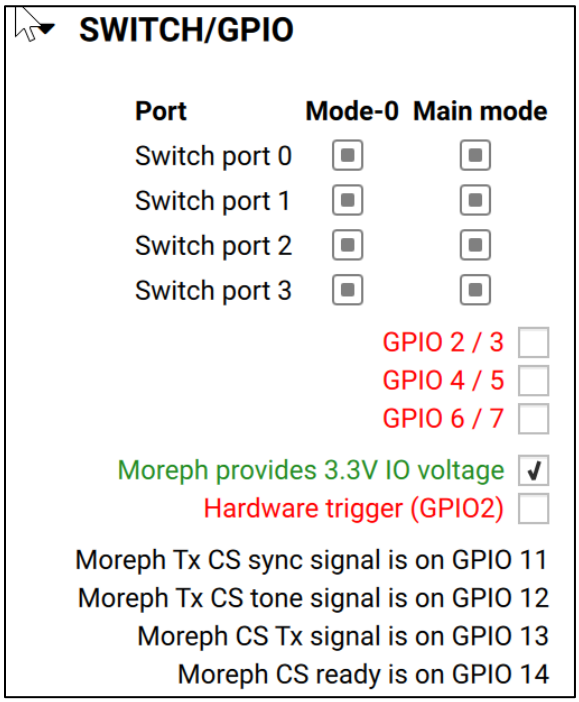
The logic pod can be used to sense digital signals on the DUT. These can then be displayed on the lower graph and referenced to analogue data displayed in the upper graph. Up to 6 logic lines can be sensed. These are located on GPIO 2 to 7. The number of GPIO sensed can be selected in pairs. The GPIO data is transferred from the moreph to the host computer using the LSB of the I/Q data. If GPIO 2/3 are selected, then the LSB of the I/Q data is lost, ie the IQ data is reduced from 16bit I/Q to 15bit I/Q. Hence for every pair of GPIO selected 1 LSB of dynamic range is lost. Therefore it is advisable to turn off the sensing of digital signals when accurate measurements of RTT or phase are required.

If the logic pod is attached, then the moreph will also output digital timing signals. GPIO 11 is used to indicate that the moreph is transmitting a CS_SYNC. GPIO 12 is used to indicate that the moreph is transmitting a CS tone. GPIO 13 is the OR of GPIO 11 and GPIO 12.



The labels on the right hand side of the logic plot can be edited by overtyping the existing text.

The IO voltage for the logic signals can either be 3v3 supplied by the moreph or an externally applied IO voltage. If the external 8-way switch box is attached then the IO voltage must be close to 3v3 since this is also used to power the circuitry within the switch box.



UI Item	Description	API call	API parameters
Switch port 0	Determines the state of switch port 0 during Mode-0 and main mode steps.	CS_setSwitchPort(uint32_t,bool,bool)	1st argument is port number (0) 2nd argument is state of port during Mode-0 steps. 3rd argument is state of port during main mode steps.
Switch port 1	Determines the state of switch port 1 during Mode-0 and main mode steps.	CS_setSwitchPort(uint32_t,bool,bool)	1st argument is port number (1) 2nd argument is state of port during Mode-0 steps. 3rd argument is state of port during main mode steps.
Switch port 2	Determines the state of switch port 2 during Mode-0 and main mode steps.	CS_setSwitchPort(uint32_t,bool,bool)	1st argument is port number (2) 2nd argument is state of port during Mode-0 steps. 3rd argument is state of port during main mode steps.
Switch port 3	Determines the state of switch port 3 during Mode-0 and main mode steps.	CS_setSwitchPort(uint32_t,bool,bool)	1st argument is port number (3) 2nd argument is state of port during Mode-0 steps. 3rd argument is state of port during main mode steps.
GPIO 2/3	Determines whether logic signals will be collected on GPIO 2/3.	CS_setLogic(logic_t,bool)	CS_logic23 True/False
GPIO 4/5	Determines whether logic signals will be collected on GPIO 4/5.	CS_setLogic(logic_t,bool)	CS_logic45 True/False
GPIO 6/7	Determines whether logic signals will be collected on GPIO 6.7/	CS_setLogic(logic_t,bool)	CS_logic67 True/False
Moreph provides 3v3 IO voltage	Determines whether the moreph provides the IO voltage or whether the IO voltage is supplied externally.	CS_setIOvolts(volts_t)	CS_moreph CS_external

If no DUT is connected, then the “Hardware trigger (GPIO2)” checkbox will be displayed. If this option is checked, then when the “Run” button is pressed the moreph will not start the CS

procedure until GPIO 2 goes high. This allows the CS procedure to be synchronised with external hardware.

GPIO 14 is set high prior to the moreph commencing a CS procedure. This signal can also be used to synchronise external hardware.

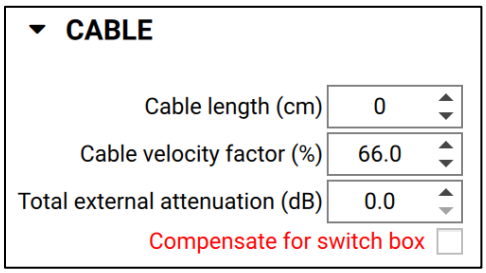
9.10 Cable

It is essential that the information in the “Cable” menu is entered accurately prior to making measurements. This applies to all modes of operation, apart from calibrate.

The cable length and the cable velocity factor are used to determine the group delay and phase delay between the moreph and the DUT. These are used to compensate the RTT ToF measurements and the linear phase regression measurements.

As an alternative to entering the cable length and cable velocity factor, it is possible to use the calibrate mode to calibrate to the end of the cable used to connect the moreph to the DUT. See section Calibration. In this instance the cable length should be specified as zero.

The total external attenuation is used by the moreph to adjust its transmit power to give the correct level at the DUT antenna port. If the switch box is used, then checking “Compensate for switch box” in manual mode will cause the moreph to automatically compensate for both the group delay and loss in the external 8-way RF switch box. In scripted mode, the moreph will automatically enable the 3v3 output on the IO connector and check to see if a switch box can be detected. The state of “Compensate for switch box” will then be set accordingly. It is not possible to override this behaviour in scripted mode.

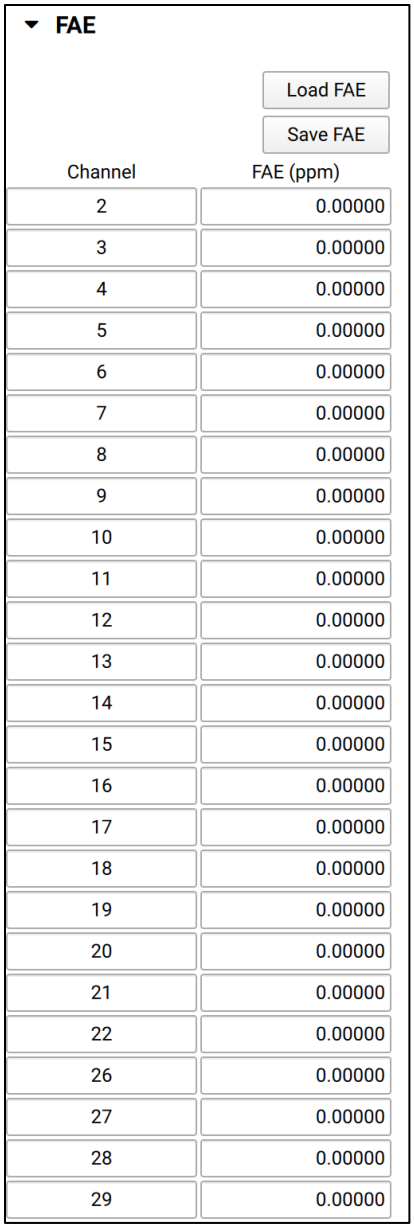


UI Item	Description	API call	API parameters
Cable length	Length of cable between moreph Tx/Rx port and DUT in cm.	`CS_setCableLength(double)`	`Cable length in cm`
Cable velocity factor	Velocity factor of cable used to connect moreph to DUT.	`CS_setCableVelcity(double)`	`Cable velocity factor as a percentage`
Total external attenuation	The total attenuation between moreph Tx/Rx port and DUT which must be compensated for.	`CS_setExtAttenuation(double)`	`Total external attenuation in dB`
Compensate for switch box	Check this box if the an external 8-way switch box is interposed between the moreph and the DUT and the loss and group delay through the switch box has not been included in the total external attenuation field and cable length field.	`CS_setSwitchBoxLoss(bool)`	`True` `False`

9.11 FAE

The HCI specification does not include a command for reading the local FAE table, presumably because this is held on the host and not the controller. The local FAE table is when the DUT is a reflector and the moreph must estimate the frequency offset of the Mode-0 tone. Therefore the FAE table must be entered manually.

Mechanisms are provided for saving and FAE table or loading an FAE table. These tables are held as XML files.


FAE
Load FAE
Save FAE
Channel
FAE (ppm)
2 0.00000
3 0.00000
4 0.00000
5 0.00000
6 0.00000
7 0.00000
8 0.00000
9 0.00000
10 0.00000
11 0.00000
12 0.00000
13 0.00000
14 0.00000
15 0.00000
16 0.00000
17 0.00000
18 0.00000
19 0.00000
20 0.00000
21 0.00000
22 0.00000
26 0.00000
27 0.00000
28 0.00000
29 0.00000

UI Item	Description	API call	API parameters
Load FAE	Loads an FAE table from an XML file. A file dialog pops-up to allow the used to select the file to load.	`CS_loadFAE(char*)`	File name of FAE XML file to be loaded
Save FAE	Saves the current FAE table to an XML file. A file dialog pops-up to allow the user to select a location where the file is to be saved.	`CS_saveFAE(char*)`	File name to which the FAE XML file will be written If the file already exists its contents will be overwritten
FAE (ppm)	The frequency actuation error in ppm for the associated channel.	`CS_setFAE(uint32_t,double)`	1st argument is channel number for FAE entry 2nd argument is the frequency actuation error in ppm

9.12 Running a channel sounding procedure

Prior to running a channel sounding procedure it is essential that the receiver front-end attenuation of the moreph is set correctly. Too little attenuation will result in the receive chain overloading whilst too much attenuation will unnecessarily degrade the receiver dynamic range.

The Tx/Rx port can accept signal up to approximately +8dBm without the need for any receiver front-end attenuation. For every dB the signal at the Tx/Rx port exceeds +8dBm and extra 1dB of receiver front-end attenuation should be added.

The receiver front-end attenuation is set using either the slider or spin box at the bottom right of the UI. The receiver front-end attenuation can be set using the call:

```
setInpAttenuation( double dB )
```

The resolution of the receiver front-end attenuator is 3dB.

The input power meter on the right-hand side will provide some indication of receiver overloading and how much margin exists. In addition, the status bar will also indicate overload conditions.

The channel sounding procedure is run by pressing the “Run” button on the toolbar. To run the channel sounding procedure from the API, call:

```
CS_runProcedure();
```

To abort a running or timed out channel sounding procedure:

```
CS_abortProcedure();
```

and to wait for a channel sounding procedure with a timeout in milli-seconds:

```
CS_waitProcedure( uint32_t timeout_ms );
```

If the moreph is configured as an reflector, then it will wait indefinitely until it sees a Mode-0 step on the correct channel and with the correct address from the initiator.

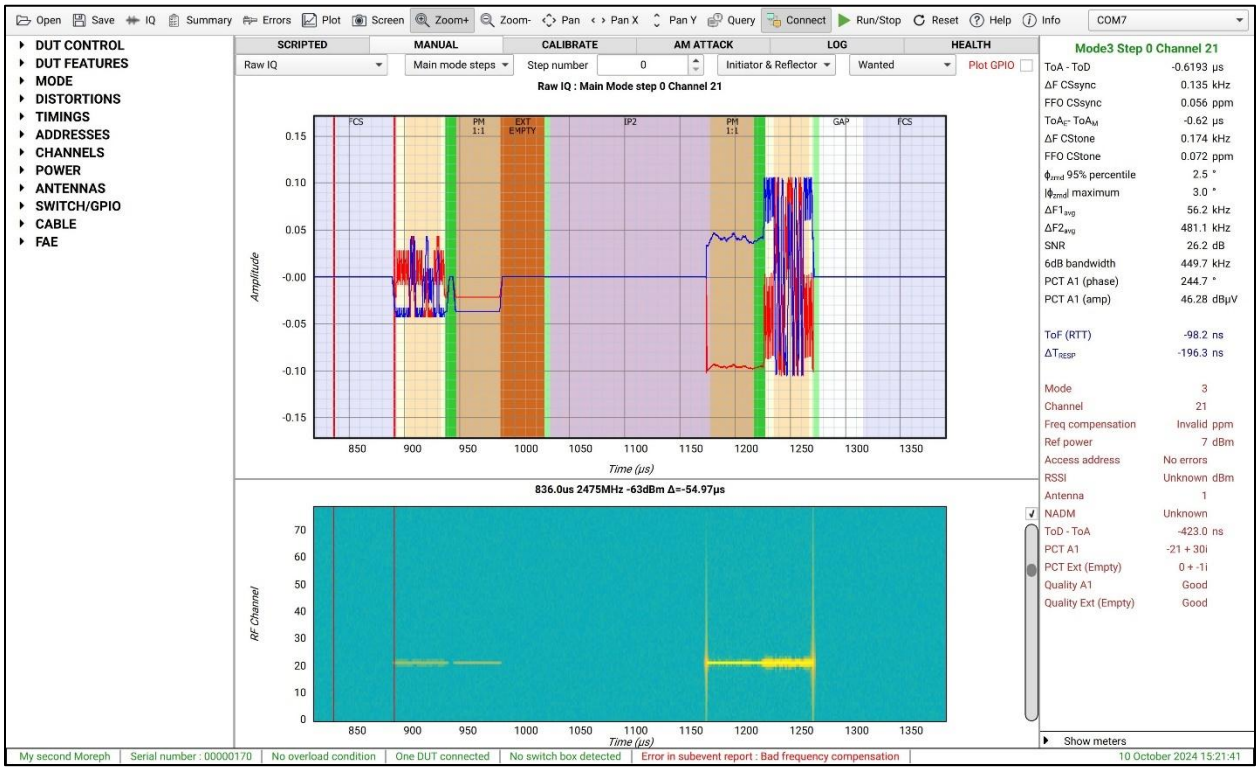
If the moreph is configured as an initiator, then it will instruct the DUT to commence a channel sounding procedure using the HCI CS test command. The moreph will then send a Mode-0 step. It is possible to alter the interval between the moreph receiving an response to the HCI CS test command and the moreph transmitting the first Mode-0 step. In the UI this interval is set by:

```
CS_setHCIinitiator( uint32_t ms );
```

9.13 Analysing a channel sounding procedure

9.13.1 Upper graph

Along the top of the upper graph are a number of menus. These determine what quantity is displayed in the graph and over what timespan.



The left most menu determines what quantity is displayed. See section Mode to understand what is displayed during the moreph transmit intervals. Possible options are:

1. Raw IQ
   This is the IQ data collected by the moreph. It has a sample rate of 20MSps. Two lines are plotted, red and blue, to denote the I and Q data.
2. Raw amplitude
   This is the magnitude of the raw IQ data. It has the same sample rate of 20MSps as the raw IQ.
3. Filtered IQ
   This is the IQ data after passing through the channel filter appropriate for current phy. It has a sample rate of 20MSps. Two lines are plotted, red and blue, to denote the I and Q data.
4. Filtered amplitude
   This is the magnitude of the raw IQ data. It has the same sample rate of 20MSps as the raw IQ.
5. Phase
   The phase of the filtered IQ data.
6. FM demodulated
   The FM demodulated data derived from the filtered IQ data. The FM demodulator differences phases which are spaced by $2/5^{th}$ of a symbol. The data is displayed so that the centre of the differentiator is aligned with the corresponding IQ sample. The red line is the output of the differentiator, the blue line is the best fit of the expected FM signal to the received signal.
7. Least squares fit
   The first part of the correlation process involves performing a least squares fit between the received FM signal and the expected FM signal. The least squares fit shows the output of this correlation process. It is only shown for a few µs either side of the expected correlation point.
8. ϕ_{zmd}
   The value of ϕ_{zmd}. This is calculated for any portion of the received signal where a tone is expected.
9. ΔF
   The estimated frequency offset between the moreph and the DUT. Separate values are shown for CSsync and tones. The values shown for CSyncs are calculated by performing a least squares fit between the received FM signal and the expected FM signal.
10. Φ_{meas}
    The phase measured by the moreph for each received tone. The value displayed is that at the expected start time of the tone.
11. Time-of-flight
    The estimated time-of-flight for each main mode step containing a CSsync. This calculation requires valid HCI subevent reports to be sent by the DUT. This graph will be empty if there are no main mode steps contain a CSsync.
12. Spectrum 100kHz RBW
    The measured peak hold spectrum of the received signal with an RBW of 100kHz. The 6dB bandwidth is shown by the blue lines. These can either be displayed individually for each step or a peak hold over the entire channel sounding procedure. The spectra are

gated between the start of the first bit of the preamble and the end of the last bit of the trailer or payload.

13. Spectrum 10kHz RBW
    A peak hold spectrum of received signal over the entire channel sounding procedure. The spectra used during the peak hold are gated between the mid-points of consecutive T_IP/T_FCS so that only the DUT signal is measured.
14. Linear regression (Phase)
    The linear regression over all Mode 2 and Mode 3 steps. The red line shows the measured phases (after calibration errors have been removed) and the blue line shows the linear regression. Different linear regression plots are available for each antenna path. This graph will be empty if there are no Mode 2 or Mode 3 steps.
15. Linear regression (sounding)
    The linear regression over all Mode 1 and Mode 3 steps which contain a sounding sequence payload. The red line shows the measured phases (after calibration errors have been removed) and the blue line shows the linear regression. This graph will be empty if there are no sounding sequences were transmitted.
16. Tone power vs channel
    The power of the tones measured in all Mode 2 and Mode 3 steps vs channel. Different plots are available for each antenna path. This graph will be empty if there are no Mode 2 or Mode 3 steps.
17. Sounding power vs channel
    The power in a single tone of a sounding sequence vs channel. This graph will be empty if there are no sounding sequences.

This menu can be controlled from the API by calling `CS_dispQuantity(menu_item)`. The argument is a character string with the same text as the menu item the user wishes to select.

The next menu alters the timespan for the graph. The spectrogram in the lower graph will always adopt the same timespan as the upper graph. Possible options are:

1. Full span
   The entire channel sounding procedure is displayed
2. Mode-0 steps
   A single Mode-0 step is displayed.
3. Main mode steps
   A single main mode step is displayed

This menu can be controlled from the API by using the `CS_dispRegion(menu_item)` function. The argument is a character string with the same text as the menu item the user wishes to select.

If either Mode-0 or main mode steps is selected, then further controls permit the user to select:

1. Which step to display. Steps are numbered from zero.
2. Whether to display initiator, reflector or both.

From the API, the displayed step can be controlled by calling `CS_dispStep(step_number)`. Whether initiator, reflector or both are displayed can be controlled by calling `CS_dispSignal(menu_item)`, where the argument is a character string containing the same text as the menu.

If companion signals are selected, then a further menu permit the plot to be changed between the upper companion signal, the lower companion signal or the wanted signal. From the API call `CS_dispCompanion(menu_item)`, where the argument is a character string containing the same text as the menu.

If linear regression plots or 10kHz spectrum plots are selected, then the timespan will revert to full span.

For the *Linear regression (phase)* and the *Tone power vs channel* plots, a drop down menu is available to select which antenna path to display.

9.13.2Lower graph

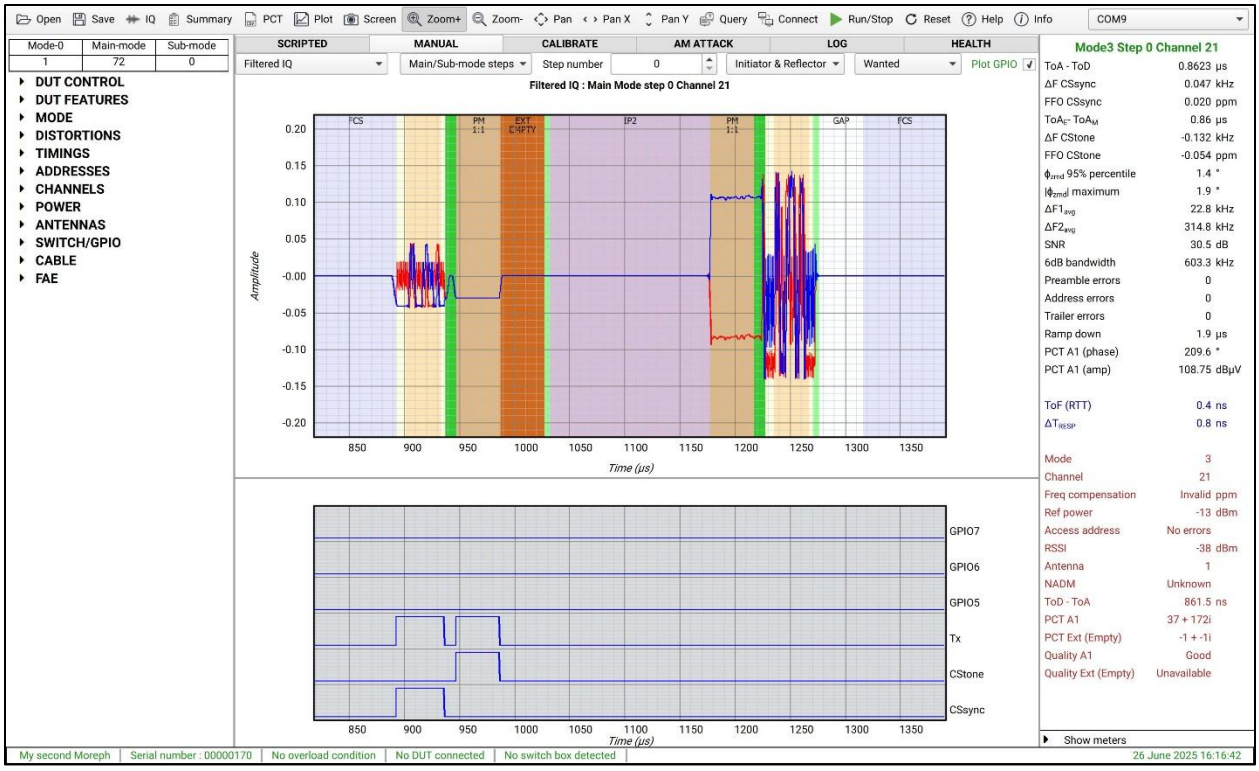
By default, the lower graph shows a spectrogram of the channel sounding procedure. The resolution of the spectrogram is 1MHz by 1µs. The title of the spectrogram shows the time/frequency/power of the point under the current cursor position. If Δ cursors are in operation (see Delta cursors), then the title also shows the time delta between the two cursor positions.

The contrast of the spectrogram can be controlled using the slider on the right hand side. The checkbox above the slider determines whether the spectrogram is displayed in grayscale or colour.

The spectrogram can only be zoomed in time, not in frequency. The spectrogram can be made to occupy more vertical space by pulling up the split window view and compressing the upper graph.

If logic probe data has been collected (see Switch/GPIO), then the “Plot GPIO” checkbox will appear on at the top right of the window. If this is checked, then the spectrogram is replaced by a plot of logic probe data.

The logic probe data is displayed as 6 horizontal traces, one line for each potential GPIO which can be selected. Unselected GPIO are drawn in the logic 0 state. The traces are labelled on the right hand side. The labels are editable.



9.13.3 Delta cursors

By default vertical coupled cursors will be displayed on both the upper and lower graphs. The title of the lower graph will show the corresponding cursor time.

If the user clicks on either the upper or lower graph, then a fixed cursor will be inserted into the plot. The title of the lower graph will now show the time delta between the fixed cursor and the active cursor. Clicking again will remove the fixed cursor.

9.13.4 Summary window

The contents of the summary depend on the timespan selected for the upper/lower graphs. If “Full span” is selected, then the quantities in the summary window relate to the entire channel sounding procedure, otherwise they relate to the selected step in the channel sounding procedure.

The information shown in the summary window is divided into 3 sections:

1. Upper section in black
   Measurements the moreph has made on the DUT transmissions.
2. Lower section in red
   Measurements made by the DUT on the moreph transmission and reported in the HCI CS Subevent Result event.
3. Mid-section in blue
   Quantities calculated by the moreph by combining measurements made by the moreph of the DUT transmissions with measurements made by the DUT on the moreph transmissions.

If “Full span” is selected for the time axis of the upper/lower graphs, then the following quantities may be displayed in the summary window:

Summary Quantity	DUT role	Description	API call parameter
Average SNR	Both	Average of SNR measurements over all steps within CS procedure.	CS_SNR
Min SNR	Both	Minimum of SNR measurements over all steps within CS procedure.	CS_minSNR
Max SNR	Both	Maximum of SNR measurements over all steps within CS procedure.	CS_maxSNR
Min ToA_E-ToA_M	Both	Minimum of expected arrival time minus actual arrival time over all steps within CS procedure.	CS_min_ToAe_ToAm
Max ToA_E-ToA_M	Both	Maximum of expected arrival time minus actual arrival time over all steps within CS procedure.	CS_max_ToAe_ToAm
Mode0 ΔF_{Tone}-ΔF_{CSsync}	Reflector	Difference in frequency offset of tone minus frequency offset of CS_SYNC for 1st Mode-0 step.	CS_Mode0_DFtone_DFsync
Min Mode0 ΔF_{Tone}-ΔF_{CSsync}	Reflector	Minimum of frequency offset of tone minus frequency offset of CS_SYNC over all Mode-0 steps.	CS_min_Mode0_DFtone_DFsync
Max Mode0 ΔF_{Tone}-ΔF_{CSsync}	Reflector	Maximum of frequency offset of tone minus frequency offset of CS_SYNC over all Mode-0 steps.	CS_max_Mode0_DFtone_DFsync
FFO Mode0 tone	Reflector	FFO of tone relative to moreph time base for 1st Mode-0 step.	CS_Mode0_FFO_tone
Min FFO Mode0 tone	Reflector	Minimum of FFO of tone relative to moreph time base over all Mode-0 steps.	CS_min_Mode0_FFO_tone
Max FFO Mode0 tone	Reflector	Maximum of FFO of tone relative to moreph time base over all Mode-0 steps.	CS_max_Mode0_FFO_tone
Min Mode0 tone FFO_0-FFO_k	Reflector	Minimum of FFO of tone for 1st Mode-0 step minus FFO of tone for kth Mode-0 step over all Mode-0 steps.	CS_min_Mode0_FFOo_FFOk
Max Mode0 tone FFO_0-FFO_k	Reflector	Maximum of FFO of tone for 1st Mode-0 step minus FFO of tone for kth Mode-0 step over all Mode-0 steps.	CS_max_Mode0_FFOo_FFOk
ϕ_{zmd} 95% percentile	Both	ϕ_{zmd} is calculated for every tone in the CS procedure. This value is the	CS_PHIzmd_95

Summary Quantity	DUT role	Description	API call parameter
		maximum 95% percentile over all measurements.	
Max $\lvert\phi_{zmd}\rvert$	Both	ϕ_{zmd} is calculated for every tone in the CS procedure. This value is the maximum absolute error over all measurements.	CS_max_abs_PHIzmd
Preamble errors	Both	Total number of preamble bit errors counted in the CS procedure.	
Address errors	Both	Total number of address bit errors counted in the CS procedure.	
Trailer errors	Both	Total number of trailer bit errors counted in the CS procedure.	
Avg ToF (RTT)	Both	Average of ToF measurements over all Mode-1/3 steps in the CS procedure.	CS_avg_ToF
STD ToF (RTT)	Both	Standard deviation of ToF measurements over all Mode-1/3 steps in the CS procedure.	CS_std_ToF
Min ToF (RTT)	Both	Minimum of ToF measurements over all Mode-1/3 steps in the CS procedure.	CS_min_ToF
Max ToF (RTT)	Both	Maximum of ToF measurements over all Mode-1/3 steps in the CS procedure.	CS_max_ToF
α N:M	Both	Value of phase linear regression slope, α, over all Mode-2/3 steps in the CS procedure relating to antenna path N:M.	CS_alpha_1_1 CS_alpha_1_2 etc
Avg fit error N:M	Both	Average error of phase linear regression over all Mode-2/3 steps in the CS procedure relating to antenna path N:M.	CS_avg_err_1_1 CS_avg_err_1_2 etc
STD fit error N:M	Both	Standard deviation of errors of phase linear regression over all Mode-2/3 steps in the CS procedure relating to antenna path N:M.	CS_std_err_1_1 CS_std_err_1_2 etc
Min fit error N:M	Both	Minimum error of phase linear regression over all Mode-2/3 steps in the CS procedure relating to antenna path N:M.	CS_min_err_1_1 CS_min_err_1_2 etc
Max fit error N:M	Both	Maximum error of phase linear regression over all Mode-2/3 steps in the CS procedure relating to antenna path N:M.	CS_max_err_1_1 CS_max_err_1_2 etc
$\Delta T_{RTT,PROC}$	Both	Value of $\Delta T_{RTT,PROC}$ for the CS procedure.	CS_DTrtt_proc
STD ΔT_{RTT}	Both	Standard deviation of ΔT_{RTT} values for CS procedure.	CS_std_DTrtt
Freq compensation	Both	Frequency compensation applied by DUT.	CS_freq_comp

If “Mode-0 steps” is selected for the time axis of the upper/lower graphs, then the following quantities may be displayed in the summary window:

Summary Quantity	DUT role	Description	API call
ΔF CSsync	Both	Frequency offset of CS_SYNC portion of Mode-0 step.	CS_DFsync
FFO of CSsync	Both	FFO of CS_SYNC portion of Mode-0 step.	CS_FFO_sync
ΔF CStone	Reflector	Frequency offset of tone portion of Mode-0 step.	CS_DFtone
FFO CStone	Reflector	FFO of tone portion of Mode-0 step.	CS_FFO_tone
ϕ_{zmd} 95% percentile	Reflector	The 95% percentile of ϕ_{zmd} measurements on Mode-0 tone.	CS_PHIzmd_95
$\lvert\phi_{zmd}\rvert$ maximum	Reflector	The maximum value of the absolute value of ϕ_{zmd} measured on the Mode-0 tone.	CS_max_abs_PHIzmd
ToA_E-ToA_M	Both	The expected arrival time minus the actual arrival time of the DUT transmission for this step.	CS_ToAe_ToAm
$\Delta F1_{avg}$	Both	Estimated value of $\Delta F1_{avg}$ for the GFSK modulated portion of the Mode-0 step. Applicable to 1Msps and 2Msps BT = 0.5 phy.	CS_DF1avg
$\Delta F2_{avg}$	Both	Estimated value of $\Delta F2_{avg}$ for the GFSK modulated portion of the Mode-0 step. Applicable to 1Msps and 2Msps BT = 0.5 phy.	CS_DF2avg
$\Delta F3_{avg}$	Both	Estimated value of $\Delta F3_{avg}$ for the GFSK modulated portion of the Mode-0 step. Applicable to 2Msps BT = 2.0 phy.	CS_DF3avg
$\Delta F4_{avg}$	Both	Estimated value of $\Delta F4_{avg}$ for the GFSK modulated portion of the Mode-0 step. Applicable to 2Msps BT = 2.0 phy.	CS_DF4avg
SNR	Both	Estimated SNR of modulated portion of Mode-0 step.	CS_SNR
6dB bandwidth	Both	6dB bandwidth of the Mode-0 step. The spectral measurement starts from the first bit of the preamble and ends at the last bit of the postamble.	CS_6dB_bandwidth
Preamble errors	Both	Number of preamble bit errors counted in the mode-0 step.	CS_preamble_errors
Address errors	Both	Number of address bit errors counted in the mode-0 step.	CS_aa_errors
Trailer errors	Both	Number of trailer bit errors counted in the mode-0 step.	CS_trailer_errors
Ramp down	Both	Ramp down time	CS_T_RD
Mode	Both	The mode of this step as reported by the DUT.	CS_mode
Channel	Both	The channel for this step as reported by the DUT.	CS_channel
Freq compensation	Both	The frequency compensation applied by the DUT.	CS_freq_comp
Ref power	Both	The reference power level reported by the DUT.	CS_ref_pwr
Access address	Both	The number of errors in the received access address as reported by the DUT.	CS_aa_errors
RSSI	Both	The RSSI reported by the DUT.	CS_RSSI
Antenna	Both	The antenna used by the DUT to receive the Mode-0 step.	CS_antenna
FFO this step	Initiator	Measured frequency offset reported by the DUT for this Mode-0 step.	CS_FFO

If “Main mode steps” is selected for the time axis of the upper/lower graphs, then the following quantities may be displayed in the summary window:

Summary Quantity	DUT Role	API call	API parameters
ToA – ToD	Reflector	Value of ToA – ToD as measured by the moreph after calibration corrections.	CS_ToA_ToD
ToD – ToA	Initiator	Value of ToD – ToA as measured by the moreph after calibration corrections	CS_ToD_ToA
ΔF CSsync	Both	Frequency offset of the CS_SYNC transmitted by the DUT as measured by the moreph.	CS_DFsync
FFO of CSsync	Both	FFO of the CS_SYNC transmitted by the DUT as measured by the moreph.	CS_FFO_sync
ΔF CStone	Both	Frequency offset of the CS tone transmitted by the DUT as measured by the moreph.	CS_DFtone
FFO CStone	Both	FFO of the CS tone transmitted by the DUT as measured by the moreph.	CS_FFO_tone
ϕ_{zmd} 95% percentile	Both	The 95% percentile of ϕ_{zmd} measurements on tones transmitted by the DUT.	CS_PHIzmd_95
$\lvert\phi_{zmd}\rvert$ maximum	Both	The maximum value of the absolute value of ϕ_{zmd} measured on tones transmitted by the DUT.	CS_max_abs_PHIzmd
ToA_E-ToA_M	Both	The expected arrival time minus the actual arrival time of the DUT transmission for this step.	CS_ToAe_ToAm
$\Delta F1_{avg}$	Both	Estimated value of $\Delta F1_{avg}$ for the GFSK modulated portion of the Mode-1/3 step. Applicable to 1Msps and 2Msps BT = 0.5 phy.	CS_DF1avg
$\Delta F2_{avg}$	Both	Estimated value of $\Delta F2_{avg}$ for the GFSK modulated portion of the Mode-1/3 step. Applicable to 1Msps and 2Msps BT = 0.5 phy.	CS_DF2avg
$\Delta F3_{avg}$	Both	Estimated value of $\Delta F3_{avg}$ for the GFSK modulated portion of the Mode-1/3 step. Applicable to 2Msps BT = 2.0 phy.	CS_DF3avg
$\Delta F4_{avg}$	Both	Estimated value of $\Delta F4_{avg}$ for the GFSK modulated portion of the Mode-1/3 step. Applicable to 2Msps BT = 2.0 phy.	CS_DF4avg
SNR	Both	Estimated SNR of modulated portion of Mode-1/3 step.	CS_SNR
6dB bandwidth	Both	6dB bandwidth of the Mode-1/3 step. The spectral measurement starts from the first bit of the preamble and ends at the last bit of the postamble.	CS_6dB_bandwidth
PCT1 (phase)	Both	Phase of the sounding sequence PCT1 measurement performed by the Moreph after calibration corrections have been applied.	CS_PCT1_phi
PCT1 (amp)	Both	Amplitude of the sounding sequence PCT1 measurement performed by the moreph after calibration corrections have been applied. Units of dB re 1μV.	CS_PCT1_amp
PCT2 (phase)	Both	Phase of the sounding sequence PCT2 measurement performed by the moreph after calibration corrections have been applied.	CS_PCT2_phi
PCT2 (amp)	Both	Amplitude of the sounding sequence PCT2 measurement performed by the moreph after calibration corrections have been applied. Units of dB re 1μV.	CS_PCT2_amp

PCT An (phase)	Both	Phase of the n^{th} Mode-2/3 tone PCT measurements made by the moreph after calibration corrections have been applied.	CS_PCT_A1_A1_phi CS_PCT_A2_A1_phi CS_PCT_A3_A1_phi CS_PCT_A4_A1_phi CS_PCT_A1_A2_phi CS_PCT_A1_A3_phi CS_PCT_A1_A4_phi CS_PCT_A2_A2_phi CS_PCT_Aext_phi
PCT An (amp)	Both	Amplitude of the n^{th} Mode-2/3 tone PCT measurements made by the moreph after calibration corrections have been applied. Units of dB re 1µV.	CS_PCT_A1_A1_amp CS_PCT_A2_A1_amp CS_PCT_A3_A1_amp CS_PCT_A4_A1_amp CS_PCT_A1_A2_amp CS_PCT_A1_A3_amp CS_PCT_A1_A4_amp CS_PCT_A2_A2_amp CS_PCT_Aext_amp
Preamble errors	Both	Number of preamble bit errors counted in the step. Not present for Mode-2 steps.	CS_preamble_errors
Address errors	Both	Number of address bit errors counted in the step. Not present for Mode-2 steps.	CS_aa_errors
Trailer errors	Both	Number of trailer bit errors counted in the step. Not present for Mode-2 steps.	CS_trailer_errors
Payload errors		Number of payload bit errors counted in the step. Not present for Mode-2 steps.	CS_payload_errors
Ramp down	Both	Ramp down time	CS_T_RD
ToF (RTT)	Both	ToF calculated by the moreph.	CS_ToF
ΔT_{RESP}	Reflector	Timing error of DUT.	CS_DTresp
ΔT_{RTT}	Initiator	Timing error of DUT.	CS_DTrtt
Mode	Both	The mode of this step as reported by the DUT.	CS_mode
Channel	Both	The channel for this step as reported by the DUT.	CS_channel
Freq compensation	Both	The frequency compensation applied by the DUT.	CS_freq_comp
Ref power	Both	The reference power level reported by the DUT.	CS_ref_pwr
Access address	Both	The number of errors in the received access address as reported by the DUT.	CS_aa
Payload errors	Both	The number of errors in the received payload as reported by the DUT.	CS_payload_err
RSSI	Both	The RSSI reported by the DUT.	CS_RSSI
Antenna	Both	The antenna used by the DUT to receive the RTT.	CS_antenna
NADM	Both	Packet NADM reported by the DUT.	CS_NADM
ToD – ToA	Reflector	Time of departure minus time of arrival reported by the DUT.	CS_ToD_ToA
ToA - ToD	Initiator	Time of arrival minus time of departure reported by the DUT.	CS_ToA_ToD
PCT1	Both	Phase compensation term 1 reported by DUT for sounding sequence.	CS_PCT1_phi CS_PCT1_amp
PCT2	Both	Phase compensation term 2 reported by DUT for sounding sequence.	CS_PCT2_phi CS_PCT_amp

PCT An	Both	The phase compensation term for antenna path n reported by the DUT.	CS_PCT_A1_A1_phi CS_PCT_A2_A1_phi CS_PCT_A3_A1_phi CS_PCT_A4_A1_phi CS_PCT_A1_A2_phi CS_PCT_A1_A3_phi CS_PCT_A1_A4_phi CS_PCT_A2_A2_phi CS_PCT_A1_A1_amp CS_PCT_A2_A1_amp CS_PCT_A3_A1_amp CS_PCT_A4_A1_amp CS_PCT_A1_A2_amp CS_PCT_A1_A3_amp CS_PCT_A1_A4_amp CS_PCT_A2_A2_amp
PCT Ext (Full)	Both	Phase compensation term for an occupied tone extension slot reported by the DUT.	CS_PCT_Aext_phi CS_PCT_Aext_amp
PCT Ext (Empty)	Both	Phase compensation term for an empty tone extension slot reported by the DUT.	CS_PCT_Aext_phi CS_PCT_Aext_amp
Quality An	Both	Tone quality for antenna path n reported by the DUT.	CS_Quality_A1_A1 CS_Quality_A2_A1 CS_Quality_A3_A1 CS_Quality_A4_A1 CS_Quality_A1_A2 CS_Quality_A1_A3 CS_Quality_A1_A4 CS_Quality_A2_A2
Quality Ext (Full)	Both	Tone quality for an occupied tone extension slot reported by the DUT.	CS_Quality_Aext
Quality Ext (Empty)	Both	Tone quality for an empty tone extension slot reported by the DUT.	CS_Quality_Aext

The results displayed in the summary window can be read back using the API command:

```
CS_readResult( results_t res , int32_t step , measure_t m , double &out )
```

The value of `res` indicates which quantity is to be read back, it's value being taken from the tables above. The value of `step` indicates which step in the channel sounding procedure the results should be taken from. The first Mode-0 step is step 0. If this parameter is set to -1, then an average overall steps is performed. The final parameter, `m`, is used to indicate whether the results refer to the moreph or the DUT. Possible values are `CS_Moreph`, `CS_DUT` and `CS_Both`.

9.13.5 Linear phase regression API

The data used in the Mode-2 linear phase regression can be read back over the API:

```
CS_getLinearRegressionPnts()
```

Returns the number of data points used in the linear phase regression.

```
CS_getLinearRegressionChan( uint32_t n )
```

Returns the channel number for the n^{th} data point in the linear regression.

```
CS_getLinearRegressionTheta( uint32_t n )
```

Returns the value of Θ for the n^{th} data point in the linear regression.

The corresponding functions to use for sounding sequence payloads are:

```
CS_getLinearRegressionSSPnts()

CS_getLinearRegressionSSChan( uint32_t n )

CS_getLinearRegressionSSTheta( uint32_t n )
```

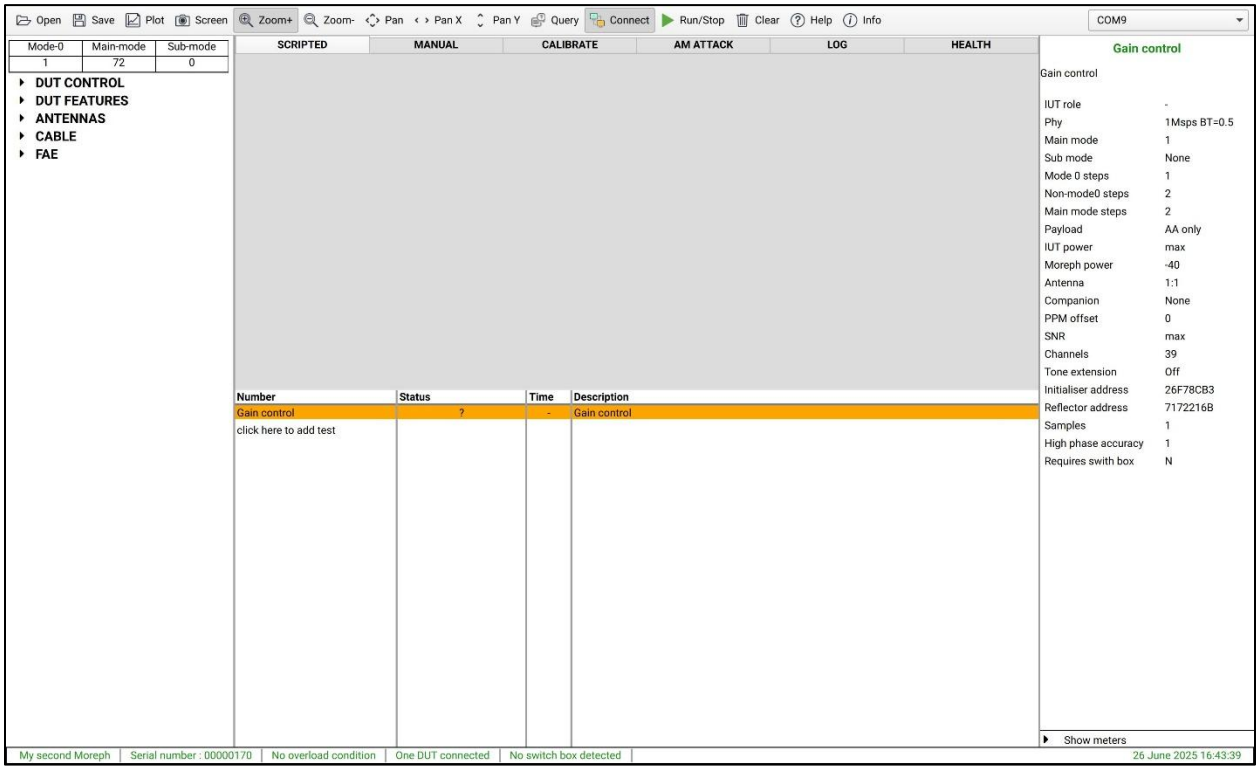
10 Scripted mode

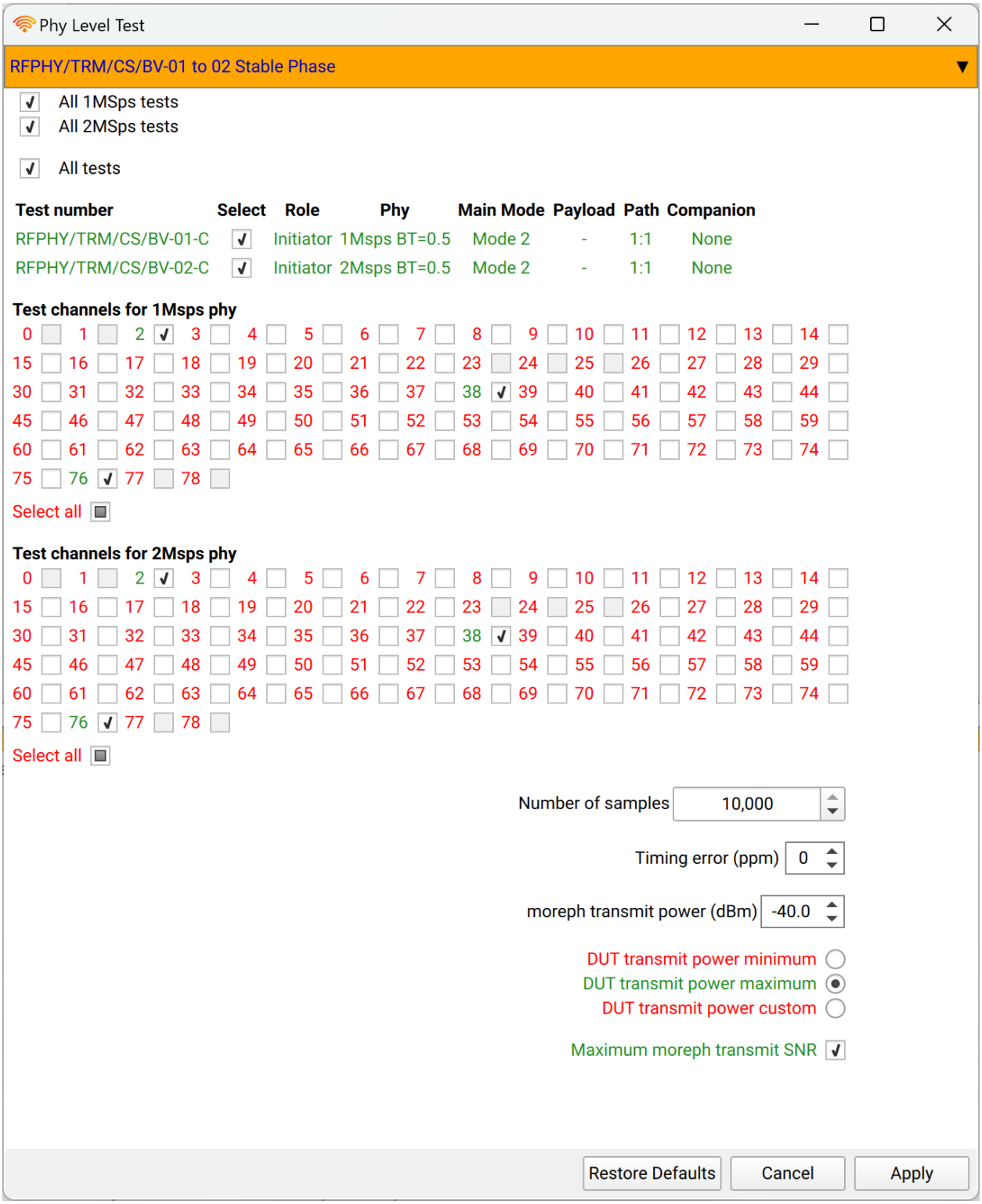
10.1 Overview

The scripted mode of operation permits the user to construct a list of tests to be executed, run the tests and then examine the results. All the tests are based on the Bluetooth test specifications. Some of the tests appeared in early versions of the Bluetooth test specifications but have since been removed.

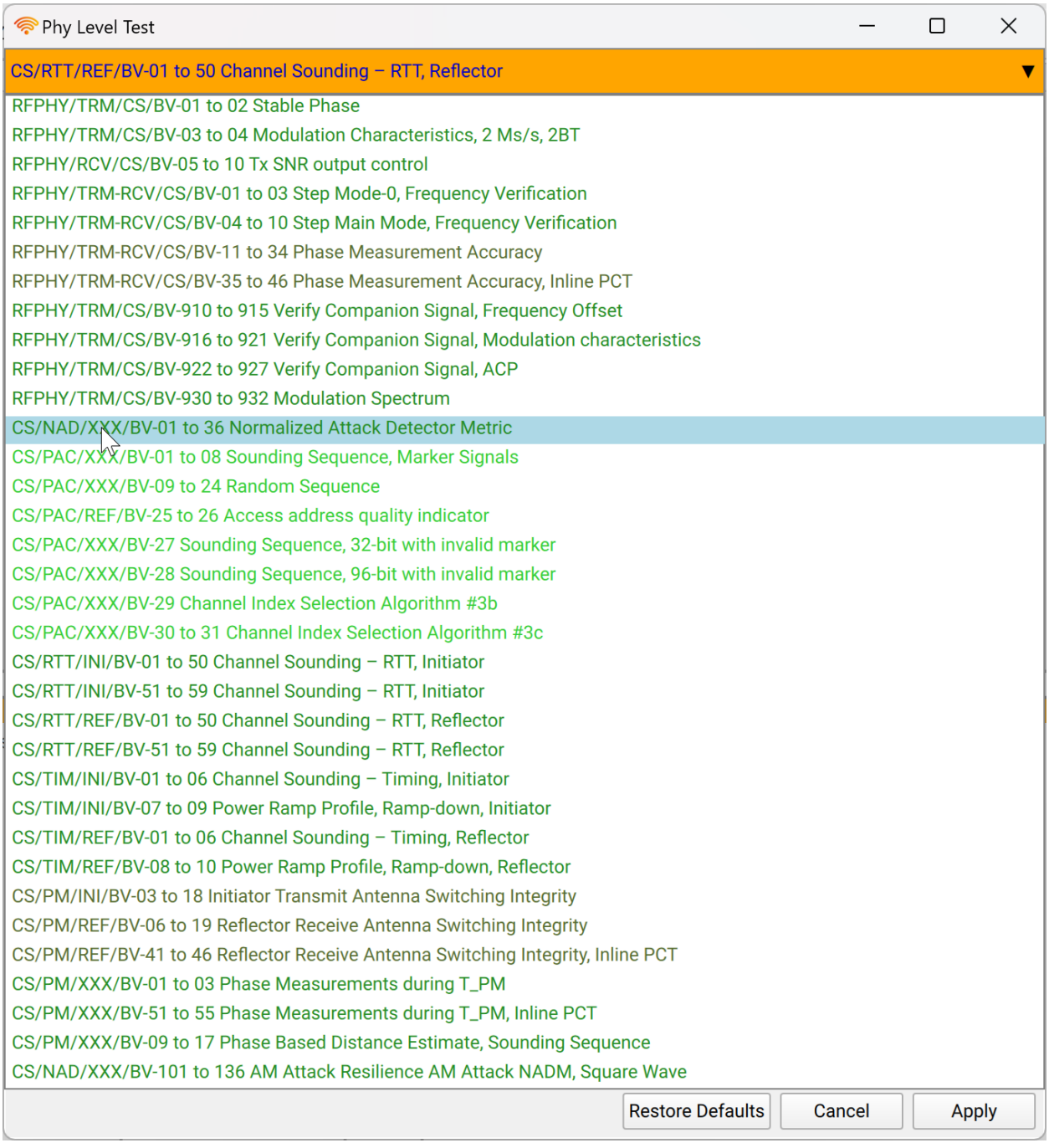
10.2 Creating a script

To start creating a script, select the “Scripted” tab and the click on “click here to add test”. A pop-up window will then appear listing the tests which can be added.




Phy Level Test
RFPHY/TRM/CS/BV-01 to 02 Stable Phase
All 1MSps tests
All 2MSps tests
All tests
Test number Select Role Phy Main Mode Payload Path Companion
RFPHY/TRM/CS/BV-01-C Initiator 1Msps BT=0.5 Mode 2 - 1:1 None
RFPHY/TRM/CS/BV-02-C Initiator 2Msps BT=0.5 Mode 2 - 1:1 None
Test channels for 1Msps phy
Select all
Test channels for 2Msps phy
Select all
Number of samples 10,000
Timing error (ppm) 0
moreph transmit power (dBm) -40.0
DUT transmit power minimum
DUT transmit power maximum
DUT transmit power custom
Maximum moreph transmit SNR
Restore Defaults
Cancel
Apply

The tests are grouped into sets of similar tests, for example the same test methodology but for different payloads or modes. A list of the test groups can be found in the drop down menu at the top of the pop-up window.



Select the group of tests you are interested in performing. The window will then show a list of all tests in that group together with key parameters such as role, phy, main mode, payload and whether a companion signal is present (not currently in 6.0 specification).

Individual tests can be selected or deselected using the checkbox next to the test name. Tests displayed in red cannot be selected because they required support for features which have not been selected in the “DUT features” menu.

The menu make contain quick selection checkboxes at the top of the window. These allow all tests to be selected or deselected as well as those tests pertaining to specific roles, phys, main modes or payloads.

Some tests require a channel sounding procedure to cover a number of channels, for example linear phase regression tests, whilst others are performed on specific channels, for example transmitter output SNR control. For those tests which are performed on specific channels, a series of checkboxes are provided to select the channels used by the test. A checkbox to select or deselect all channels is also provided. There are separate channel selections for each phy.

The “Number of samples” spin box provides a means to alter how long the test will run for. The meaning is test specific. Each test in the Bluetooth test specification defines a number for how many times a procedure should be repeated. For example, in the stable phase test the number of measurements of ϕ_{zmd} is defined, whilst in the Step Mode-0 Frequency Verification test the number of Mode-0 steps analysed is defined. The “Number of samples” can be used to modify these numbers in the test specification. This enables the tests to be run for shorted durations whilst debugging or for longer durations when testing for robustness.

The moreph ppm clock offsets at which the test will be run at can also be parameterised. If the test is usually run at a single clock offset, then a spin box will be present to permit the clock offset to be varied within the range ±100pm. If the test is usually run over a range of clock offsets, then a text box containing a comma separated list of ppm values will be present. This can be edited to select the require number of clock offsets and their values.

The moreph transmit power at which the test will be can be parameterised. A spin box is provided to allow this to be customised.

The DUT power used in the test can also be customised. Checkboxes are provided to permit the DUT to be set at minimum power, maximum power or a custom power level.

Finally the SNR at which the moreph transmits can also be varied. The moreph can either be made to transmit at maximum SNR or a user defined SNR. The SNR is varied by adding AWGN to the moreph transmissions.

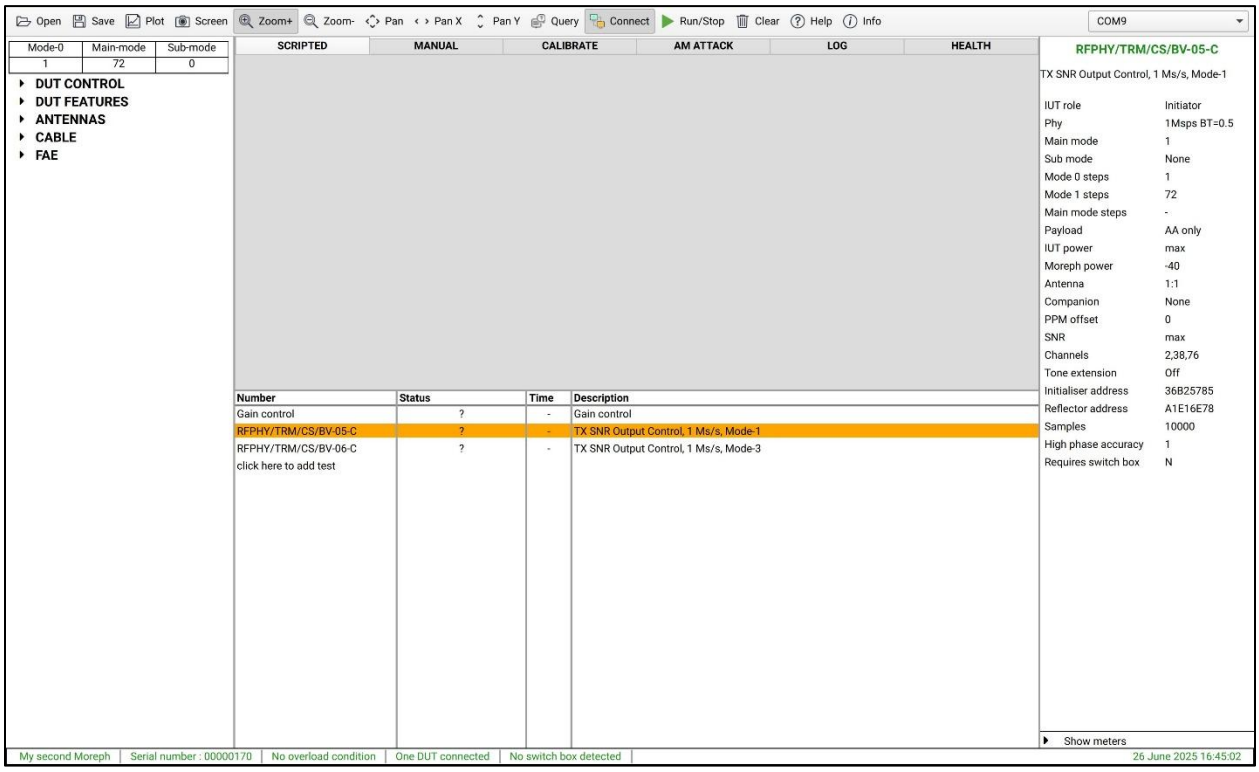
At the bottom of the window are various buttons:

1. Restore defaults
   Will select all available tests, restore the channel selection (if applicable) to the channels defined in the test specification and restore the “Number of samples” to the value defined in the test specification.

2. Apply
   Closes the pop-up window and enters the selected tests into the test script.
3. Cancel
   Closes the pop-up window without entering the selected tests into the test script.
4. Discard
   Only visible when editing existing tests in the test script. Will result in the group of tests being edited being deleted from the test script.



On closing the pop-up window by clicking “Apply” the selected tests are displayed in the test script. The details of the highlighted test are displayed in the summary window.



Further groups of tests can be added by clicking on “click here to add test” which is always displayed at the bottom of the test script.

A test can be edited by double clicking on it. When the test is edited, the entire group to which the test belongs is opened up for editing.

To delete a group of tests, double click on any test within the group and then click “discard” in the pop-up menu.

10.3 Gain control test

At the top of the test script there is a “Gain control” test which cannot be edited or deleted. When this test is run, a simple channel sounding procedure is executed with the DUT transmitting at maximum output power. The purpose of this test is to allow the moreph to measure the maximum output power of the DUT. This is then used to set the moreph input attenuation so as to maximise the dynamic range of the unit.

This test can be skipped by selecting “Skip gain control test” under the “DUT control” menu. However, if the test is skipped it is essential the moreph input attenuation is set prior to run the test script. The input attenuation should be adjusted such that the input stages to the moreph do not overload when the DUT is transmitting at maximum output power. Overload conditions

are shown in the status bar and by the label over the “Input power” meter on the right hand side of the window turning red.

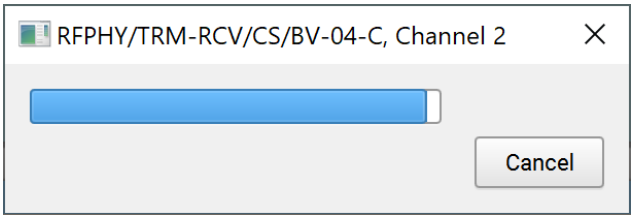
10.4 Running a script

Prior to running a script it will be necessary to:

1. Connect to a DUT
2. Ensure that the “DUT features” menu accurately represents the features the DUT supports, either by using the “Query” button on the toolbar or entering the values manually.
3. Generate a script to run.

The script is run by clicking the green “Run” icon on the toolbar which will then turn into a red “Stop” icon.

Whilst the test script is running a pop-up window will appear showing the progress of each test. The title shows which test is being executed, the channel being tested (if applicable) and the fraction of the test executed. This window also contains a “Cancel” button which can be used to abort the execution of the test script.



As each test is executed a green tick or red cross is added to the status column in the test script to indicate whether the test passed or failed. For tests which are performed on more than one channel a tick or cross is added for each channel tested. The time column is also updated to show how long the test took to complete.

To run a script from the API, call:

```
CS_runProcedure();
```

To abort a running test script call:

```
CS_abortProcedure();
```

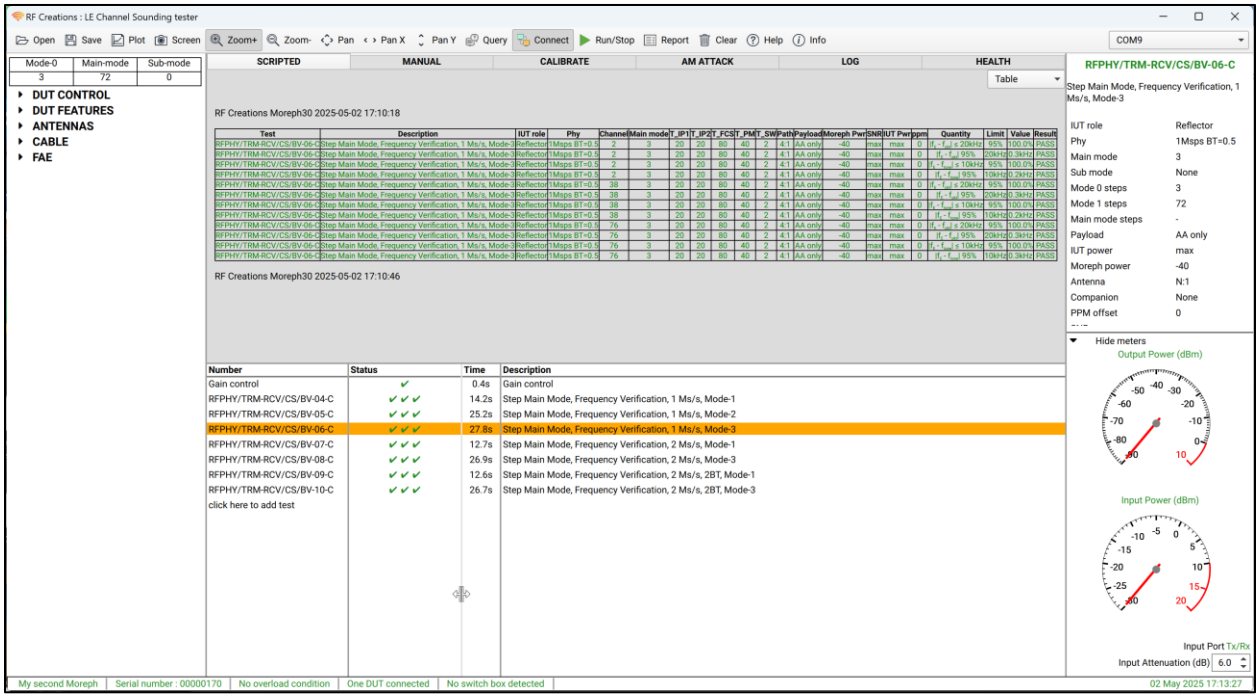
and to wait for a test script to complete channel sounding procedure with a timeout in milli-seconds:

```
CS_waitProcedure( uint32_t timeout_ms );
```

10.5 Examining results

After a test script has been run the status column in the test script will indicate whether the test passed or failed. For tests which run on a single channel, there will be one tick or cross per channel the test was executed on. An orange warning triangle will be displayed if it was not possible to run the test. The time column will indicate the total test time for each test.

The numerical or graphical results for a particular test can be displayed by highlighting the test in the test script. The results will then be displayed in the upper portion of the window.



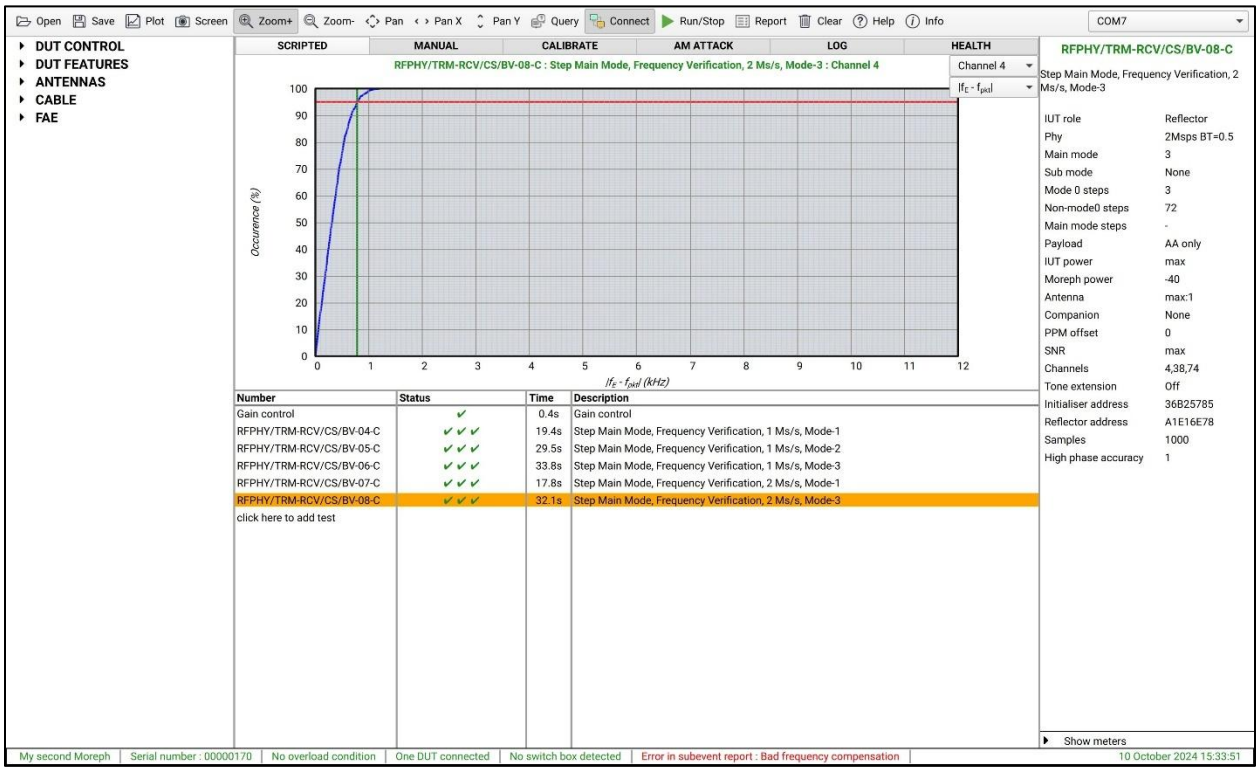
At the top right of the window is a drop down menu which enables the results to be displayed in different ways.

The first option in the menu is for a tabular view of the results. The table shows the test number, test description and the parameters used in the test. The last four columns show:

1. Quantity
   The quantity which is being tested.
2. Limit
   The numerical value of the limit which the quantity must satisfy.
3. Value
   The value of the measured quantity.
4. Result
   A pass of fail result determined by comparing the measured quantity with the limit.

Tests which pass are printed in green and those that fail in red. Lines printed in blue are information lines and are designed to show additional numerical information which might aid in understanding why a test passed or failed.

The drop down menu may also provide options for displaying the results graphically. There may be different graphs available for the different channels tested, different quantities measured or runs of the test with different test parameters.



The graphs may contain three different coloured lines:

1. Blue
   These lines show the measured data.
2. Red
   These lines represent the limit lines for the test.
3. Green
   There lines represent where the measured data met the test limits.

The results of the tests can be exported in html format by clicking the “Report” button on the toolbar. The exported data will include both the tabular results and the graphical results.

All the results can be cleared by clicking the “Discard” button on the toolbar.

10.6 Debugging test scripts

If a test script fails, delete items from the test script until the script contains a single failing test case. Run the test script with this single failing test. Once the test script has run, switch to manual mode. On switching to manual mode:

1. All the channel sounding procedure settings will be set to those used by the last channel sounding procedure run in the failing test script.
2. The data displayed in manual mode will be the data from the last channel sounding procedure run in the failing test script.

It may now be possible to examine the displayed data to determine why the test failed.

Sometimes it is not the last channel sounding procedure which was run in scripted mode which failed. For example, many tests are run at +50ppm, 0ppm and -50ppm moreph clock offsets. The tests which fail might relate to -50ppm, whilst the last tests run might relate to +50ppm. However, the majority of the channel sounding procedure parameters will be set correctly for the failing test case and it is just a matter of adjusting one or two which may have changed before re-running the test manually. For the example above, it would be necessary to reset the moreph clock to the failing -50ppm setting and then run a procedure in manual mode.

10.7 Running test scripts from the API

After connecting to the moreph and launching the channel sounding application, the application should be placed into scripted mode:

```
CS_setOperatingMode( CS_scripted );
```

A test script should then be loaded from a file:

```
CS_loadTestScript( filename );
```

and the test script executed:

```
CS_runProcedure();
```

Wait until the test script completes with a timeout in milli-seconds:

```
CS_waitProcedure( timeout_ms );
```

The results from running the test script are contained in a single table. The first index into the table is 0. The row of a given test within the table can be obtained by calling:

```
CS_findTest( const char *testName, uint32_t  occurrence, int32_t &row);
```

where the `testName` is a character string containing either a portion or the entire test name. `occurrence` is an unsigned integer specify which occurrence of `testName` should be searched

for. A value of 0 will return the first occurrence. The function returns the corresponding row in the table or -1 if the specified test name and occurrence are not found or the test is still running.

For a given row in the results table it is possible to read back the following quantities:

Quantity	API call
Test name	`CS_getTestName( uint32_t row , char *testName , uint32_t n )`
Test description	`CS_getTestDescription(uint32_t row ,char *descrip ,uint32_t n)`
DUT role	`CS_getTestRole( uint32_t row , role_t *role )`
CS procedure Phy	`CS_getTestPhy( uint32_t row , phy_t *phy )`
CS procedure channel	`CS_getTestChan( uint32_t row , int32_t *chan )`
Main mode	`CS_getTestMainMode( uint32_t row , int32_t *mainMode )`
Payload	`CS_getTestPayload( uint32_t row , rtt_t *payload )`
T_IP1	`CS_getTestT_IP1( uint32_t row , int32_t *T_IP1 )`
T_IP2	`CS_getTestT_IP2( uint32_t row , int32_t *T_IP1 )`
T_FCS	`CS_getTestT_FCS( uint32_t row , int32_t *T_IP1 )`
T_PM	`CS_getTestT_PM( uint32_t row , int32_t *T_IP1 )`
T_SW	`CS_getTestT_SW( uint32_t row , int32_t *T_IP1 )`
Initiator antenna	`CS_getTestInitiatorAntenna( uint32_t row , int32_t *ant )`
Reflector antenna	`CS_getTestReflectorAntenna( uint32_t row , int32_t *ant )`
Moreph Tx power	`CS_getTestMorephPwr( uint32_t row , double *pwr )`
DUT Tx power	`CS_getTestDUTpwr( uint32_t row , int32_t *pwr )`
DUT Tx SNR	`CS_getTestSNR( uint32_t row , int32_t *snr )`
Moreph clock ppm	`CS_getTestPPM( uint32_t row , int32_t *ppm )`
Quantity being tested	`CS_getTestQuantity(uint32_t row , char *quantity, uint32_t n )`
Limit value for test	`CS_getTestLimit( uint32_t row , double *limit )`
Measured value	`CS_getTestValue( uint32_t row , double *value )`
Units for quantity	`CS_getTestUnits( uint32_t row , char *units , uint32_t n )`
Upper or lower limit	`CS_getTestLimitType( uint32_t row , limit_t *limit )`
Test enum	`CS_getTestEnum( uint32_t row , test_t *test )`
Test pass or fail	`CS_getTestPass( uint32_t row , bool pass )`

Notes:

These functions return a `CS_error_t` which should be checked.

For functions which return a character string, the unsigned integer argument `n` is the maximum length of the character string which can be returned.

If the channel sounding procedure involved frequency hopping, then the test channel will be returned as -1.

The test description, the quantity being tested and the units in which it is measured are all returned as strings which may contain html embedded characters.

Where there is insufficient data to determine a test outcome, the measured value will be returned as NaN.

It the moreph antenna is returned as -1, it indicates that the moreph is simulating all antenna paths supported by the DUT. If the DUT antenna is returned as -1, it indicates that the all supported DUT antennas are connected to the moreph via a power splitter/combiner.

For each test there is an enum which can be retrieved which describes the quantity being measured. The possible enums are:

Category	Description	API enum	Explanation
MOREPH	Gain control	CS_rx_CS_attenuation	moreph receiver attenuation used whilst performing tests.
TRM	Stable phase	CS_phi_zmd_20	Percentage of ϕ_{zmd} which are below 20kHz.
		CS_phi_zmd_95	The value of ϕ_{zmd} at the 95% percentile.
	Modulation Characteristics 2M 2BT	CS_2BT_DF3	Value of $\Delta F3_{avg}$.
		CS_2BT_DF4_420	Percentage $\Delta F4_{avg}$ which are above 420kHz.
		CS_2BT_DF4_997	The value of $\Delta F4_{avg}$ at the 99.7% percentile.
		CS_2BT_ratio	The ratio of $\Delta F4_{avg}/\Delta F3_{avg}$.
	Tx output SNR control	CS_snr_3dB	Percentage of SNR_{Tx}^{Error} which are below 3dB.
		CS_snr_95	The value of SNR_{Tx}^{Error} at the 95% percentile.
		CS_snr_std	The standard deviation of SNR_{Tx}^{Error}.
TRM-RCV	Step Mode-0 frequency verification	CS_Mode0_FFOk	Value of FFO[k]
		CS_Mode0_FFOk_FFOo	Value of FFO[k] – FFO[0]
		CS_Mode0_Ftone_Fpkt_20	The percentage of $\lvert F_{tone} - f_{pkt}[k] \rvert$ which are below 20kHz.
		CS_Mode0_Ftone_Fpkt_95	The value of $\lvert F_{tone} - f_{pkt}[k] \rvert$ at the 95% percentile.
	Step main mode frequency verification	CS_Main_Fe_Fpkt_20	The percentage of $\lvert f_E - f_{pkt}[k] \rvert$ which are below 20kHz.
		CS_Main_Fe_Fone_10	The percentage of $\lvert f_E - f_{tone}[k] \rvert$ which are below 20kHz.
		CS_Main_Fe_Fpkt_95	The value of $\lvert f_E - f_{pkt}[k] \rvert$ at the 95% percentile.
		CS_Main_Fe_Ftone_95	The value of $\lvert f_E - f_{tone}[k] \rvert$ at the 95% percentile.
	Phase measurement accuracy	CS_phi_accuracy_ns	The percentage of $\lvert \alpha[p] \rvert$ which are below $1.7x2\pi$ ns
		CS_phi_accuracy_95	The value of $\lvert \alpha[p] \rvert$ at the 95% percentile.

Category	Description	API enum	Explanation
		CS_phi_accuracy_fit_rad	The percentage of $\alpha[p].fE[k] + \beta[p] - \Theta_{c,uw}[k,p]$ above 0.28rads
		CS_phi_accuracy_fit_95	The value of $\alpha[p].fE[k] + \beta[p] - \Theta_{c,uw}[k,p]$ at the 95% percentile.
	Phase measurement accuracy, inline PCT	CS_phi_accuracy_ns	The percentage of $\lvert\alpha[p]\rvert$ which are below 1.7x2π ns
		CS_phi_accuracy_95	The value of $\lvert\alpha[p]\rvert$ at the 95% percentile.
		CS_phi_accuracy_fit_rad	The percentage of $\alpha[p].fE[k] + \beta[p] - \Theta_{c,uw}[k,p]$ above 0.28rads
		CS_phi_accuracy_fit_95	The value of $\alpha[p].fE[k] + \beta[p] - \Theta_{c,uw}[k,p]$ at the 95% percentile.
		CS_q_zero	The number of PCT values returned by the DUT In which the Q value was zero.
NADM	Phase-based normalised attack detection metric	CS_NADM_correct	The percentage of reports with the correct NADM value.
		CS_false_pos	The percentage of reports with false positives. For information only.
		CS_false_neg	The percentage of reports with false negatives. For information only.
		CS_nadm_unknown	The percentage of reports with unknown status. For information only.
PAC	Sounding and random sequences	CS_pac_preamble	Number of packets with bit errors in the preamble.
		CS_pac_address	Number of packets with bit errors in the address.
		CS_pac_trailer	Number of packets with bit errors in the trailer.
		CS_pac_payload	Number of packets with bit errors in the payload.
	Access address quality indicator	CS_pac_miss_pkts	Number of missing packets with access address quality indicator not equal to 2.
		CS_pac_bit_flip	Number of packets with access address errors and access address quality indicator equal to 0.
		CS_pac_valid_pkts	Number of valid packets with access address quality indicator equal to 0.
	Sounding sequence with invalid marker	CS_pac_marker	Percentage of reports with correct packet quality and/or NADM.
	Channel selection algorithm #3b	CS_chB_mode0	Number of Mode-0 steps with CSsync on correct channel.
		CS_chB_main	Number of main mode steps CSsync/tone on correct channel.
		CS_chB_reports	Number of reports containing the correct channel.
		CS_chB_repeats	The number of repeats of a channel prior to all channels being used.

Category	Description	API enum	Explanation
	Channel selection algorithm #3c	CS_chC_mode0	Number of Mode-0 steps with CSsync on correct channel.
		CS_chC_main	Number of main mode steps CSsync/tone on correct channel.
		CS_chC_reports	Number of reports containing the correct channel.
RTT	Round trip timing	CS_sigma_plus_B	The value of 2σ+β.
		CS_RTT_sigma	The value of σ. For information only.
		CS_RTT_B	The value of B. For information only
TIM	CSsync packets timing verification	CS_Mode0_min_Trx_Texpected	The minimum value of TOD_{IUT} – $TOD_{IUT,expected}$ for all Mode-0 steps.
		CS_Mode0_max_Trx_Texpected	The maximum value of TOD_{IUT} – $TOD_{IUT,expected}$ for all Mode-0 steps.
		CS_Main_min_Trx_Texpected	The minimum value of TOD_{IUT} – $TOD_{IUT,expected}$ for all main mode steps.
		CS_Main_max_Trx_Texpected	The maximum value of TOD_{IUT} – $TOD_{IUT,expected}$ for all main mode steps.
	Ramp down profile	CS_ramp_40	The percentage of ramp downs which achieved 40dB within 5µs.
		CS_ramp_90	The value of the ramp down after 5µs at the 90% percentile.
PM	Antenna switching integrity	CS_integrity	The value of P_{on} – P_{off}.
	Antenna switching integrity, inline PCT	CS_integrity	The value of P_{on} – P_{off}.
		CS_q_zero	The number of PCT values returned by the DUT In which the Q value was zero.
	Phase measurement during PM	CS_all_measurements	The number of reports received from the DUT.
		CS_Mode3_tone_ext_correct	The number of Mode-3 IUT initiator absent tone extension slots with tone quality = 3 & PCT = 0
		CS_tone_quality_correct	Percentage of reports containing the correct tone quality for those slots were a transmission is expected.
		CS_tone_quality_7_4	Percentage of reports with bit [7:4] of the tone quality indicator set correctly.
		CS_refpwr_correct	Percentage of reports in which the reference power is between -127 and _20dBm.
		CS_low_correct	Percentage of reports with the low tone quality correct. For information only.
		CS_high_correct	Percentage of reports with the high tone quality correct. For information only.

Category	Description	API enum	Explanation
	Phase measurement during PM, inline PCT	CS_all_measurements	The number of reports received from the DUT.
		CS_Mode3_tone_ext_correct	The number of Mode-3 IUT initiator absent tone extension slots with tone quality = 3 & PCT = 0
		CS_tone_quality_correct	Percentage of reports containing the correct tone quality for those slots were a transmission is expected.
		CS_tone_quality_7_4	Percentage of reports with bit [7:4] of the tone quality indicator set correctly.
		CS_refpwr_correct	Percentage of reports in which the reference power is between -127 and _20dBm.
		CS_low_correct	Percentage of reports with the low tone quality correct. For information only.
		CS_high_correct	Percentage of reports with the high tone quality correct. For information only.
		CS_q_zero	The number of PCT values returned by the DUT In which the Q value was zero.
	Phase-based distance estimation, sounding sequence	CS_sounding_ns	Percentage of subevents with $\|\alpha\| < 10.2x2\pi$ ns.
		CS_sounding_95	The value of $\|\alpha\|$ at the 95% percentile.

11 AM attack mode

11.1 Overview

Access to the AM attack mode will require a Channel Sounding 6.1 license to be installed on the moreph unit.

An AM attacker is characterised by:

1. The pulse profile. The application supports rectangular, sine wave and custom profiles.
2. The duty cycle of the profile. The application shrinks the profile so that it completely fills the selected duty cycle.
3. The time offset of the profile. This is the gap between the start of a symbol and the start of the AM profile.
4. The gain of the profile. The application supports gains in the range 1x to 2x.

The AM attack mode permits a duty cycle / time offset ToF slice to be generated for a fixed gain. All other parameters for the channel sounding procedure are taken from left hand menus, including the pulse profile. If no pulse profile is defined then a rectangular pulse is assumed.

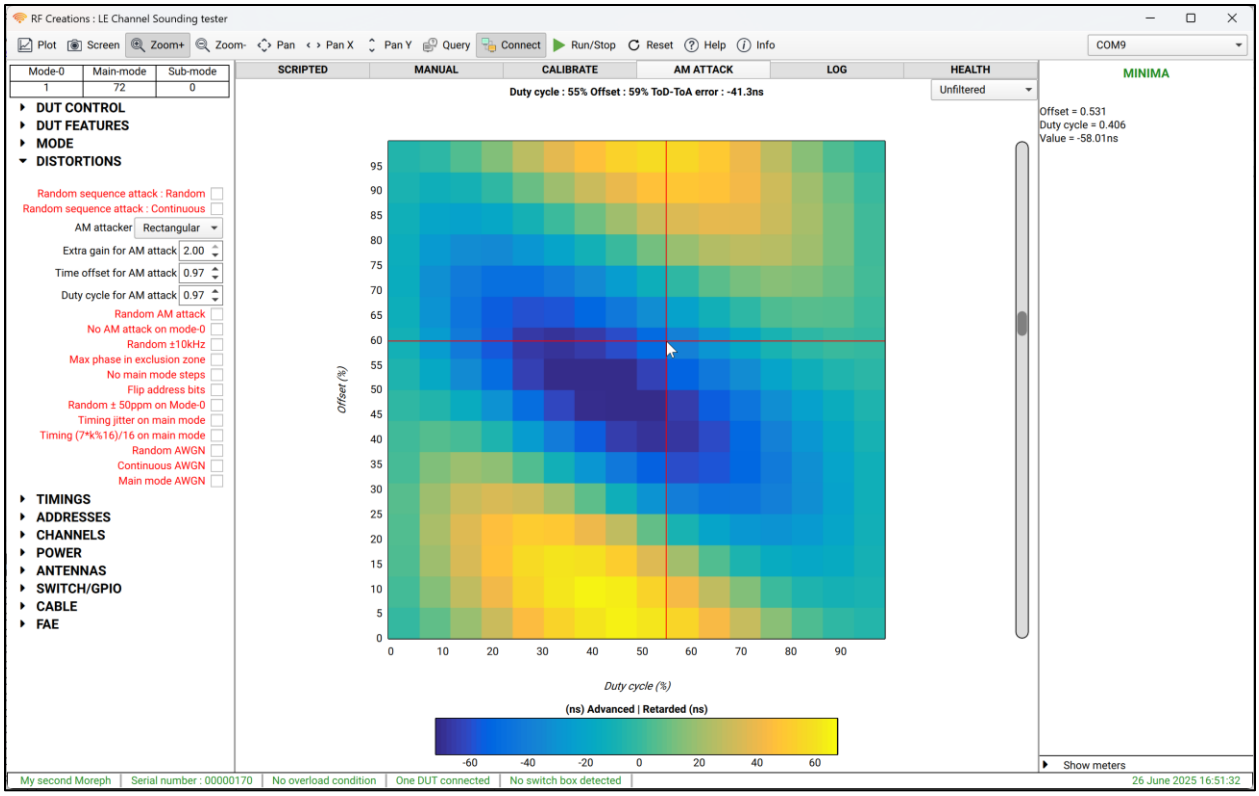
In this mode of operation all ToF calculations are performed in real-time. The rate at which channel sounding procedures can be performed is typically limited by the rate at which DUT HCI subevent reports can be sent over the HCI interface. It is therefore advantageous to run the HCI interface to the DUT at the highest supported baudrate.

11.2 Generating an AM attack slice

To generate an AM attack slice:

1. Connect to the DUT.
2. Either query the DUT supported CS features using the “Query” button on the toolbar or set them manually in the “DUT features” menu.
3. Switch to AM attack mode.
4. Set the channel sounding procedure parameters in the various menu.
5. Click the “Run” button on the toolbar.

The AM attack slice will then be generated. If a point in the slice appears erroneous, three attempts will be made at repeating the measurement before moving onto the next duty cycle / time offset setting.



The cross-hair cursors can be used to read off the variations in the ToD – ToA.

A drop down menu at the top right of the plot allows the user to display either filtered or unfiltered data.

The summary window contains a list of the minima which have been discovered on the filtered version of the slice.

The scrollbar to the right of the plot can be used to change the z-axis scaling.

11.3 AM attack API

The AM slice is a 16x16 array containing ToD – ToA error values. The values in the slice can be read back via the api:

```
double slice[16][16];

CS_getAMslice( slice );
```

11.4 Custom AM attack pulses

Custom AM attack pulses can be defined. These need to be described in csv files. Each row in the csv file should contain a {time,gain} pair. There should be no headers or footers in the csv file. The entries do not need to be time order.

The time coordinate is in terms of symbols and should be in the range 0.0 ≤ time ≤ 1.0.

The gain represents the additional gain to be applied at that time and should be in the range 1.0 ≤ gain ≤ 2.0.

The channel sounding application will perform linear interpolation between adjacent {time,gain} pairs.

An example file contain a simple triangle pulse would be:

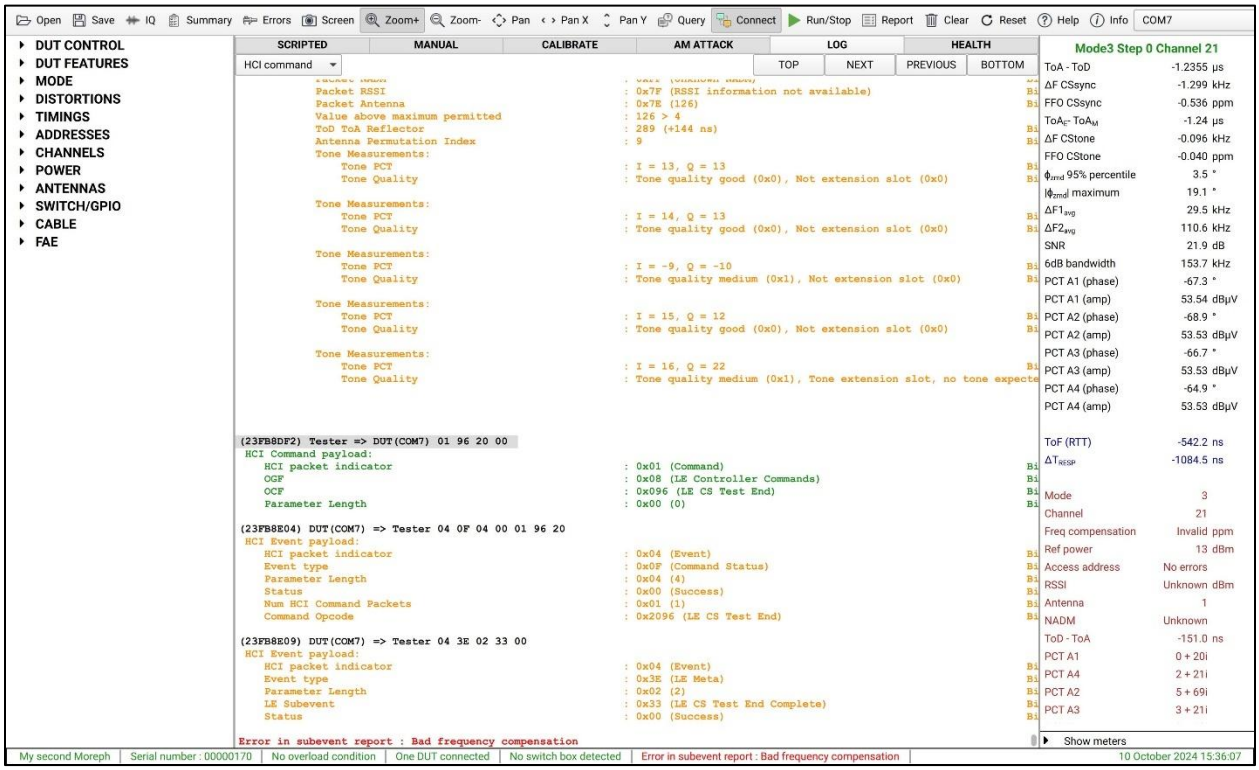
```
0.0,1.0
0.5,2.0
1.0,1.0
```

12 Log window

12.1 Overview

The log window is used to display:

1. Raw octets transferred over the HCI interface with the DUT.
2. Decoded HCI messages transferred between the moreph and the DUT.
3. Channels, addresses and modes calculated by the moreph for the channel sounding procedure.
4. Informational messages from the moreph.
5. Errors detected by the moreph.



12.2 Raw octets

The raw octets transferred over the HCI interface will only be include in the log window if “Log raw HCI octets” has been checked under the “DUT control” menu.

Raw octets are displayed in the log window in black.

At the start of each line in parentheses is a timestamp in units of milliseconds.

If the octets are being transferred from the moreph to the DUT, the line will commence:

```
Tester => DUT(comport)
```

If the octets are being transferred from the DUT to the moreph, the line will commence:

```
DUT(comport) => Tester
```

The remainder of the line contains the octets transferred in hex.

12.3 HCI decode

An HCI decode will always be entered into the log window. This will normally be displayed on multiple lines. To display on a single line check “Single line HCI decode” in the DUT control menu.

HCI messages from the moreph to the DUT are displayed in green. Those from the DUT to the moreph are displayed in orange.

12.4 Channels, addresses and modes

If “Log DRGB values” is checked in the “DUT control” menu, then for each channel sounding procedure the log window will contain for each step:

1. Channel
2. Mode
3. Initiator address
4. Reflector address

These values are printed in blue.

12.5 Informational messages

Informational messages are entered into the log window in black. Examples of informational messages are those generated when a DUT is connected or disconnected from the moreph.

12.6 Error messages

Error message are entered into the log window in red.

12.7 Test script messages

Test script messages are entered into the log window in blue. These messages are entered every time a new test script of phase of a test script is started. They contain:

a. The test name
b. The channels on which the test will be performed
c. The ppm timing offset which will be used
d. The antenna which will be used
e. The “round” for the test phase
f. The number of samples the test will require

The messages appear immediately before the first corresponding HCI test command. They therefore provide a convenient method of navigating through the log file to find the HCI traffic associated with a particular test.

12.8 Navigating within the log window

It is possible to navigate within the log window using the menu and push buttons at the top of the window.

The “Top” button will navigate to the start of the log. The “Bottom” button will navigate to the last entry in the log.

The “Previous” and “Next” buttons will navigate to the previous or next entity described by the drop down menu at the top left of the window. Possible entities to navigate to are:

1. HCI command
2. HCI test command
3. HCI event
4. HCI sub-event report
5. Scripted test
6. DRBG output
7. Error
8. Information

12.9 Exporting log window contents

The contents of the log window can be exported in html format by clicking the “Save” icon on the toolbar whilst the log window is visible. From the API call `CS_saveLog( filename)`.

The log window can be cleared by clicking the “Discard” button on the toolbar. From the API call `CS_clearLog()`.

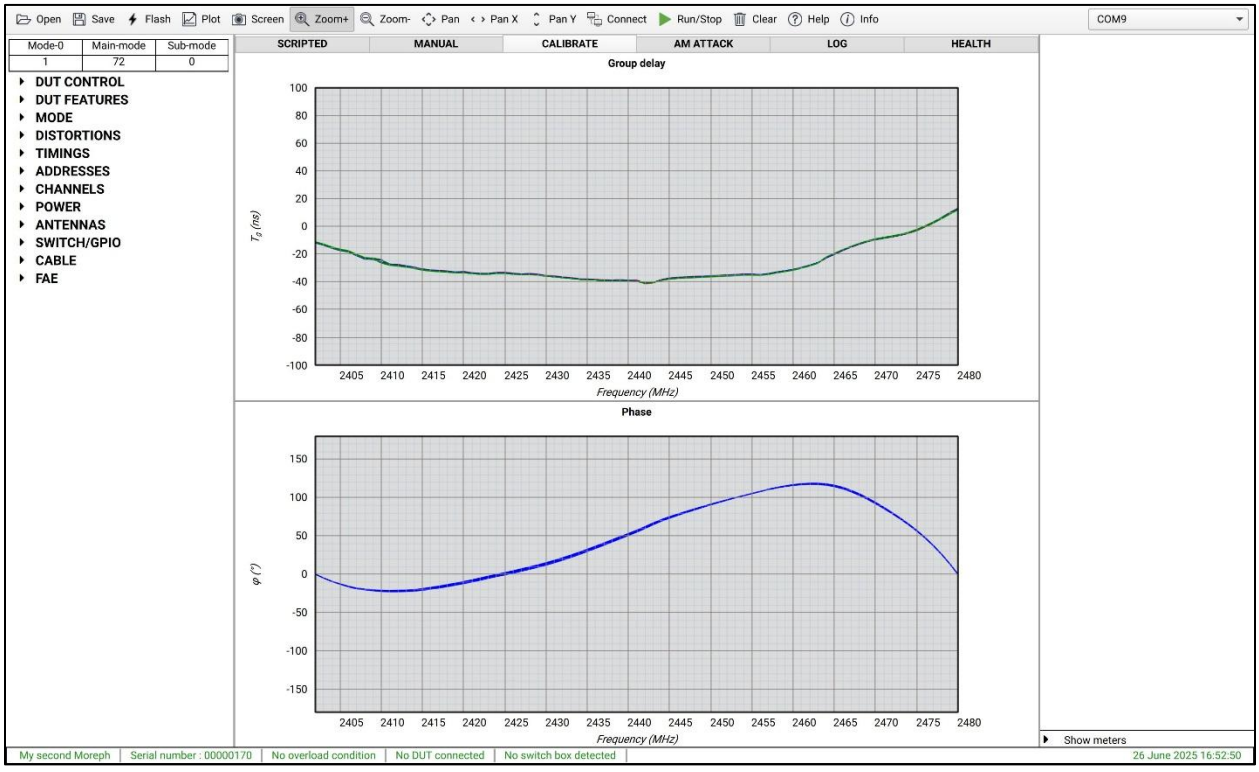
13 Calibration

13.1 Overview

The group delay and phase response of the moreph can be calibrated by the user using an RF open and RF short, as found in a network analyser calibration kit. Group delay and phase response calibration is required for the moreph to perform tests relating to time-of-flight.

By default the group delay and phase response of the moreph is calibrated to the SMA connector on the Tx/Rx port. The cable is then compensated for by entering the correct cable length and velocity factor into the “Cable” menu. However, in principal it is possible to calibrate to the end of the cable and this avoid having to enter details of the cable into the “Cable” menu.

The amplitude response of the transmit and receive chains cannot be calibrated by the user. For channel sounding the amplitude response of the receive chain is not critical. The amplitude response of the transmit chain does require calibration so that the correct signal level can be generated at the DUT input.



13.2 Calibration procedure

Calibration is not possible via the API.

Switch to calibration mode and click the “Run” button on the toolbar. The user will requested to connect an RF open to the Tx/Rx port. It is not sufficient just to leave the SMA port unconnected, a true RF open must be connected to the port. One the RF open is connected press “OK”.

The moreph will perform a number of measurements with the RF open attached. These measurements will take approximately 4minutes. Once the measurements have been completed, the user will prompted to replace the RF open with an RF short. When the user has completed this operation click “OK” to start the second set of measurements. The second set of measurements will take approximately 4 minutes to complete.

Once the second set of measurements has been completed, the moreph will be have a new set of calibration data which will be used for future measurements. However, this calibration data is volatile and will be lost if the application closes or the moreph shuts down.

13.3 Save and restoring calibration data

The new calibration data can be saved for a file by clicking the “Save” icon on the toolbar.

Previously saved calibration data can be recalled by clicking the “Open” icon on the toolbar.

The current calibration data can be saved to non-volatile memory in the moreph. This will then become the default calibration data which is loaded every time the channel sounding application is launched. To save the current calibration data to the moreph non-volatile memory, click the “Flash” icon on the toolbar.

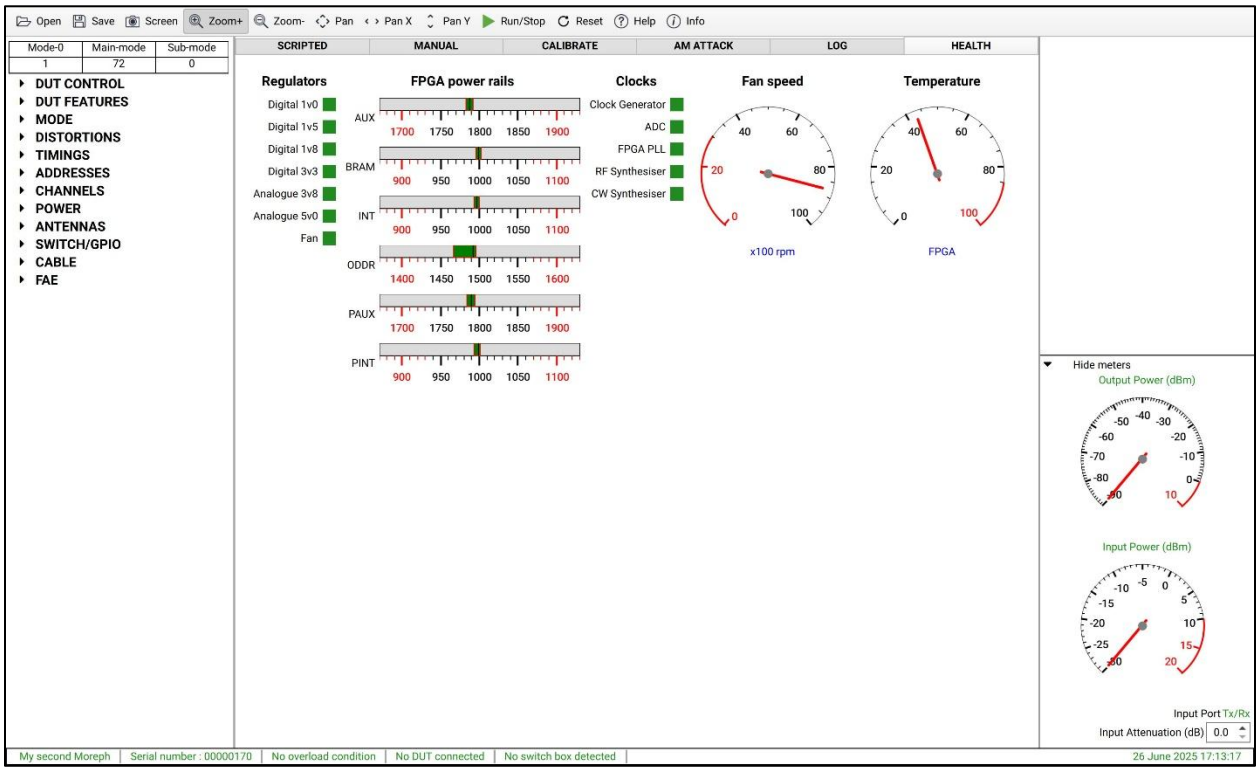
To save a calibration file from the API call `CS_saveCalibration( filename )`. To load a calibration file from the API call `CS_loadCalibration( filename )`.

14 Health

The health tab displays information on:

1. Power rails
2. Clocks
3. FPGA die temperature
4. Fan speed

The moreph will automatically shut down if any of these parameters remained outside their specified operating regions for a prolonged period.



15 C API

15.1 Overview

The C API provides access to most features of the UI. It currently does not support the generation of test scripts, but it can load and run test scripts created in the UI.

The API is reliant on an API server. It is the user's responsibility to start and terminate the API server appropriately. This differs from earlier versions of the API where the API server was launched and terminated automatically.

The C API permits the UI to be either visible or hidden.

The header file CSapi.h must be included in any user code. This file also includes CSresults.h. The executable must have access to the dynamic libraries contained in the install directory. Note that Ruby.rfcplugin is a dynamic library.

An example program named CS_example.cpp is available in the install directory.

15.2 API server

The API is reliant on the API server running. The API server must be started prior to any API calls.

The API server executable is named CSapi_server.exe and resides in the install directory, which by default is C:/Program Files/RFcreations/Phy.

The C example code does not include statements for launching the API server since these may be OS dependent. It is therefore essential that the user either adds code to launch the API server or starts it manually before running the example code.

The API server will be terminated by any call to `CS_disconnect()`. If the user wishes to make further calls to the API after calling `CS_disconnect()`, then it is essential that they restart the API server first.

15.3 Error handling

Most API calls return a `CS_error_t` which are listed in api.h.

If the call returns `CS_ERROR_UNKNOWN` then further information on the error may be contained in the character string returned by `CS_error_msg()`.

15.4 Connecting to a moreph

The moreph devices connected to the host computer can be found using `CS_search()`. These devices can then be scanned to find the one of interest, for example, by looking for a specific serial number. The selected device can then be connected to using `CS_connect()`.

```
// Connect to Moreph
// -----------------

int serial_number = 0x150;
bool connected = false;
CS_search_result_t* results;
std::cout <<  "Searching for Moreph..." << std::endl;
while (!connected) {
    int N;
    CS_search(&results, &N);
    for (int i = 0; i < N; i++) {
        std::cout << "Found Moreph " << std::hex << std::uppercase <<
                        results[i].serial_number << std::dec << std::endl;
        if(results[i].serial_number == serial_number) {
            std::cout << "Connecting to Moreph " << std::hex << std::uppercase <<
                        serial_number << std::dec << std::endl;
            CS_connect(results[i].handle);
            connected = true;
            break;
        }
    }
    std::this_thread::sleep_for(std::chrono::milliseconds(500));
}
```

Alternatively, it is possible to connect directly to a moreph with a given serial number:

```
CS_connect_serial(uint32_t serial_number);
```

15.5 Making the UI visible

It is possible to make the UI visible whilst using the API:

```
// Make UI visible
// ---------------

CS_show_gui();
std::cout << "UI launched" << std::endl;
```

15.6 Connecting to a DUT

Prior to connecting to the DUT it is necessary to configure the UART. Once this has been done a `CS_connectDUT()` call can be made to connect to the DUT.

```
// Connect to DUT
// --------------

CS_setBaud( 115200 );
CS_setHWflow( CS_HW_RTS_CTS );
CS_setSWflow( CS_SW_none );
CS_setStopBits( 1 );
CS_setParity(CS_PARITY_none);
std::cout << "UART configured" << std::endl;

CS_error_t err;
err = CS_connectDUT("com10");
if (err != CS_NO_ERROR) {
    std::cout << "Couldn't connect to DUT" << std::endl;
    return 1;
}
std::cout << "Connected to DUT" << std::endl;
```

Having successfully connected to the DUT it is necessary to inform the application of the features which the DUT supports. This can either be done by individual commands (see section Setting supported features) or by enquiring from the DUT over HCI which features it supports:

```
// Find DUT supported features
// ---------------------------

err = CS_queryDUT();
if (err != CS_NO_ERROR) {
    std::cout << "Couldn't query DUT supported features " << uint32_t(err) <<
                 CS_error_msg() << std::endl;
    return 1;
}
std::cout << "Queried DUT supported features" << std::endl;
```

15.7 Setting the operating mode

To perform a single CS procedure the operating mode should be set to manual:

```
// Set operating mode and reset
// ----------------------------

CS_setOperatingMode(CS_manual);
```

Prior to setting up the channel sounding procedure the log window can be cleared and all channel sounding procedure parameters reset to their default values:

```
CS_clearLog();
CS_resetCS();
std::cout << "Switched to manual mode" << std::endl;
```

15.8 Set cable parameters

The channel sounding application needs to be able to compensate for the group delay between its Tx/Rx port and the DUT antenna connection. To do this, the length of the cable between the moreph and the DUT must be specified along with its velocity factor.

The scripted tests defined a power level at the DUT antenna connection. For the moreph to generate the correct power level at the DUT antenna connection it is necessary for the channel sounding application to know the loss between the moreph Tx/Rx port and the DUT antenna connection.

```
// Set cable parameters
// --------------------

CS_setCableLength( 30.0 );
CS_setCableVelocity( 67.0 );
CS_setExtAttenuation( 2.7 );
std::cout << "Cable parameters configured" << std::endl;
```

15.9 Setting up a channel sounding procedure

See section Setting up a channel sounding procedure for API calls to configure the channel sounding procedure.

```
// Define CS procedure
// -------------------

CS_setRole(CS_Initiator);
CS_setMode0Steps(3);
CS_setMainMode(CS_Mode3);
CS_setRTT(CS_Random32);
CS_setChanMapCustom(CS_mapDUeven);
CS_setChanMapLength(72);
CS_setChanMapRepeats(2);
CS_setNumSteps(90 + 3);
std::cout << "Manual CS procedure configured" << std::endl;
```

15.10 Running a channel sounding procedure

A running channel sounding procedure can be aborted by calling `CS_abortProcedure()`. If the channel sounding procedure times out whilst waiting for completion, then it should be aborted before attempting to run a new channel sounding procedure.

```
// Run CS procedure
// ----------------

CS_runProcedure();

if( CS_waitProcedure( 10000 ) == CS_ERROR_TIMEOUT ) {
    std::cout << "Manual CS procedure timed out" << std::endl;
    return 1;
}

std::cout << "Ran manual procedure" << std::endl;
```

15.11 Reading results of a channel sounding procedure

Section Summary window provides a full list of the quantities which can be read back after executing a channel sounding procedure.

```
// Examining the results
// ---------------------

double ToF_average;
CS_readResult( CS_avg_ToF , -1 , CS_Both , &ToF_average );
double ToF_std;
CS_readResult( CS_std_ToF , -1 , CS_Both , &ToF_std );
std::cout << "Average RTT ToF is " << ToF_average << " with standard deviation of "
          << ToF_std << std::endl;

double alpha;
CS_readResult( CS_alpha_1_1 , -1 , CS_Both , &alpha );
              std::cout << "Value of alpha from linear phase regression on antenna
                 path 1:1 is " << alpha << std::endl;
```

15.12 Loading a test script

To load the test script it is necessary to switch to the scripted mode of operating. Test scripts can be generated using the UI. These can then be loaded into the channel sounding application using the API.

```
// Loading a test script
// ---------------------

CS_setOperatingMode(CS_scripted);
std::cout << "Switched to scripted mode" << std::endl;

if( CS_loadTestScript( "My_test_script.tsf") != CS_NO_ERROR ) {
    std::cout << "Failed to load test script" << std::endl;
    return 1;
}
std::cout << "Test script loaded" << std::endl;
```

15.13 Creating a test script

The current test script can be cleared by calling `CS_clearTestScript()`.

New tests can be appended to the current test script by calling `CS_clearTestScript()`. This function takes the following arguments:

const char *	testName	Name of the test to add to the test script. Must be match one of the test names available in the UI. For example, “RFPHY/TRM/CS/BV-01-C”.
const char *	channels	The channels on which the test must be performed. Only applicable to those tests which are run on a single channel, ie primarily the RF-PHY tests. The channels are represented as common separated channel numbers with no spaces, eg "2,38,76”. Specifying an empty string, ie ‘”, will lead to the default channels for the test being used.
const char *	ppms	The moreph clock offsets in ppm at which the test must be performed. Some tests can be performed at more than one clock offset in which case a comma separated list of ppm values can be specified with no spaces, eg “-50,0,50”. Specifying an empty string, ie ‘”, will lead to the default clock offsets for the test being used.
const char *	morephPwr	The power at which DUT antenna provided by the moreph in dBm. For examples “-70.0”. Specifying an empty string, ie ‘”, will lead to the default moreph transmit power for the test being used.
const char *	morephSNR	The signal to noise ratio of the moreph transmission at the DUT antenna in dB. For examples, “25”. It is also possible to specify maximum SNR as “max”. Specifying an empty string, ie ‘”, will lead to the default moreph transmission SNR for the test being used.
const char *	DUTpwr	The DUT transmit power in dBm. Can be specified as “max”, “min” or a custom value, eg “3”. Specifying an empty string, ie ‘”, will lead to the default DUT transmit power for the test being used.
uint32_t	nSamples	The number of samples used in the test. Specifying a value of 0 will result in the default number of samples being used for the test.

```
// Clear the test script and add a new test
// ----------------------------------------

CS_setOperatingMode(CS_scripted);
std::cout << "Switched to scripted mode" << std::endl;

CS_clearTestScript();
CS_addTest("RFPHY/TRM/CS/BV-01-C", "", "", "", "", "", 0);
```

15.14 Running a test script

The test script is run in exactly the same manner as a manual channel sounding procedure. Again, if the test script times out whilst waiting for completion, it should be aborted prior to attempting to run a new test script.

```
// Run the test script
// -------------------

CS_runProcedure();
err = CS_waitProcedure( 2 * 60 * 1000 );
if( err == CS_ERROR_TIMEOUT ) {
    std::cout << "Test script timed out" << std::endl;
    return 1;
} else if( err != CS_NO_ERROR ) {
    std::cout << "Error waiting for test script" << std::endl;
    return 1;
}
std::cout << "Ran test script" << std::endl;
```

15.15 Reading the results of a test script

The results of running a test script are contained in a single table. This table is the concatenation of all the tables shown in the UI.

It is possible to use the test name to locate the row in the table at which that test commenced. The example code then loops through each of the subsequent rows until and start of a new test is encountered. For each row the code reads back the test enum, the value and the role of the DUT. The results for 2σ + β are written out in this example.

A full list of test quantities which can be read and their enums are described in section Examining results .

```
// Read the results of the test script
// ---------------------------------

int32_t row;
CS_findTest( "CS/RTT/INI/BV-01-C" , 0 , &row );
if( row < 0 ) {
    std::cout << "Test results not found" << std::endl;
    return 1;
}

while( 1 ) {
    char name[64];
    if( CS_getTestName( row , name , 64 ) != CS_NO_ERROR ) break;
    if( strcmp( name , "CS/RTT/INI/BV-01-C" ) ) break;

    double val;
    CS_getTestValue( row , &val );

    role_t role;
    CS_getTestRole( row , &role );

    int32_t ppm;
    CS_getTestPPM( row , &ppm );

    test_t test;
    if( CS_getTestEnum( row , &test ) == CS_NO_ERROR ) {

        switch (test) {
        case CS_sigma_plus_B: {
            std::cout << "DUT role " << ((role == CS_Initiator) ? "Initiator" :
                "Reflector") << " PPM = " << ppm << " 2xSigma + Beta = " << val <<
                std::endl;
            break;
        }
        default: {
            break;
        }
        }
    }
    row++;
}
```

The results of the test script can also be saved to an html file:

```
// Print the test script results
// -----------------------------

err = CS_saveResults( "C:/Users/TimNewton/Documents/TestResults.html" );
if( err != CS_NO_ERROR ) {
    std::cout << "Failed to write test results to file" << std::endl;
    if (err == CS_ERROR_UNKNOWN)
        std::cout << CS_error_msg() << std::endl;
} else {
    std::cout << "Test results written to file" << std::endl;
}
```

15.16 Disconnecting

To disconnect the DUT:

```
// Disconnect DUT
// --------------

CS_disconnectDUT();
std::cout << "DUT disconnected" << std::endl;
```

To closing the UI:

```
// Close the GUI
// -------------

CS_hide_gui();
std::cout << "GUI hidden" << std::endl;
```

To disconnect the moreph and terminate the API server:

```
// Disconnecting the moreph and terminating the API server
// ------------------------------------------------------

CS_disconnect();
std::cout << "Disconnecting" << std::endl;
```

If the program terminates without hiding the UI and not disconnecting from the API server, then the UI will remain visible and active. This provides a mechanism of running a problematic test via a script and then switching to an interactive mode to examine the data.

15.17 Resetting moreph hardware

The moreph hardware can be reset by first performing a `CS_search()`, selecting which device to reset from the search results and then executing a `CS_hardwareReset(idx)`, using the index of the selected device.

16 Python API

16.1 Overview

Function calls in the Python API are identical to those for the C API with the following exceptions:

1. Enums are replaced by character strings. For example, the C enum CS_Initiator simply becomes the string “CS_Initiator”.
2. The calls for reading back test script results return the result directly, for example:

   ```
   CS_error_t CS_getTestPass( int32_t row , bool *testPass )
   ```

   becomes:

   ```
   testPass = CS_getTestPass(row: int)
   ```

The python library is held in CSapi.py in the install directory.

An example Python program named CS_example.py is provided in the installation directory.

16.2 API server

The API server can be started from a Python script as follows:

```
import subprocess

# Start the API server
# --------------------

subprocess.Popen(["C:/Program Files/RFcreations/Phy/CSapi_server"])
```

The API server will be closed by any call to `CS_disconnect()`, and must be restarted before any further API calls can be made.

16.3 Reseting moreph hardware

An example Python program for resetting the moreph hardware can be found in CS_example_reset.py.

17 Standalone mode

The channel sounding application can be run without a moreph being attached. In this standalone mode the UI can be used to view capture files which were saved from manual mode. The capture files contain the channel sounding procedure configuration, the collected IQ data, the sub-event reports from the DUT and the calibration data used for the capture. The capture files do not contain the HCI traffic. This can must be saved separately either as a BTsnoop log or as an html dump of the log window.

Standalone mode can be entered using the following Python script:

```
import subprocess

import sys

sys.path.append("C:/Program Files/RFcreations/Phy")

from CSapi import *

subprocess.Popen(["C:/Program Files/RFcreations/Phy/CSapi_server"])

CS_standalone()

CS_show_gui()
```

This script is contained in the installation folder with the name CS_standalone.py.

18 Supported graphics formats

The following graphics formats are supported for saving plots or screen captures from the UI:

1. Windows bitmap files (*.bmp)
2. Joint photographic expert group files (*.jpg)
3. Portable network graphics files (*.png)
4. Portable bitmap files (*.pbm)
5. Portable graymap files (*.pgm)
6. Portable pixmap files (*.ppm)
7. X11 bitmap files (*.xbm)
8. X11 pixmap files (*.xpm)