

Kinetis Thread Stack Certification

Contents

1. About this document

Thread v1.1 Certification ensures via rigorous testing that commercial products can connect effortlessly and securely to a Thread network.

Products need to execute a set of test cases to demonstrate correct device behavior within a Thread network. The Thread certification process covers various Thread areas such as commissioning, functionality, interoperability, and specification conformance.

Thread v1.1 specification is available to non-members through www.threadgroup.org/ourresources

The latest Thread Certification Test Plan can be downloaded from the Thread Group member's website.

For membership information visit www.threadgroup.org/joinus

Thread members have the option to completely replicate the entire Authorized Test Lab (ATL) equipment configuration to ensure the highest confidence in passing formal Thread certification.

For Thread certification, automation is not possible to run tests consecutively without human intervention. Tests can be queued up, however a person will need to react to requests from the Harness for information such as the Factory MAC address, the ML64 address, and so on.

This document shows how to run Thread Certification test cases against a NXP device running the Kinetis Thread software which acts as a Device Under Test (DUT).

1.	About this document.....	1
2.	Introduction	2
3.	Thread certification process.....	2
4.	GRL test harness architecture	3
5.	Pre-validation setup	4
6.	DUT software configuration.....	6
7.	Thread test plan	10
8.	Running thread certification cases	10
9.	DUT Commands.....	38
10.	Filling the Thread Certification application document	60
11.	Revision history.....	60

2. Introduction

Thread is an IPv6-based mesh networking protocol developed by industry leading technology companies, such as NXP, for connecting products around the home and in buildings to each other, to the internet, and to the cloud. Consumers will buy Thread products from various vendors, and expect all those products to work together easily and quickly. Thread product certification ensures interoperability out of the box.

All Thread products must undergo Thread Product Certification in order to use the term “Thread” in association with the product and/or in order to display the appropriate Thread logo.

Thread certification involves testing a product against a defined list of test cases to measure its compliance with the Thread specification while inside a mixed network of certified stacks to measure its interoperability with other Thread products.

The certification process consists of product testing and verification of the implementation of Thread technology. Thread certification products can be broken down into three categories: component, module, and end-product.

A component contains a Thread stack implementation, for instance the NXP KW41Z or KW2xD Thread Software Stack. End-products are component-based solutions with Thread support. A Thread module is a self-contained RF device with the intention to be integrated into an end-product. All products need to undergo the same testing.

Commercial end-products must be certified to receive the Thread logo. The scope of this certification only covers compliance with the Thread specifications.

3. Thread certification process

The following figure shows a high-level diagram of the Thread Certification Process

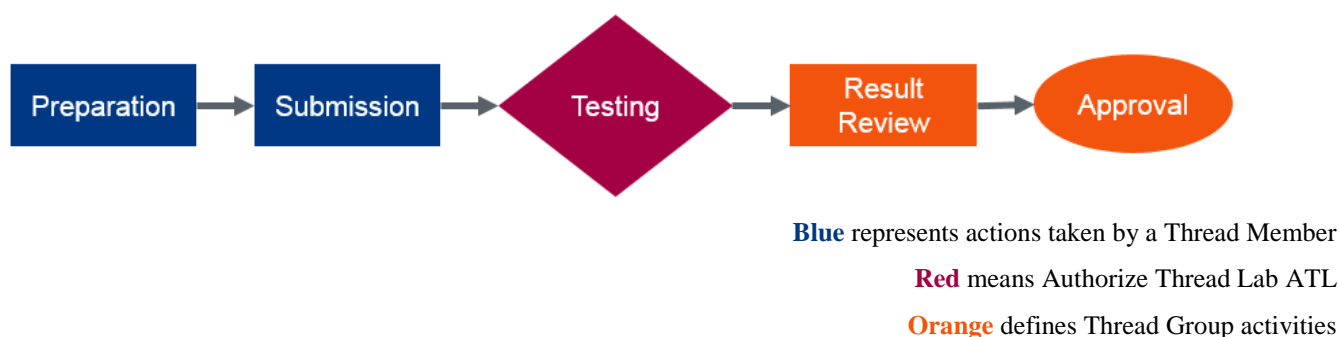


Figure 1. Thread Certification Process

Preparation: A Thread member selects an ATL to do the certification work and compiles the submission package. It is recommended to perform pre-testing to ensure the product will pass the official testing process.

Submission: Thread member then submits a product certification application package including: product information, Thread Certification Application form, Thread Conformance Document checklist, proof of successful completion of all other related technologies such as 802.15.4, WiFi, BLE, and so on.

Testing: Upon arrival of the product sample and accessories at the ATL, testing shall be scheduled and conducted in accordance with Thread certification policies and procedures.

Result Review: Test result are reviewed by the Thread Certification Authority CA.

Approval: Thread CA issues a certificate and post the information on the Thread Group website.

The following figure shows the Thread Certification flow that includes pre-validation or pre-testing in light blue.

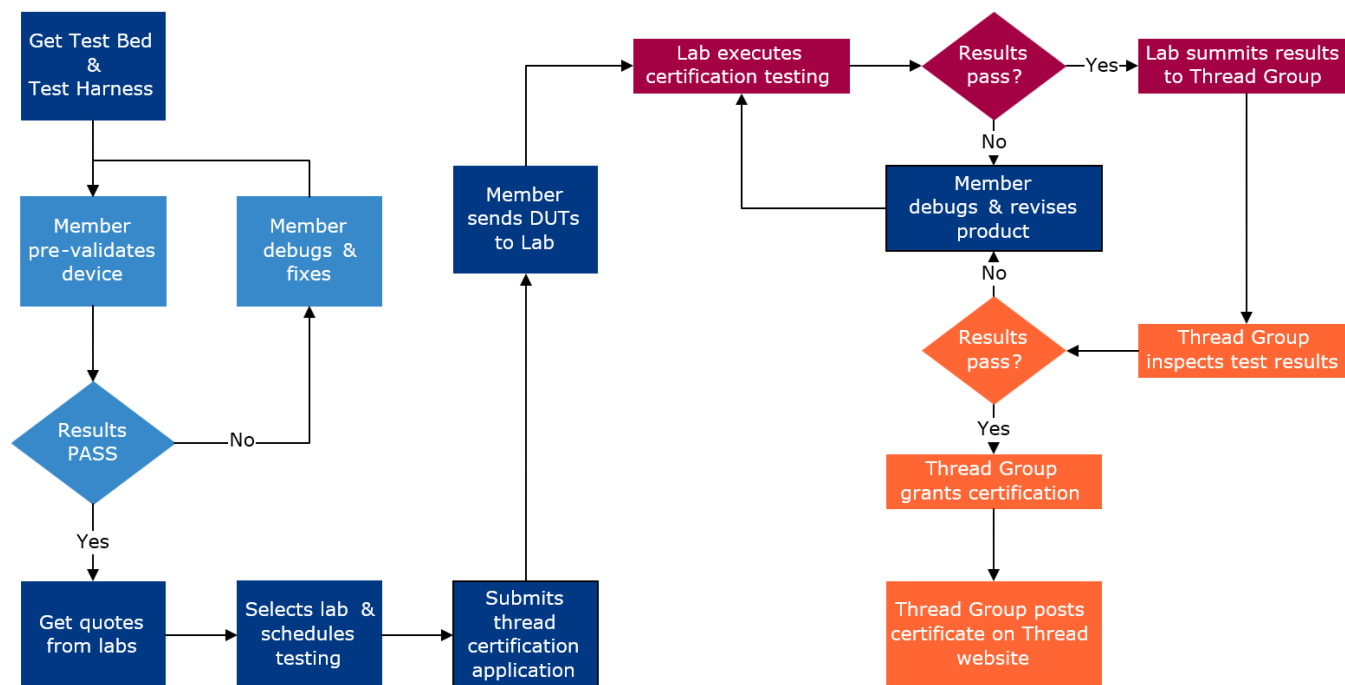


Figure 2. Thread Certification Flow

4. Thread test harness architecture

A Thread Test Bed consists of

1. Host computer installed with Thread Test Harness Software
2. Up to two NXP 802.15.4 packet sniffers
3. Thread Test Bed golden reference devices
4. USB hub to connect Thread Test Bed devices

The Thread Test Harness semi-automates the test cases defined in the Thread Test Plan. The harness sets up the required topology using the Thread Test Bed reference devices, and runs the test steps within the Test Plan.

One or two 802.15.4 packet sniffers are used for capturing over the air traffic. A second sniffer may be required to validate test cases using two-channel network topologies. Capture packets are decoded using Wireshark. The functionality of the DUT is validated by verifying that the packets meet the pass criteria in the test plan.

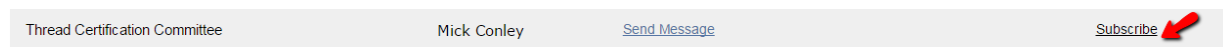
5. Pre-validation setup

The majority of Thread Certification information can only be shared with Thread members. After joining the Thread Group follow these steps to be able to download certification documents from the members-only website.

1. Go to the Thread members website <https://portal.threadgroup.org/>
2. Join the Thread Certification Committee
 - a) Search for Thread Certification Committee



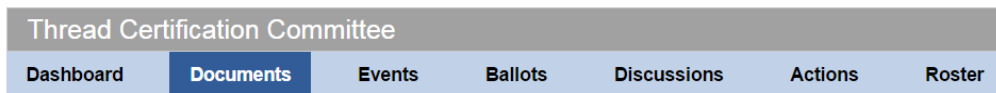
- b) In the Actions column click on Subscribe



3. Go to the Thread Certification Committee

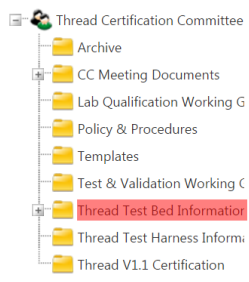
- c) Select Thread Certification Committee in **Go to a Work Group**


 - d) Click on Documents



5.1. Ordering a Thread Test Bed

1. Download the latest Thread Test Bed catalog from the Thread Certification Committee – Thread Test Bed Information folder.



2. Select between base or large node network topologies. A base topology uses up to 6 test bed golden devices plus sniffers and covers ~85% of the Thread Test Plan. The other 15% test cases need a large node topology with up to 33 test bed golden devices.
3. Follow the ordering instructions included in the Test Bed Catalog

5.2. Installing Thread Test Harness

After receiving the Thread Test Bed hardware, install the pre-validation Thread Test Harness software downloadable from <http://graniteriverlabs.com/thread/> and follow the quick start guide at the download tab.

Thread Test Harness Software

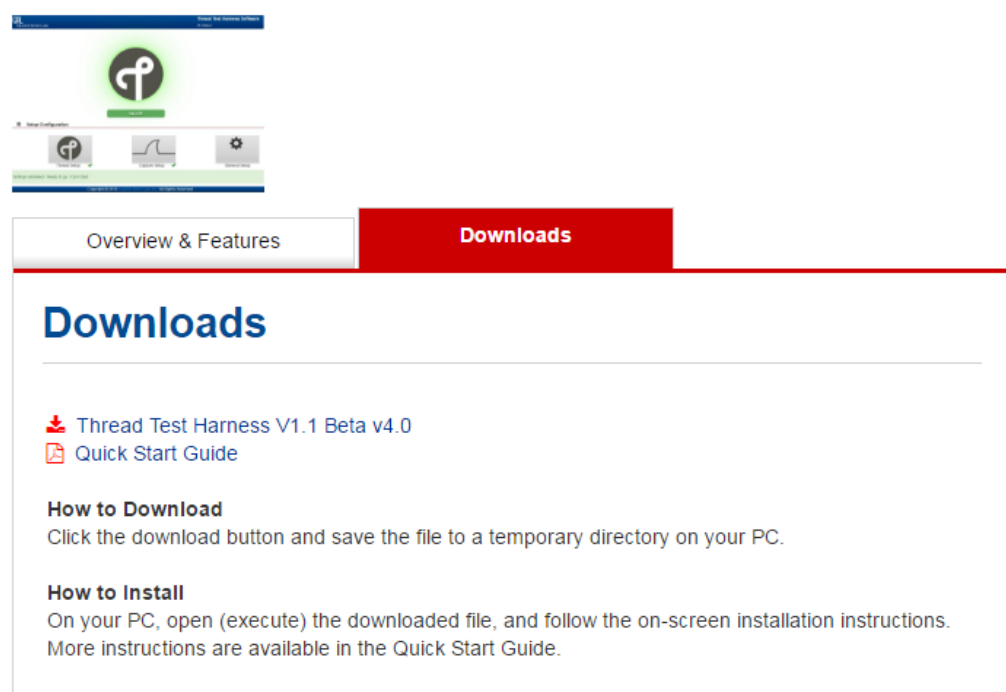


Figure 3. GRL Thread Test Harness

Do not forget to use your company email address when filling in the download form.

 The screenshot shows a modal form titled 'GRL - Thread Test Harness V1.1 Beta v4.0' overlaid on the website. The form contains several input fields: 'Email (required)', 'First name (required)', 'Last name (required)', 'Title', and 'Company (required)'. At the bottom of the form is a 'Download' button. The background of the website is visible but dimmed.

5.3. Device Under Test (DUT)

A DUT is also known as the product being tested. It can be a software-hardware combination that will be submitted for certification testing. This document uses a FRDM-KW24D512 programmed with a Kinetis Thread software example software as DUT.

The Thread example project depends on the Thread device functionality submitted for certification. The following table shows the examples projects used in this document.

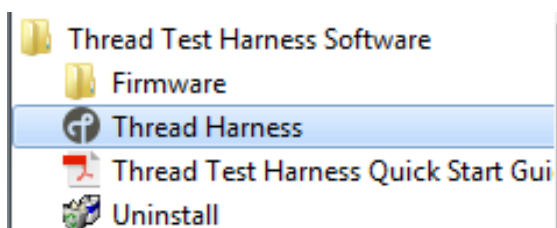
Table 1. Examples projects used in this document

Thread Device Function	Software Example Project	
	Shell DUT	THCI DUT
SED only	low_power_end_device	host_controlled_device
MED only	end_device	
Router only	router_eligible_device	
Router + Commissioner	router_eligible_device	
Router + Commissioner + Border Router	border_router	

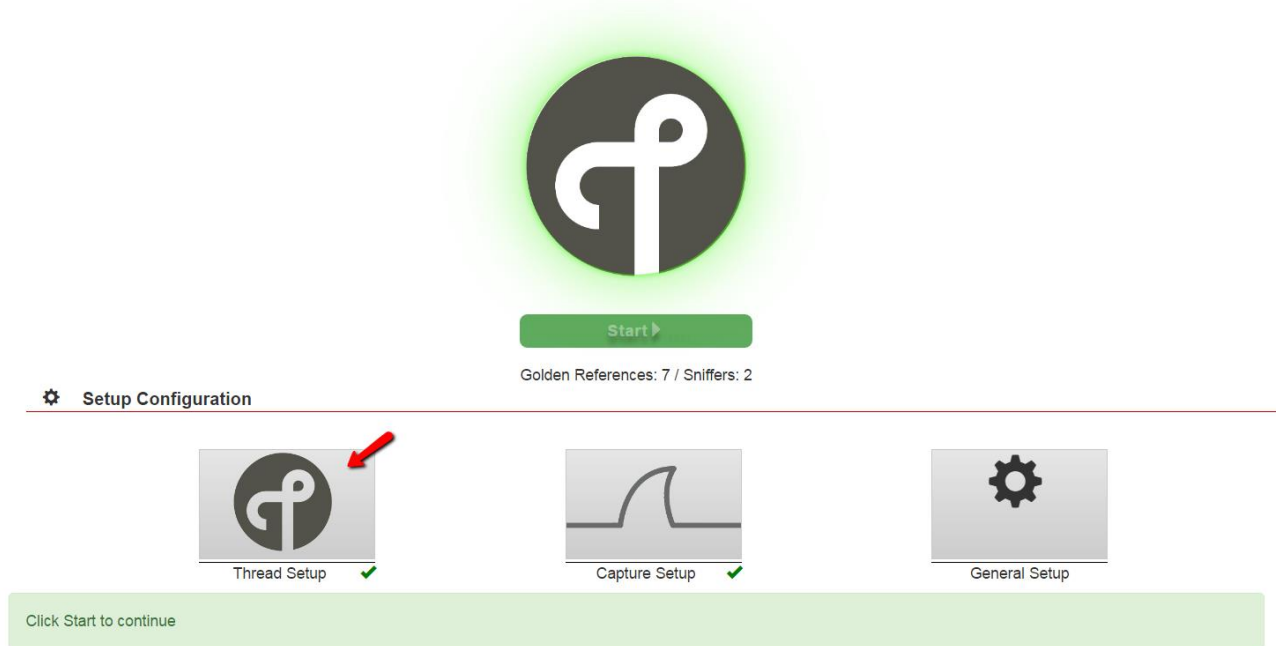
6. DUT software configuration

To understand the Thread Test Harness setup follow these steps.

1. Open the Thread Harness application by double clicking on Start → Menu → All Programs → Thread Test Harness Software → Thread Harness.



2. Click the **Thread Setup** logo.

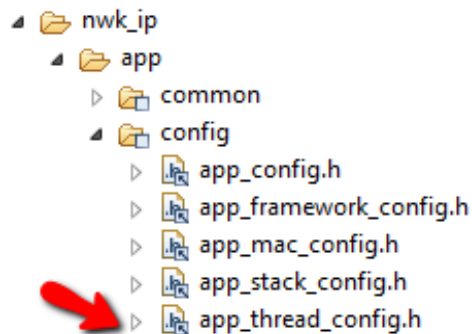


3. A new window will pop-up with the Mandatory Thread Settings required for a DUT
4. Follow section 6.1.1
5. Click on Save Settings
6. Click on General Setup
7. Configure the General Settings instructions as per section 6.1.2
8. Click on Save Settings.

6.1. Kinetis Thread Software Configuration

Kinetis Thread software architecture permits pre-compile settings such as: Network Name, xPANID, PSKc, PSKd, Network Key, and so on. Follow these steps to configure application parameters prior to programming the DUT:

1. Open the DUT Kinetis Thread Certified Software project inside an IDE
2. Inside the IDE, navigate to the `nwk_ip->app->config` folder
3. Double click `app_thread_config.h`



6.1.1. Thread Setup

Main channel: Look for the THR_SCANCHANNEL_MASK value to configure the RF Channel for the DUT and ensure that the harness Thread setup value is the using the same channel number.

```

#ifndef THR_SCANCHANNEL_MASK
#define THR_SCANCHANNEL_MASK (0x00000001 << 12)
#endif

```

Network Settings

Main Channel*

12

Second Channel

13

Network Key: There is no need to modify this macro defining the THR_MASTER_KEY as the same network key is being used by the Test Harness.

```

#ifndef THR_MASTER_KEY
#define THR_MASTER_KEY {0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, \
0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff}
#endif

```

Network Name: Change the THR_NETWORK_NAME macro value to {3,"GRL"}.

```

/*! The default network name */
#ifndef THR_NETWORK_NAME
#define THR_NETWORK_NAME {3, "GRL"}
#endif

```

{3, "GRL"}

Network Name*

GRL

PAN ID: Replace the current THR_PAN_ID definition from THR_ALL_FFfs (random PAN ID) to 0xFACE.

```

/*! The PAN identifier.
If this value is 0xFFFF a random PAN ID will be generated on network creation */
#ifndef THR_PAN_ID
#define THR_PAN_ID 0xFACE
#endif

```

0xFACE

PAN ID*

FACE

Extended PAN ID: Change the THR_EXTENDED_PAN_ID macro from 0xFF,..... to the value shown at the Thread Settings.

```

/*! The extended Pan ID.
If this is set to all FFfs, a random extended PAN ID will be generated on network creation */
#ifndef THR_EXTENDED_PAN_ID
#define THR_EXTENDED_PAN_ID {0x00, 0x00, 0xB8, 0x00, 0x00, 0x00, 0x00, 0x00}
#endif

```

Extended PAN ID*

000DB80000000000

MLE Prefix: The software application generates a random MLE prefix, therefore this value will not be able to be configured at compile time. This document will describe how to configure at run time, and what test cases require this setting.

PSKc: When THR_PSK_C is made up of all zeros the application software computes the PSKc using the PSKd as the commissioning credential. The PSKd value is mirrored in the PSKc.

```

#ifndef THR_PSK_D
#define THR_PSK_D {15, "threadjaketest"}
#endif

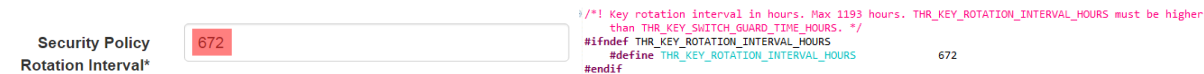
/*! If the THR_PSK_C have all bytes zero, the PSKc is computed using the PSKd as
commissioning credential for PBKDF2() (see "Derivation of PSKc" section) */
#ifndef THR_PSK_C
#define THR_PSK_C {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
#endif

```

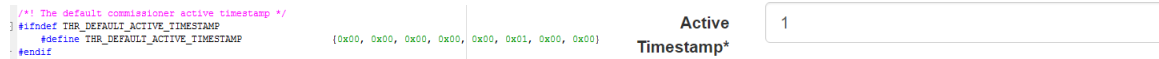
PSKc*

threadjaketest

Security Policy Rotation Interval: Modify the Thread Harness value to match the software macro THR_KEY_ROTATION_INTERVAL_HOURS.



Active Timestamp: At the THR_DEFAULT_ACTIVE_TIMESTAMP macro change the six 0 to 1.



6.1.2. General Setup

Child Update Wait Timeout: Verify that the THR_CHILD_ED_TIMEOUT_PERIOD_SEC matches the harness value.



SED Polling Rate: Change the THR_SED_POLLING_INTERVAL_MS to match the harness value.

Note that the harness value is in seconds, while the macro value in the code is in milliseconds.



After modifying all the above configuration options, compile and flash the software to the DUT.

Human interaction may be required while running Thread Test Harness certification cases. The test plan may ask the user to get the DUT random address, set a provisioning URL, add steering, and so on. To make these actions possible the NXP Kinetis software offers two options:

1. THCI DUT – when the end-product or module use a host_controlled_device software. This type of DUT needs Test Tool v12.6.3 or above to receive/transmit commands to the DUT.
2. Shell DUT – other projects like the router_eligible_device use a UART terminal with shell support to perform Thread Harness user actions.

7. Thread Test Plan

The Thread Certification Test Plan is a document that includes details for all test cases, network topology, purpose, test procedure, and pass criteria. It documents which test case are applicable to the type of Thread device being tested (Router, End Node, and so on), it lists the cases deprecated between Thread Specifications, and sets requirements for the test scenarios.

To download the latest Certification Test Plan, go to the members-only Thread Certification Committee and download it from the Thread Test Harness Information Folder.

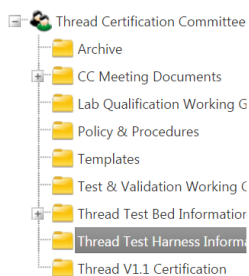
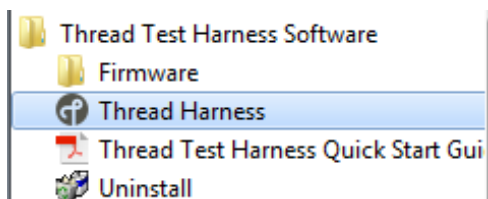


Figure 4. Thread Test Harness Information

8. Running Thread Certification Test Cases

To start running test cases

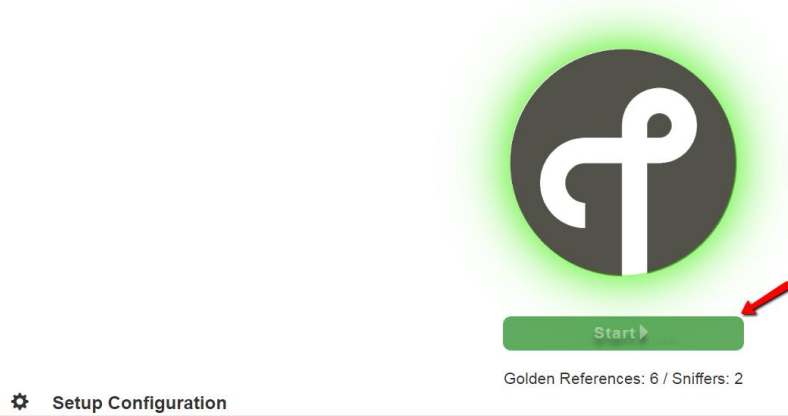
1. Connect all the golden devices to your PC using the USB hub
2. Open the Thread Harness application by double clicking on Start → All Programs → Thread Test Harness Software → Thread Harness.



3. Wait for the Discovering Connected Devices process to complete.



4. Click on Start.



5. After all Thread Test Bed board statuses have turned green, press the Next button to move to the Test Selection.



6. Connect the DUT and open Test Tool for THCI or a COM port for the other projects.

8.1. General considerations

Depending on the DUT functionality being tested (router, leader, commissioner, joiner) the DUT needs to be either out-of-band commissioned or not. Out-of-band commissioning means that DUT has all network parameters to directly attach to a network such as: extended PAN ID, network name, mesh-local, master key, and so on. There are two ways to configure for out-of-band commissioning, either through run-time commands or by using a pre-configured macro.

8.1.1. Run-time commands

Follow instructions from sections 9.12 and 9.13 of this document.

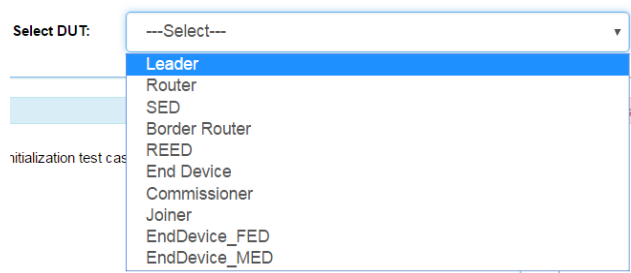
8.1.2. Pre-configured macro

Change the `THR_DEV_IS_OUT_OF_BAND_CONFIGURED` macro found in `app_thread_config.h` to `TRUE`, and re-program the DUT for this setting to take effect.

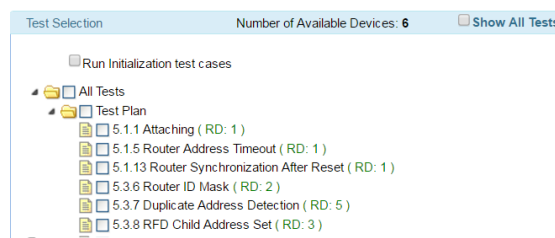
```
#ifndef THR_DEV_IS_OUT_OF_BAND_CONFIGURED
#define THR_DEV_IS_OUT_OF_BAND_CONFIGURED    TRUE
#endif
```

8.2. Running a DUT as Thread Leader

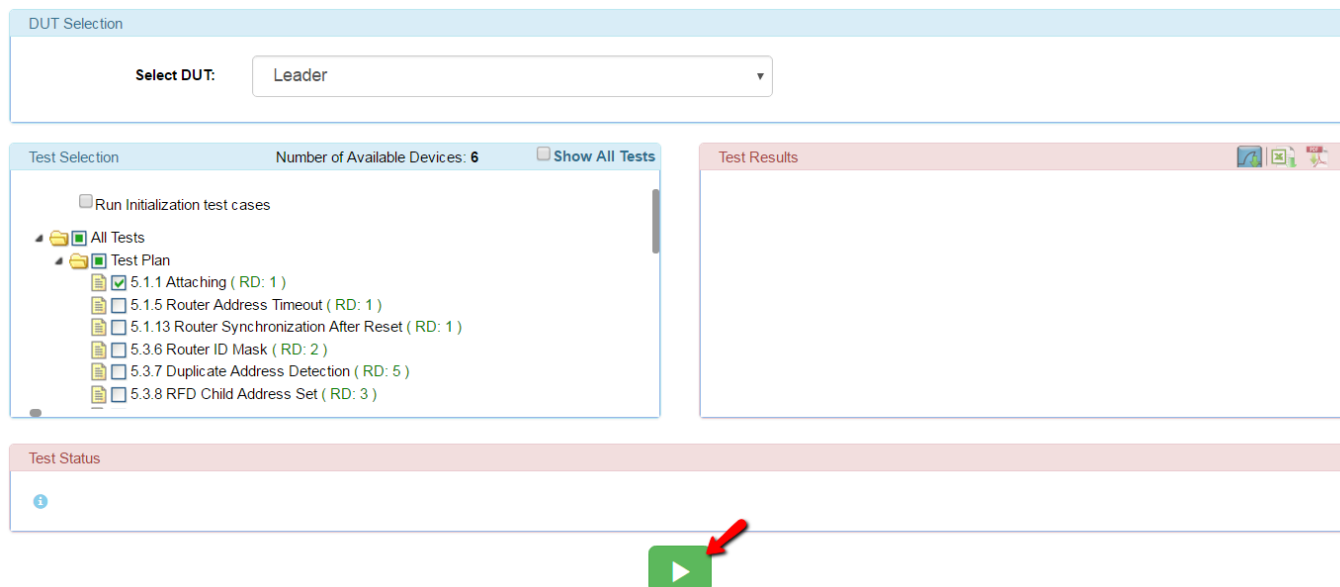
To start testing Leader test cases first select the Leader option in the DUT Selection within the Test Harness software.



The Test Selection section is now populated with the runnable test cases available for the selected Thread Harness setup.



Check one or more cases to run and click the green start button.



Follow the window pop-up to perform the test.

Start DUT !

Boot up the DUT and form network

OK

- Factory Reset the DUT using the instructions in Section 9.5
- Follow the instructions in Section 9.12 to set `iscommissioned = 1`
- Set `MLPrefix Mandatory` value as described in Section 9.26
- Create a new Thread network with the DUT By using the following commands (depending on if DUT is connected via a shell terminal or THCI in Test Tool).

DUT Shell

```
$thr create
```

DUT THCI

```
THR_CreateNwkRequest (InstanceID = 0)
```

Command: THR_CreateNwkRequest

InstanceID[1] 0x00

Look for the following information to verify that the DUT has taken the Leader role. Once you see that it is the Leader, press OK in the Test Harness application so that it knows that the network is up and running.

DUT Shell

```
$ thr create
Creating network...
$
Node has taken the Leader role
Created a new Thread network on channel 12 and PAN ID:0xfaca

Interface 0: 6LoWPAN
  Mesh local address (ML16): fd00:db8::ff:fe00:0
  Mesh local address (ML64): fd00:db8::6cec:4ad5:32e8:c506
(Local) Commissioner Started
```

DUT THCI



```

RX: THR_CreateNwkRequest 02 CE 1B 00 00 00 D4
RX: THR_CreateNwkConfirm 02 CF 1B 01 00 00 D5
RX: THR_EventNwkCreateConfirm 02 CF 51 CE 00 00 03 00 C9 00 00 11
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 60 1E
Sync [1 byte] = 02
OpGroup [1 byte] = CF
OpCode [1 byte] = 51
Length [2 bytes] = 00 CE
InstanceId [1 byte] = 00
EventStatus [2 bytes] = 00 03 (SelectBestChannel)
DataSize [2 bytes] = 00 C9
Data [201 bytes] = 00, 10, 00, 00, 00, 03, 02, 1E, 00, 01,
00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00,
00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00,
00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00,
00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00,
00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00,
CRC [1 byte] = 1E
RX: THR_EventGeneralConfirm 02 CF 54 05 00 00 02 00 00 9C
Sync [1 byte] = 02
OpGroup [1 byte] = CF
OpCode [1 byte] = 54
Length [2 bytes] = 00 05
InstanceId [1 byte] = 00
EventStatus [2 bytes] = 00 02 (Connected)
DataSize [2 bytes] = 00 00
CRC [1 byte] = 9C
RX: THR_EventGeneralConfirm 02 CF 54 05 00 00 0C 00 00 92
Sync [1 byte] = 02
OpGroup [1 byte] = CF
OpCode [1 byte] = 54
Length [2 bytes] = 00 05
InstanceId [1 byte] = 00
EventStatus [2 bytes] = 00 0C (Device is REED)
DataSize [2 bytes] = 00 00
CRC [1 byte] = 92
RX: THR_EventGeneralConfirm 02 CF 54 05 00 00 0B 00 00 95
Sync [1 byte] = 02
OpGroup [1 byte] = CF
OpCode [1 byte] = 54
Length [2 bytes] = 00 05
InstanceId [1 byte] = 00
EventStatus [2 bytes] = 00 0B (Device is Router)
DataSize [2 bytes] = 00 00
CRC [1 byte] = 95
RX: THR_EventGeneralConfirm 02 CF 54 05 00 00 0A 00 00 94
Sync [1 byte] = 02
OpGroup [1 byte] = CF
OpCode [1 byte] = 54
Length [2 bytes] = 00 05
InstanceId [1 byte] = 00
EventStatus [2 bytes] = 00 0A (Device is Leader)
DataSize [2 bytes] = 00 00
CRC [1 byte] = 94
RX: THR_EventNwkCreateConfirm 02 CF 51 05 00 00 04 00 00 9F
Sync [1 byte] = 02
OpGroup [1 byte] = CF
OpCode [1 byte] = 51
Length [2 bytes] = 00 05
InstanceId [1 byte] = 00
EventStatus [2 bytes] = 00 04 (GeneratePSKc)
DataSize [2 bytes] = 00 00
CRC [1 byte] = 9F
RX: THR_EventNwkCreateConfirm 02 CF 51 05 00 00 01 00 00 9A
Sync [1 byte] = 02
OpGroup [1 byte] = CF
OpCode [1 byte] = 51
Length [2 bytes] = 00 05
InstanceId [1 byte] = 00
EventStatus [2 bytes] = 00 01 (Success)
DataSize [2 bytes] = 00 00
CRC [1 byte] = 9A
RX: THR_EventNwkCommissioningIndication 02 CF 55 05 00 00 0A 00 00 9S
Sync [1 byte] = 02
OpGroup [1 byte] = CF
OpCode [1 byte] = 55
Length [2 bytes] = 00 05
InstanceId [1 byte] = 00
EventStatus [2 bytes] = 00 0A (CommissionerPetitionAccepted)
CRC [1 byte] = 9S

```

While the test is being executed, some user action windows pop-ups will appear follow section 9.2 to get the network Channel; section 9.6 to get the LL64 Address; and section 9.17 to get the Random Address.

After the current test case execution finishes, the result will be displayed in the Test Results Section.

Test Results	  
5.1.1 Attaching	Pass

Some test cases will require user actions besides the network creation. The following lists all the Leader cases that require an extra interaction, and a link to a section within this document on how to perform them.

Test 5.1.1

Get LL64 – section 9.6

Test 5.3.7

Get ML64 – section 9.11

Test 5.8.4

Get ML64 – section 9.11

Test 7.1.1

Set Prefix 2001 – section 9.14

Set Prefix 2002 – section 9.14

Test 7.1.3

Set Prefix 2001 – section 9.14

Set Prefix 2002 – section 9.14

Test 9.2.6

Set Active Timestamp – section 9.1

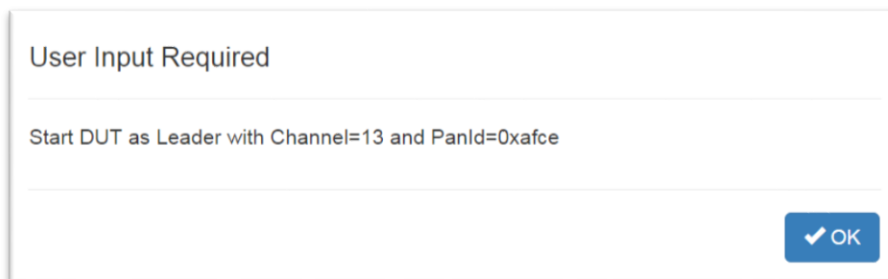
Test 9.2.11

Get RLOC – section 9.18

Test 9.2.12

Set Active Timestamp – section 9.1

Start DUT as Leader with Channel = secondary and PanId = 0xAFCE



Shell Commands

```
$ thr set iscommissioned 1
$ thr set mlprefix 0xFD000DB800000000
$ thr set channel 13
$ thr set panid 0xAFCE
$ thr create
```

```
$ thr set iscommissioned 1
Success!
$ thr set mlprefix 0xFD000DB800000000
Success!
$ thr set channel 13
Success!
$ thr set panid 0xafce
Success!
$ thr create
```

THCI Commands

```
THR_SetAttrRequest (InstanceId = 0, AttributeId = IsDevCommissioned, Index = 0x00, AttrSize = 0x01, Value = TRUE)
```

```
THR_SetAttrRequest (InstanceId = 0, AttributeId = MLPrefix, Index = 0x00, AttrSize = 0x11, PrefixData =
0xFD,0x00,0x0D,0xB8,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, Prefix Length = 0x10)
```

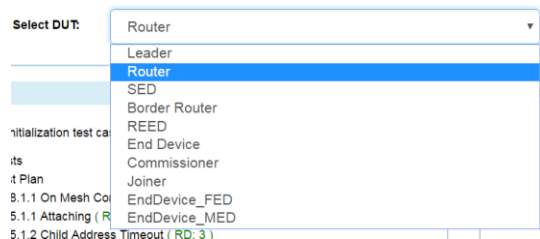
```
THR_SetAttrRequest (InstanceId = 0, AttributeId = Channel, Index = 0x00, AttrSize = 0x01, Value = 13)
```

```
THR_SetAttrRequest (InstanceId = 0, AttributeId = ShortPanId, Index = 0x00, AttrSize = 0x02, Value = 0xAFCE)
```

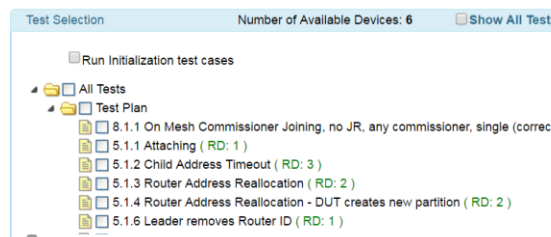
```
THR_CreateNwkRequest (InstanceId = 0)
```

8.3. Running a DUT as a Thread Router

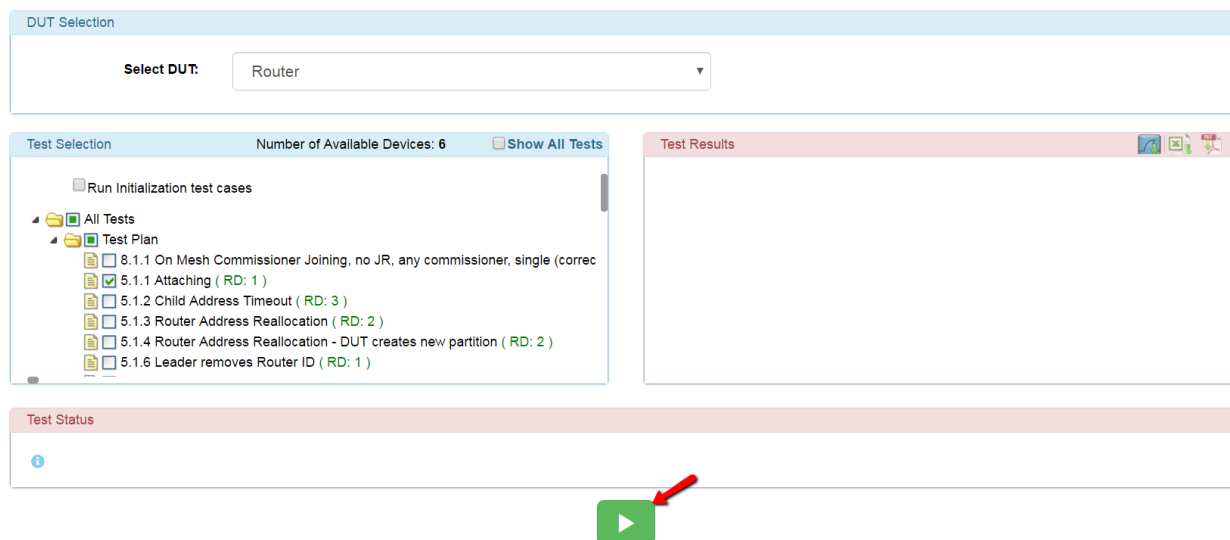
To start testing Router test cases first select this option at the DUT Selection within the Test Harness software.



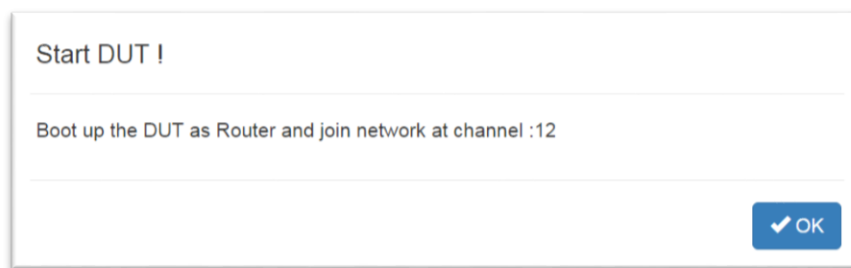
The Test Selection section is now populated with the runnable test cases available for the current Thread Harness setup.



Check one or more test cases and click the green start button.



A window pop-up like this will appear to perform the test

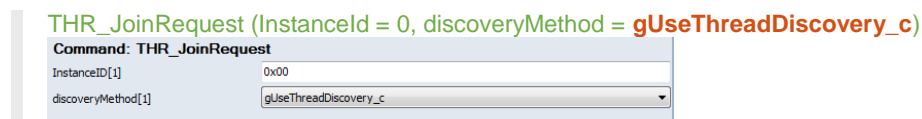


- Follow the instruction in section 9.12 to set `iscommissioned = 1`
- Attach the DUT to the Thread Harness network as a router, by using the following commands (depending if the DUT is connected via a shell terminal or THCI in Test Tool)

DUT Shell

```
$thr join
```

DUT THCI



- c) Look for the following information to verify that the DUT has taken the REED role. Once you see that it is REED, press OK in the Thread Test Harness application so it can wait up to 120s for the REED upgrade to Router.

DUT Shell

```
Joining network...
$
Joining a Thread network...
Attached to network with PAN ID: 0xface
```

DUT THCI

```

▶ RX: THR_Join.Confirm 02 CF 1C 01 00 00 D2
▲ RX: THR_EventNwkJoin.Confirm 02 CF 52 05 00 00 01 00 00 00 99
  Sync [1 byte] = 02
  OpGroup [1 byte] = CF
  OpCode [1 byte] = 52
  Length [2 bytes] = 00 05
  InstanceId [1 byte] = 00
  EventStatus [2 bytes] = 00 01 (Attaching)
  DataSize [2 bytes] = 00 00
  CRC [1 byte] = 99
▲ RX: THR_EventGeneral.Confirm 02 CF 54 05 00 00 07 00 00 00 99
  Sync [1 byte] = 02
  OpGroup [1 byte] = CF
  OpCode [1 byte] = 54
  Length [2 bytes] = 00 05
  InstanceId [1 byte] = 00
  EventStatus [2 bytes] = 00 07 (Connecting started)
  DataSize [2 bytes] = 00 00
  CRC [1 byte] = 99
▲ RX: THR_EventGeneral.Confirm 02 CF 54 05 00 00 02 00 00 00 9C
  Sync [1 byte] = 02
  OpGroup [1 byte] = CF
  OpCode [1 byte] = 54
  Length [2 bytes] = 00 05
  InstanceId [1 byte] = 00
  EventStatus [2 bytes] = 00 02 (Connected)
  DataSize [2 bytes] = 00 00
  CRC [1 byte] = 9C
▲ RX: THR_EventGeneral.Confirm 02 CF 54 05 00 00 16 00 00 00 88
  Sync [1 byte] = 02
  OpGroup [1 byte] = CF
  OpCode [1 byte] = 54
  Length [2 bytes] = 00 05
  InstanceId [1 byte] = 00
  EventStatus [2 bytes] = 00 16 (Child Id assigned)
  DataSize [2 bytes] = 00 00
  CRC [1 byte] = 88
▲ RX: THR_EventNwkJoin.Confirm 02 CF 52 05 00 00 02 00 00 00 9A
  Sync [1 byte] = 02
  OpGroup [1 byte] = CF
  OpCode [1 byte] = 52
  Length [2 bytes] = 00 05
  InstanceId [1 byte] = 00
  EventStatus [2 bytes] = 00 02 (Join Success)
  DataSize [2 bytes] = 00 00
  CRC [1 byte] = 9A
▲ RX: THR_EventGeneral.Confirm 02 CF 54 05 00 00 0C 00 00 00 92
  Sync [1 byte] = 02
  OpGroup [1 byte] = CF
  OpCode [1 byte] = 54
  Length [2 bytes] = 00 05
  InstanceId [1 byte] = 00
  EventStatus [2 bytes] = 00 0C (Device is REED)
  DataSize [2 bytes] = 00 00
  CRC [1 byte] = 92
```

- d) After the current test case execution finishes, the result will be displayed in the Test Results Section



Some test cases will require user actions besides network joining. The following lists all the Router cases that require an extra interaction, and a section link to this document with instructions on how to perform them.

Test 5.1.1

Get LL64 – section 9.6

Test 5.1.6

Get Short Address – section 9.19

Test 5.1.10

Get Random Address – section 9.17

Test 5.1.11

Get Random Address – section 9.17

Test 5.3.2

Get ML64 Address – section 9.11

Test 5.3.5

Get Short Address – section 9.19

Test 5.5.4

Get ML64 Address – section 9.11

Test 5.5.7

Get ML64 Address – section 9.11

Test 5.6.1

Get GUA Address – section

Test 5.6.9

Get Short Address – section 9.19

Test 5.7.1

Get ML64 Address – section 9.11

Test 5.8.2

Before joining the network Set the KeySwitchGuardTime attribute – section 9.24

Test 5.8.3

Before joining the network Set KeySwitchGuardTime attribute – section 9.24

Test 7.1.2

Set Prefix 2001 – section 9.14

Set Prefix 2002 – section 9.14

Test 7.1.4

Set Prefix 2001 – section 9.14

Set Prefix 2002 – section 9.14

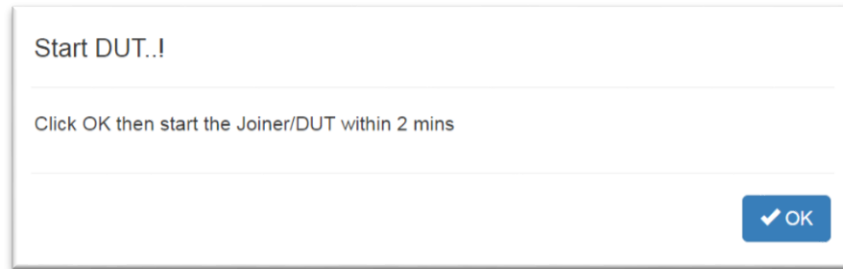
Test 7.1.5

Set Prefix 2001 – section 9.14

Set Prefix 2002 – section 9.14

Set Prefix 2003 – section 9.14

Test 8.x



Section 8 of the Thread Test Plan validates On-Mesh Commissioning. After a device is commissioned, it will receive the Thread network credentials. All Section 8 test cases require the DUT to be a Joiner Router with no network parameters pre-configured, hence Out-of-band commissioning must be disabled.

Get Factory MAC Address – section 9.4

Get PSKd – section 9.16

Test 8.2.5

Provisioning URL to www.threadgroup.org – section 9.15

Test 9.2.11

Set Active Timestamp – section 9.1

8.4. Running a DUT as Thread Sleepy End Device (SED)

Before starting SED cases in the Thread Harness application, some changes are required in the Kinetis Thread Certified IDE example project.

1. Open the DUT low_power_end_device
2. Inside the IDE, navigate to source folder and double click source.h
3. Change the THREAD_USE_SHELL macro value to 1.
4. Change the cPWR_UsePowerDownMode macro value to 0.

```
#define SOCK_DEMO          0
#define UDP_ECHO_PROTOCOL  0
#define USE_TEMPERATURE_SENSOR 1
#define THREAD_USE_SHELL   1
#define THREAD_USE_THCI    0
#define glpnIncluded_d     1
#define cPWR_UsePowerDownMode 0
```

5. Inside the IDE, navigate to nwk_ip → app → config folder
6. Double click app_thread_config.h
7. Inside the THR_DEFAULT_ATTR macro look for the .childEDReqFullNwkData attribute and change its value to FALSE

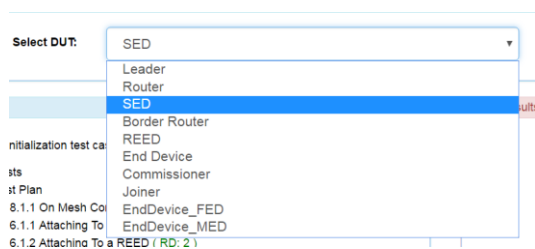
```

/*! Thread default attributes */
#define THR_DEFAULT_ATTR

.ieeeAddr = THR_EUI64_ADDR, \
.scanDuration = THR_SCAN_DURATION, \
.rxOnWhenIdle = RX_ON_IDLE, \
.sedPollInterval = THR_SED_POLLING_INTERVAL_MS, \
.uniqueExtAddr = THR_UNIQUE_EUI64_ADDR_ENABLED, \
.shortAddr = THR_ALL_FF16, \
.thrNwkCapabilitiesBitMap = THR_NWK_CAPABILITIES_BITMASK, \
.nwkKeySeq = THR_KEY_SEQ_NUMBER, \
.deviceType = THR_DEVICE_TYPE, \
.childReqFullNwkData = FALSE, \
.sedTimeoutPeriod = THR_SED_TIMEOUT_PERIOD_SEC, \
.edTimeoutPeriod = THR_CHILD_ED_TIMEOUT_PERIOD_SEC, \
.sedFastPollInterval = THR_FAST_POLLING_INTERVAL_MS, \
.devsCommissioned = THR_DEV_IS_OUT_OF_BAND_CONFIGURED, \
.joinLqiThreshold = THR_SELECT_PARENT_LQI_THRESHOLD, \
.selBestChannelEdThreshold = THR_SEL_BEST_CHANNEL_ED_THRESHOLD, \
.keySwitchGuardTime = THR_KEY_SWITCH_GUARD_TIME_HOURS, \
.parentHoldTime = 90, \
.nvmRestoreAutoStart = TRUE, \
.nvmRestore = THR_NVMRESTORE_ENABLE, \
.slaacPolicy = gThrSlaacRandom_c, \
hrGlobalNwkPrefix = THR_R0_GLOBAL_NWMPREFIX

```

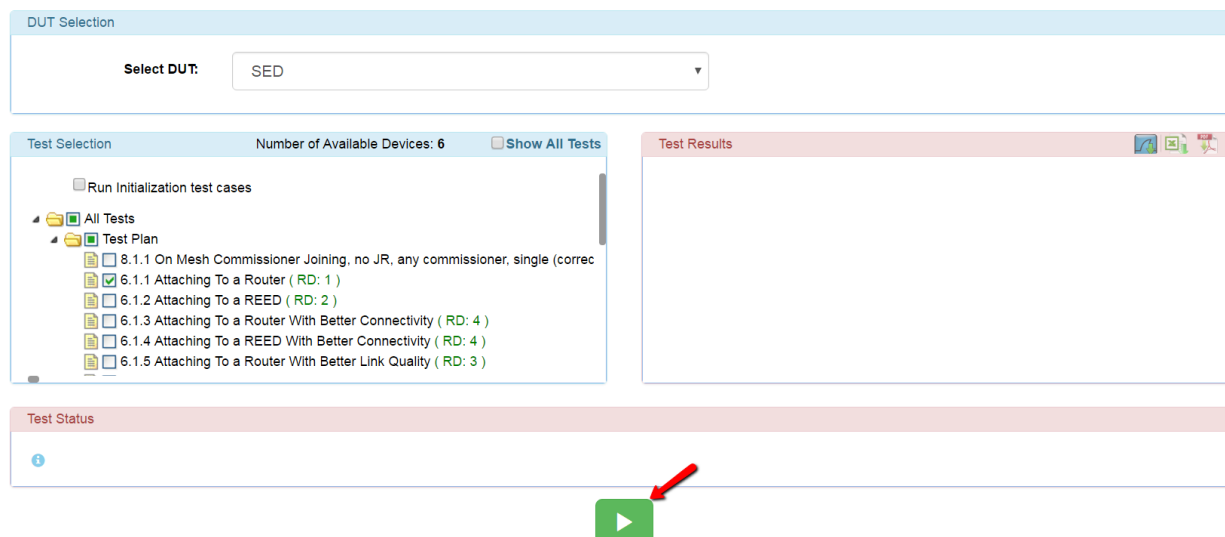
8. After modifying all the above configuration options, compile and flash the software to the DUT. To start testing Sleepy End Device test cases first choose this option at the DUT Selection



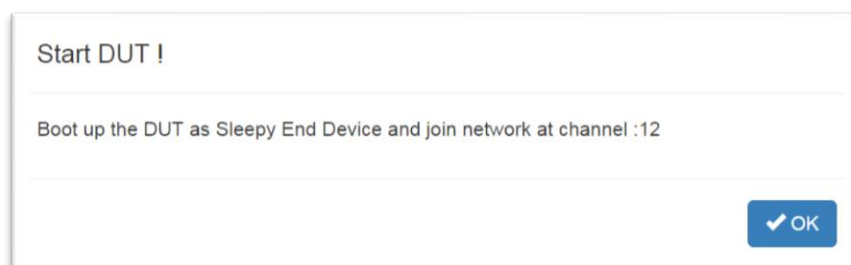
The Test Selection section is now populated with the runnable test cases available for the current Thread Harness setup



Check one or more test cases and click the green start button.



Follow the window pop-up to perform the test



- Follow Section 9.12 to set `iscommissioned = 1`
- Attach the DUT like a SED to the Thread Harness network by using the following commands (depending on if the DUT is connected by shell terminal or THCI in Test Tool)

DUT Shell

```
$thr join
```

DUT THCI

THR_SetAttr.Request (InstanceID = 0, AttributeID = **NwkCapabilities**, Index = 0, AttrSize = 1, Can create new network? = **FALSE**, Can become active router? = **FALSE**, Is polling end node? = **TRUE**, Reserved = **FALSE**)

Command: THR_SetAttrRequest	
InstanceID[1]	0x00
AttributeID[1]	NwkCapabilities
Index[1]	0x00
NwkCapabilities	
AttrSize[1]	0x01
Value[1]	
Can create new network?[1]	<input type="checkbox"/>
Can become active router?[1]	<input type="checkbox"/>
Is polling end node?[1]	<input checked="" type="checkbox"/>
Reserved[1]	<input type="checkbox"/>

THR_SetAttr.Request (InstanceID = 0, AttributeID = **EndDevice_ChildEDReqFullNwkData**, Index = 0, AttrSize = 01, Value = **FALSE**)

Command: THR_SetAttrRequest	
InstanceID[1]	0x00
AttributeID[1]	EndDevice_ChildEDReqFullNwkData
Index[1]	0x00
EndDevice_ChildEDReqFullNwkData	
AttrSize[1]	0x01
Value[1]	<input type="checkbox"/>

THR_JoinRequest (InstanceID = 0, discoveryMethod = **gUseThreadDiscovery_c**)

Command: THR_JoinRequest	
InstanceID[1]	0x00
discoveryMethod[1]	gUseThreadDiscovery_c

- c) Press OK after the device joined the network as a SED. To know when this event occurs look for the following information

DUT Shell

```
Joining network...
$
Joining a Thread network...
Attached to network with PAN ID: 0xfac6
Node started as End Device

Interface 0: 6LoWPAN
  Mesh local address <ML64>: fd00:db8::54f0:5736:176c:a625
  Mesh local address <ML16>: fd00:db8::ff:fe00:1
$
$ thr get devicerole
devrole: Sleepy End Device
```

DUT THCI

```
TX: THR_Join.Request 02 CE 1C 02 00 00 01 D1
RX: THR_Join.Confirm 02 CF 1C 01 00 00 D2
RX: THR_EventNwkJoin.Confirm 02 CF 52 05 00 00 01 00 00 00 99
RX: THR_EventGeneral.Confirm 02 CF 54 05 00 00 07 00 00 00 99
RX: THR_EventGeneral.Confirm 02 CF 54 05 00 00 02 00 00 00 9C
  Sync [1 byte] = 02
  OpGroup [1 byte] = CF
  OpCode [1 byte] = 54
  Length [2 bytes] = 00 05
  InstanceId [1 byte] = 00
  EventStatus [2 bytes] = 00 02 (Connected)
  DataSize [2 bytes] = 00 00
  CRC [1 byte] = 9C
RX: THR_EventGeneral.Confirm 02 CF 54 05 00 00 16 00 00 00 88
  Sync [1 byte] = 02
  OpGroup [1 byte] = CF
  OpCode [1 byte] = 54
  Length [2 bytes] = 00 05
  InstanceId [1 byte] = 00
  EventStatus [2 bytes] = 00 16 (Child Id assigned)
  DataSize [2 bytes] = 00 00
  CRC [1 byte] = 88
RX: THR_EventNwkJoin.Confirm 02 CF 52 05 00 00 02 00 00 00 9A
  Sync [1 byte] = 02
  OpGroup [1 byte] = CF
  OpCode [1 byte] = 52
  Length [2 bytes] = 00 05
  InstanceId [1 byte] = 00
  EventStatus [2 bytes] = 00 02 (Join Success)
  DataSize [2 bytes] = 00 00
  CRC [1 byte] = 9A
RX: THR_EventGeneral.Confirm 02 CF 54 05 00 00 0E 00 00 00 90
  Sync [1 byte] = 02
  OpGroup [1 byte] = CF
  OpCode [1 byte] = 54
  Length [2 bytes] = 00 05
  InstanceId [1 byte] = 00
  EventStatus [2 bytes] = 00 0E (Device is Sleepy End Device)
  DataSize [2 bytes] = 00 00
  CRC [1 byte] = 90
```

- d) After the current test case execution finishes, the result will be displayed in the Test Results Section



Some test cases will require user actions besides network joining. The following lists all the SED cases that require an extra interaction, and a section link within this document with instructions on how to perform them.

Test 6.4.2

Get ML64 – section 9.11

Test 6.6.1

Before joining the network disable the Key Switch Guard Time attribute by following section 9.24

Test 6.6.2

Before joining the network disable the Key Switch Guard Time attribute by following section 9.24

8.5. Running a DUT as Thread End Device

There are 3 types of End Device DUT: End Device, Full End Device, and Minimal End Device

8.5.1. End Device

For shell DUT devices, it's required to change made some changes in the Kinetis Thread Certified IDE example project before starting to run Thread Harness test cases.

1. Open the DUT end_device
2. Inside the IDE, navigate to source folder and double click source.h
3. Change the THR_DEFAULT_IS_FULL_END_DEVICE macro value to 0

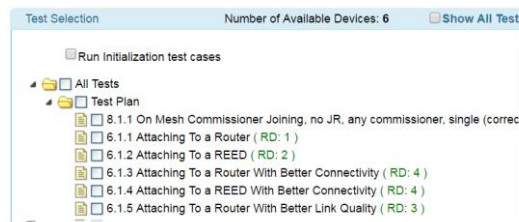
```
/* Node which cannot start network nor become its Leader nor act as Active Router */
#define THR_DEFAULT_CAN_CREATE_NEW_NETWORK 0
#define THR_DEFAULT_CAN_BECOME_ACTIVE_ROUTER 0
#define THR_DEFAULT_IS_FULL_END_DEVICE 0
#define THR_DEFAULT_IS_POLLING_END_DEVICE 0
```

4. After modifying all the above configuration options, compile and flash the software to the DUT.

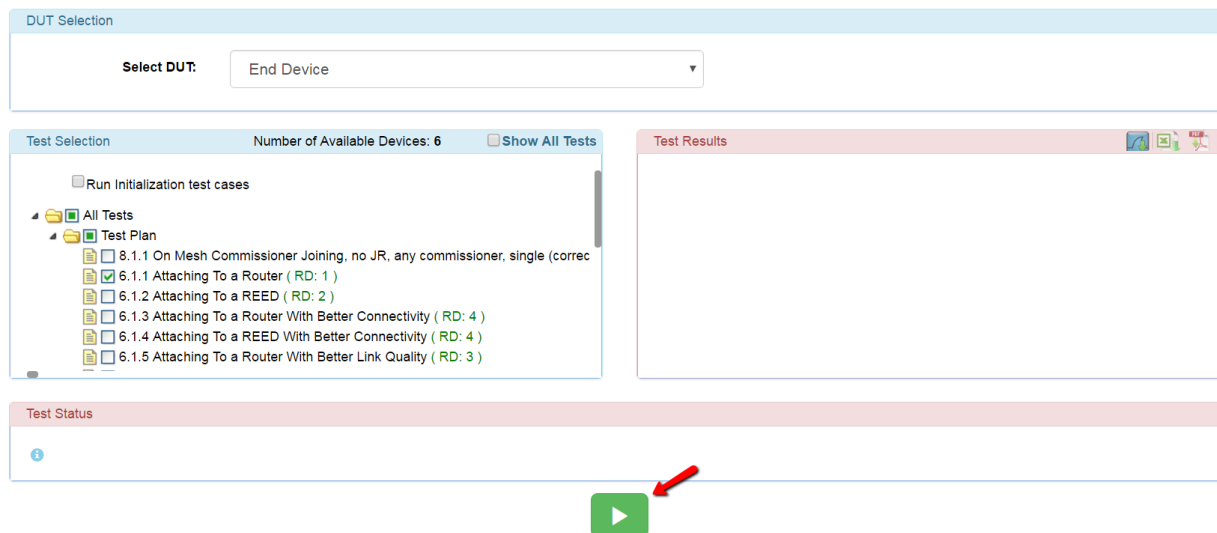
To start testing End Device test cases first select the End Device option in the DUT Selection within the Test Harness software.



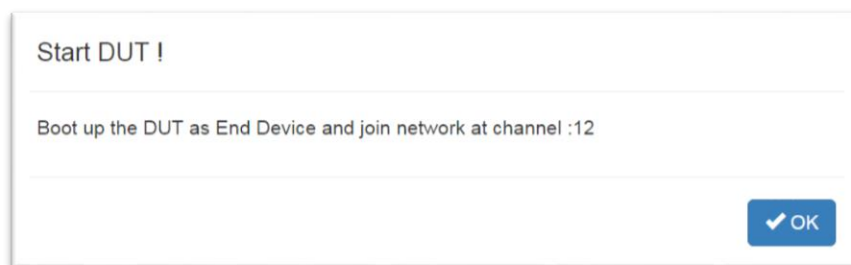
The Test Selection section is now populated with the runnable test cases available for the current Thread Harness setup



Check one or more test cases and click the green start button.



Follow the window pop-up to perform the test



- Follow the instruction in Section 9.12 to set `iscommissioned = 1`
- Set `MLPrefix Mandatory` value as described in Section 9.26
- Join the DUT to the Thread Test Harness Network as a Sleepy End Device by using the following commands (depending on if the DUT is connected via a shell terminal or THCI in Test Tool)

DUT Shell

```
$thr join
```

DUT THCI

```
THR_SetAttr.Request (InstanceId = 0, AttributeId = NwkCapabilities, Index = 0, AttrSize = 1, Can create new network? = FALSE, Can become active router? = FALSE, Is polling end node? = FALSE, Reserved = FALSE)
```

Command: THR_SetAttrRequest	
InstanceId[1]	0x00
AttributeId[1]	NwkCapabilities
Index[1]	0x00
NwkCapabilities	
AttrSize[1]	0x01
Value[1]	
Can create new network?[1]	<input type="checkbox"/>
Can become active router?[1]	<input type="checkbox"/>
Is polling end node?[1]	<input type="checkbox"/>
Is full Thread Device?[1]	<input type="checkbox"/>

```
THR_JoinRequest (InstanceId = 0, discoveryMethod = gUseThreadDiscovery_c)
```

Command: THR_JoinRequest

InstanceID[1] 0x00

discoveryMethod[1] gUseThreadDiscovery_c

- d) Look for the following information to verify that the device joined the network as a SED. Once you see that the device is attached to the network, press OK in the Thread Harness application so that it knows the DUT is ready for testing.

DUT Shell

```
$ thr join
Joining network...
$
Joining a Thread network...
Attached to network with PAN ID: 0xface

Interface 0: 6LoWPAN
  Mesh local address <ML64>: fd00:db8::f42e:3edb:9c4c:3c08
  Mesh local address <ML16>: fd00:db8::ff:fe00:1
$ thr get devicerole
devicerole: Minimal End Device
```

DUT THCI

```
► TK: THR_Join.Request 02 CE 1C 02 00 00 01 D1
► RK: THR_Join.Confirm 02 CF 1C 01 00 00 D2
► RK: THR_EventNwkJoin.Confirm 02 CF 52 05 00 00 01 00 00 00 99
▲ RK: THR_EventGeneral.Confirm 02 CF 54 05 00 00 07 00 00 00 99
  Sync [1 byte] = 02
  OpGroup [1 byte] = CF
  OpCode [1 byte] = 54
  Length [2 bytes] = 00 05
  InstanceId [1 byte] = 00
  EventStatus [2 bytes] = 00 07 (Connecting started)
  DataSize [2 bytes] = 00 00
  CRC [1 byte] = 99
▲ RK: THR_EventGeneral.Confirm 02 CF 54 05 00 00 02 00 00 00 9C
  Sync [1 byte] = 02
  OpGroup [1 byte] = CF
  OpCode [1 byte] = 54
  Length [2 bytes] = 00 05
  InstanceId [1 byte] = 00
  EventStatus [2 bytes] = 00 02 (Connected)
  DataSize [2 bytes] = 00 00
  CRC [1 byte] = 9C
▲ RK: THR_EventGeneral.Confirm 02 CF 54 05 00 00 16 00 00 00 88
  Sync [1 byte] = 02
  OpGroup [1 byte] = CF
  OpCode [1 byte] = 54
  Length [2 bytes] = 00 05
  InstanceId [1 byte] = 00
  EventStatus [2 bytes] = 00 16 (Child Id assigned)
  DataSize [2 bytes] = 00 00
  CRC [1 byte] = 88
► RK: THR_EventNwkJoin.Confirm 02 CF 52 05 00 00 02 00 00 00 9A
▲ RK: THR_EventGeneral.Confirm 02 CF 54 05 00 00 0D 00 00 00 93
  Sync [1 byte] = 02
  OpGroup [1 byte] = CF
  OpCode [1 byte] = 54
  Length [2 bytes] = 00 05
  InstanceId [1 byte] = 00
  EventStatus [2 bytes] = 00 0D (Device is End Device)
  DataSize [2 bytes] = 00 00
  CRC [1 byte] = 93
```

- e) After the current test case execution finishes, the result will be displayed in the Test Results Section



Some test cases will require user actions besides network joining. The following lists all the End Devices cases that require an extra interaction, and a section link within this document on how to perform them.

Test 6.1.3

Get Random Address – section 9.17

Test 6.1.4

Get Random Address – section 9.17

Test 6.1.5

Get Random Address – section 9.17

Test 6.1.6

Get Random Address – section 9.17

Test 6.4.2

Get ML64 – section 9.11

Test 6.6.1

Before joining the network disable the Key Switch Guard Time attribute by following section 9.24

Test 6.6.2

Before joining the network disable the Key Switch Guard Time attribute by following section 9.24

Test 9.x

Get ML64 – section 9.11

8.5.2. Full End Device (EndDevice_FED)

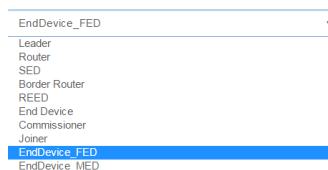
Before starting FED cases, some changes are required in the Kinetis Thread Certified IDE example project.

1. Open the DUT end_device
2. Inside the IDE, navigate to source folder and double click source.h
3. Change the THR_DEFAULT_IS_FULL_END_DEVICE macro value to 0

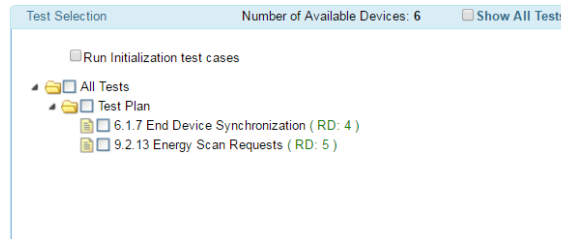
```
/* Node which cannot start network nor become its Leader nor act as Active Router */
#define THR_DEFAULT_CAN_CREATE_NEW_NETWORK 0
#define THR_DEFAULT_CAN_BECOME_ACTIVE_ROUTER 0
#define THR_DEFAULT_IS_FULL_END_DEVICE 1
#define THR_DEFAULT_IS_POLLING_END_DEVICE 0
```

4. After modifying all the above configuration options, compile and flash the software to the DUT.

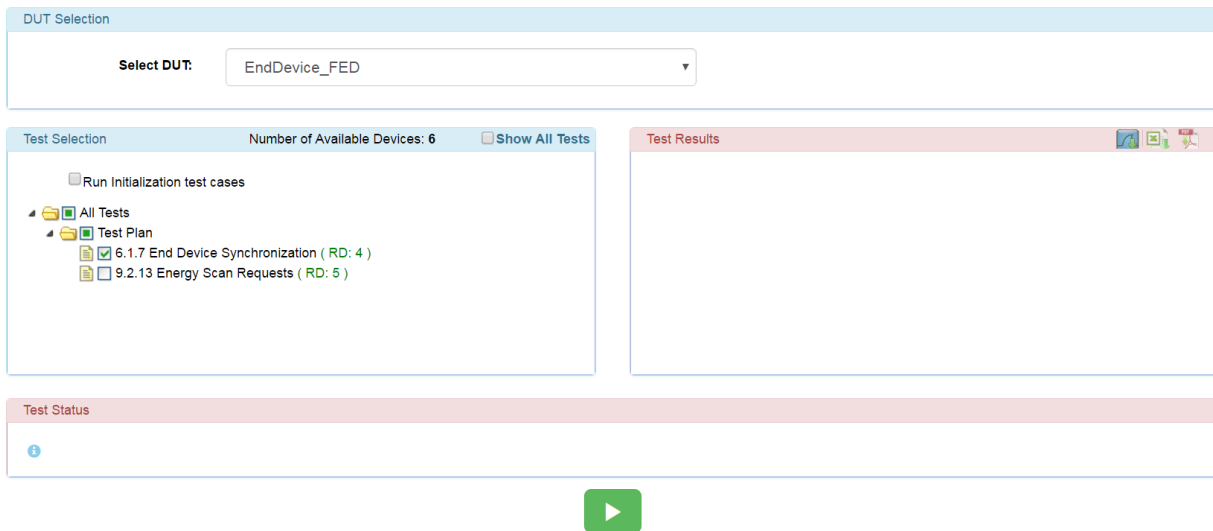
To start testing FED test cases first select the EndDevice_FED option in the DUT Selection within the Test Harness application.



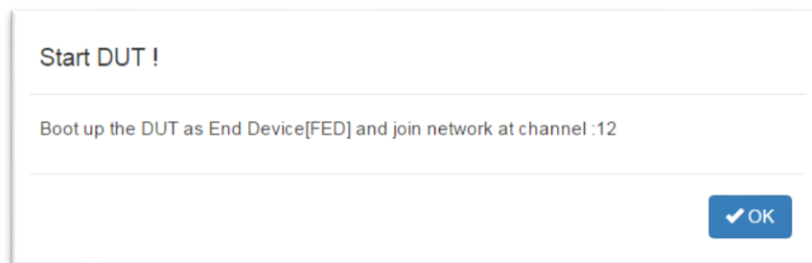
The Test Selection section is now populated with the runnable test cases available for the current Thread Harness setup



Check one or more test cases and click the green start button.



Follow the window pop-up to perform the test



- Follow Section 9.12 to set `iscommissioned = 1`
- Set `MLPrefix Mandatory` value as described in Section 9.26
- Attach the DUT as a FED to the Thread Test Harness network by using the following commands (depending on if the DUT is connected via shell terminal or THCI in Test Tool)

DUT Shell

```
$thr join
```

DUT THCI

THR_SetThresholdRequest(InstanceId = 0, ThresholdType = RouterUpgradeThreshold, Value = 0x01)

Command: THR_SetThresholdRequest	
InstanceId[1]	0x00
ThresholdType[1]	RouterUpgradeThreshold
Value[1]	0x01

THR_JoinRequest (InstanceId = 0, discoveryMethod = gUseThreadDiscovery_c)

Command: THR_JoinRequest	
InstanceId[1]	0x00
discoveryMethod[1]	gUseThreadDiscovery_c

- d) Look for the following information to verify that the DUT joined the network in the SED role. To know when this event occurs look for the following information:

DUT Shell

```
$ thr join
Joining network...
$
Joining a Thread network...
Attached to network with PAN ID: 0xface
$ thr get devicerole
devrole: Full End Device
```

DUT THCI

```
► TX: THR_Join.Request 02 CE 1C 02 00 00 01 D1
► RX: THR_Join.Confirm 02 CF 1C 01 00 00 D2
► RX: THR_EventNwkJoin.Confirm 02 CF 52 05 00 00 01 00 00 00 99
► RX: THR_EventGeneral.Confirm 02 CF 54 05 00 00 07 00 00 00 99
  Sync [1 byte] = 02
  OpGroup [1 byte] = CF
  OpCode [1 byte] = 54
  Length [2 bytes] = 00 05
  InstanceId [1 byte] = 00
  EventStatus [2 bytes] = 00 07 (Connecting started)
  DataSize [2 bytes] = 00 00
  CRC [1 byte] = 99
► RX: THR_EventGeneral.Confirm 02 CF 54 05 00 00 02 00 00 00 9C
  Sync [1 byte] = 02
  OpGroup [1 byte] = CF
  OpCode [1 byte] = 54
  Length [2 bytes] = 00 05
  InstanceId [1 byte] = 00
  EventStatus [2 bytes] = 00 02 (Connected)
  DataSize [2 bytes] = 00 00
  CRC [1 byte] = 9C
► RX: THR_EventGeneral.Confirm 02 CF 54 05 00 00 16 00 00 00 88
  Sync [1 byte] = 02
  OpGroup [1 byte] = CF
  OpCode [1 byte] = 54
  Length [2 bytes] = 00 05
  InstanceId [1 byte] = 00
  EventStatus [2 bytes] = 00 16 (Child Id assigned)
  DataSize [2 bytes] = 00 00
  CRC [1 byte] = 88
► RX: THR_EventNwkJoin.Confirm 02 CF 52 05 00 00 02 00 00 00 9A
► RX: THR_EventGeneral.Confirm 02 CF 54 05 00 00 0D 00 00 00 93
  Sync [1 byte] = 02
  OpGroup [1 byte] = CF
  OpCode [1 byte] = 54
  Length [2 bytes] = 00 05
  InstanceId [1 byte] = 00
  EventStatus [2 bytes] = 00 0D (Device is End Device)
  DataSize [2 bytes] = 00 00
  CRC [1 byte] = 93
```

- e) After the current test case execution finishes, the result will be displayed in the Test Results Section.

Test Results	
6.1.7 End Device Synchronization	Pass

Some test cases will require user actions besides joining a network. The following lists all the Full End Devices cases that require an extra interaction with instructions on how to perform them.

Test 9.2.13

Get ML64 – section 9.11

8.5.3. Minimal End Device (EndDevice_MED)

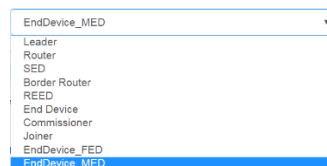
Before starting MED cases, some changes are required in the Kinetis Thread Certified IDE example project.

1. Open the DUT end_device
2. Inside the IDE, navigate to source folder and double click source.h
3. Change the THR_DEFAULT_IS_FULL_END_DEVICE macro value to 0

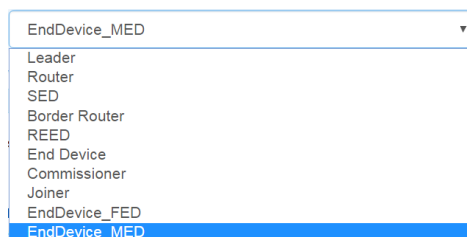
```
/* Node which cannot start network nor become its Leader nor act as Active Router */
#define THR_DEFAULT_CAN_CREATE_NEW_NETWORK 0
#define THR_DEFAULT_CAN_BECOME_ACTIVE_ROUTER 0
#define THR_DEFAULT_IS_FULL_END_DEVICE 0
#define THR_DEFAULT_IS_POLLING_END_DEVICE 0
```

4. After modifying all the above configuration options, compile and flash the software to the DUT.

To start testing MED test cases first select the EndDevice_MED option in the DUT Selection within the Test Harness application.



The Test Selection section is now populated with the runnable test cases available for the current Thread Harness setup.



Check one or more test cases and click the green start button.

The screenshot shows the Test Tool interface with the following sections:

- DUT Selection:** A dropdown menu labeled "Select DUT:" with "EndDevice_MED" selected.
- Test Selection:** A panel showing "Number of Available Devices: 6" and a "Show All Tests" checkbox. It contains a tree view with "All Tests" expanded, showing "Test Plan" with three items: "6.3.2 Network Update (RD: 1)", "9.2.12 Merging Networks Using Mle Announce (RD: 3)", and "9.2.13 Energy Scan Requests (RD: 5)". The "9.2.13 Energy Scan Requests" item is checked.
- Test Results:** An empty panel with a red header.
- Test Status:** A panel with a red header and a single status entry.



Follow the window pop-up to perform the test:

The dialog box titled "Start DUT !" contains the instruction: "Boot up the DUT as End Device[MED] and join network at channel :12". At the bottom right is an "OK" button with a checkmark icon.

- Follow Section 9.12 to set `iscommissioned = 1`
- Set `MLPrefix Mandatory` value as described in Section 9.26
- Attach the DUT as a MED to the Thread Test Harness network by using the following commands (depending on if the DUT is connected via shell terminal or THCI in Test Tool)

DUT Shell

```
$thr join
```

DUT THCI

```
THR_SetAttr.Request (Instanceid = 0, AttributeId = NwkCapabilities, Index = 0, AttrSize = 1, Can create new network? = FALSE, Can become active router? = FALSE, Is polling end node? = FALSE, Reserved = FALSE)
```

The screenshot shows the configuration for the `THR_SetAttrRequest` command. The fields are as follows:

- Instanceid[1]:** 0x00
- AttributeId[1]:** NwkCapabilities (selected from a dropdown)
- Index[1]:** 0x00
- NwkCapabilities:**
 - AttrSize[1]:** 0x01
 - Value[1]:**
 - Can create new network?[1]: ☐
 - Can become active router?[1]: ☐
 - Is polling end node?[1]: ☐
 - Is full Thread Device?[1]: ☐

```
THR_JoinRequest (Instanceid = 0, discoveryMethod = gUseThreadDiscovery_c)
```

Command: THR_JoinRequest

InstanceID[1] 0x00

discoveryMethod[1] glUseThreadDiscovery_c

- d) Look for the following information to verify that the DUT joined the network in the SED role. To know when this event occurs look for the following information:

DUT Shell

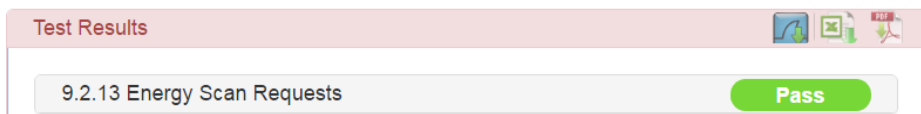
```
$ thr join
Joining network...
$
Joining a Thread network...
Attached to network with PAN ID: 0xface

Interface 0: 6LoWPAN
  Mesh local address <ML64>: fd00:db8::f42e:3edb:9c4c:3c08
  Mesh local address <ML16>: fd00:db8::ff:fe00:1
$ thr get devicerole
devrole: Minimal End Device
```

DUT THCI

```
TX: THR_Join.Request 02 CE 1C 02 00 00 01 D1
RX: THR_Join.Confirm 02 CF 1C 01 00 00 D2
RX: THR_EventNwkJoin.Confirm 02 CF 52 05 00 00 01 00 00 99
RX: THR_EventGeneral.Confirm 02 CF 54 05 00 00 07 00 00 99
  Sync [1 byte] = 02
  OpGroup [1 byte] = CF
  OpCode [1 byte] = 54
  Length [2 bytes] = 00 05
  InstanceId [1 byte] = 00
  EventStatus [2 bytes] = 00 07 (Connecting started)
  DataSize [2 bytes] = 00 00
  CRC [1 byte] = 99
RX: THR_EventGeneral.Confirm 02 CF 54 05 00 00 02 00 00 9C
  Sync [1 byte] = 02
  OpGroup [1 byte] = CF
  OpCode [1 byte] = 54
  Length [2 bytes] = 00 05
  InstanceId [1 byte] = 00
  EventStatus [2 bytes] = 00 02 (Connected)
  DataSize [2 bytes] = 00 00
  CRC [1 byte] = 9C
RX: THR_EventGeneral.Confirm 02 CF 54 05 00 00 16 00 00 88
  Sync [1 byte] = 02
  OpGroup [1 byte] = CF
  OpCode [1 byte] = 54
  Length [2 bytes] = 00 05
  InstanceId [1 byte] = 00
  EventStatus [2 bytes] = 00 16 (Child Id assigned)
  DataSize [2 bytes] = 00 00
  CRC [1 byte] = 88
RX: THR_EventNwkJoin.Confirm 02 CF 52 05 00 00 02 00 00 9A
RX: THR_EventGeneral.Confirm 02 CF 54 05 00 00 0D 00 00 93
  Sync [1 byte] = 02
  OpGroup [1 byte] = CF
  OpCode [1 byte] = 54
  Length [2 bytes] = 00 05
  InstanceId [1 byte] = 00
  EventStatus [2 bytes] = 00 0D (Device is End Device)
  DataSize [2 bytes] = 00 00
  CRC [1 byte] = 93
```

- e) After the current test case execution finishes, the result will be displayed in the Test Results Section.



Some test cases will require user actions besides joining a network. The following lists all the End Devices cases that require an extra interaction with instructions on how to perform them.

Test 9.x

Get ML64 – section 9.11

Test 9.2.12

User Input Required

Start DUT as End Device[MED] with Channel=13 and PanId=0xafce

OK

Shell Commands

```
$ thr set iscommissioned 1
$ thr set mlprefix 0xFD000DB800000000
$ thr set channel 13
$ thr set panid 0xAFCE
$ thr join
```

```
$ thr set iscommissioned 1
Success!
$ thr set mlprefix 0xFD000DB800000000
Success!
$ thr set channel 13
Success!
$ thr set panid 0xafce
Success!
$ thr create
```

THCI Commands

```
THR_SetAttrRequest (Instanceid = 0, AttributeId = IsDevCommissioned, Index = 0x00, AttrSize = 0x01, Value = TRUE)

THR_SetAttrRequest (Instanceid = 0, AttributeId = MLPrefix, Index = 0x00, AttrSize = 0x11, PrefixData =
0xFD,0x00,0x0D,0xB8,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, Prefix Length = 0x10)

THR_SetAttrRequest (Instanceid = 0, AttributeId = Channel, Index = 0x00, AttrSize = 0x01, Value = 13)

THR_SetAttrRequest (Instanceid = 0, AttributeId = ShortPanId, Index = 0x00, AttrSize = 0x02, Value = 0xAFCE)

THR_CreateNwkRequest (Instanceid = 0)
```

8.6. Running a DUT as Thread Commissioner

For shell DUT devices, it's required to make changes in the Kinetis Thread Certified IDE example project to enable management commands before starting to run Thread Harness test cases.

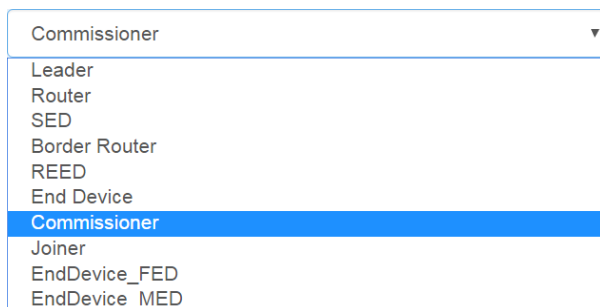
1. Open the DUT router_eligible_device or border_router example projects
2. Inside the IDE, navigate to source folder and double click source.h
3. Change the SHELL_DUT_COMMISSIONER macro value to 1

```
/* Enable CoAP Observe Client */
#define COAP_OBSERVE_CLIENT 0
#define SHELL_DUT_COMMISSIONER 1
```

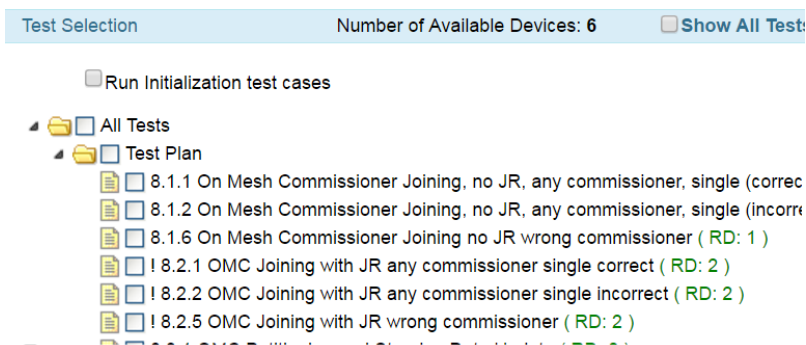
After modifying all of the above configuration options, compile and flash the software to the DUT.

To start testing Commissioner test cases, first select the Commissioner option at the DUT Selection within the Test Harness application.

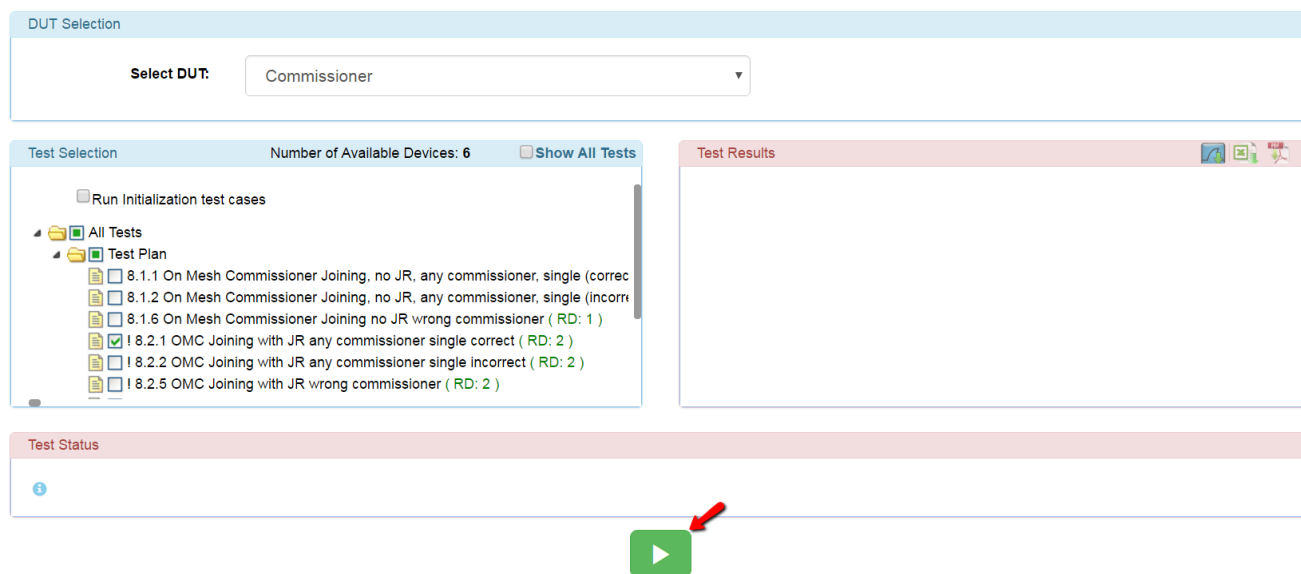
Running Thread Certification Test Cases



The Test Selection section is now populated with the test cases available for the current Thread Harness setup.

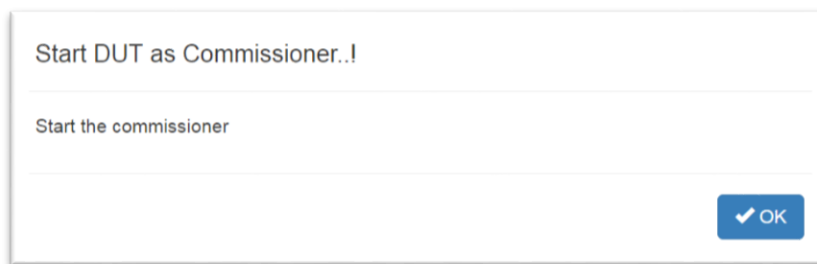


Check one or more test cases, and click the green start button.



Follow the window pop-up to perform the test.

8.6.1. DUT as Commissioner



- Factory Reset the DUT using the instructions in Section 9.5
- Follow the instructions in Section 9.12 to set is commissioned = 1
- Set MLPrefix Mandatory value as described in Section 9.26
- Clear the Provisioning URL using the instructions from Section 9.15
- Create a new Thread Network with the DUT by using the following commands (depending on if the DUT is connected via a shell terminal or THCI in Test Tool).

DUT Shell Command

```
$ thr create
$ thr set iscommissioned 1
Success!
$ thr set mlprefix 0xFD000DB800000000
Success!
$ thr set provisioningurl
Success!
$ thr create
Creating network...
```

DUT THCI Command

```
THR_CreateNwk.Request (InstanceId = 0)
```

- Look for the following information to verify that the DUT has taken the Leader role.

DUT Shell

```
Node has taken the Leader role
Created a new Thread network on channel 12 and PAN ID:0xfac
Interface 0: 6LoWPAN
Mesh local address <ML16>: fd00:db8::ff:fe00:0
Mesh local address <ML64>: fd00:db8::a484:9a6f:4b:52a7
<Local> Commissioner Started
```

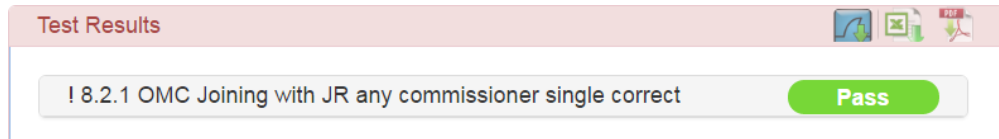
DUT THCI

```
✓ RX: THR_EventNwkCreate.Confirm 02 CF 51 05 00 00 01 00 00 00 9A
  Sync      [1 byte]   = 02
  OpGroup   [1 byte]   = CF
  OpCode    [1 byte]   = 51
  Length    [2 bytes]  = 00 05
  InstanceId [1 byte]  = 00
  EventStatus [2 bytes] = 00 01 (Success)
  DataSize  [2 bytes]  = 00 00
  CRC       [1 byte]   = 9A
```

- After you see that the DUT is the Leader it is time to start the Commissioner by following the instructions form Section 9.3

While the test is being executed some user-actions windows pop-ups will appear follow section 9.2 to get the network channel; or section 9.22 to add Steering Data.

After the current test case execution finishes the result will be displayed at the Test Results Section



8.6.2. DUT as Router and Commissioner



When this window pops-up follow the instructions from Section 9.21

Some test cases will require user actions besides Start the DUT as a commissioner. The following lists all the Commissioner cases that require an extra interaction, and a link to a section within this document on how to perform them.

All Commissioner Tests Cases

Add Steering Data – section 9.22

Test 8.1.6

Set provisioning URL to www.threadgroup.org – section 9.15

Test 8.2.5

Set provisioning URL to www.threadgroup.org – section 9.15

Test 8.3.1

Start DUT as Router and configure it as commissioner – section 9.21

Unregister the Commissioner – section 9.3

Make Commissioner [DUT] – section 9.3

Test 9.2.1

MGMT_COMM_GET – section 9.9

Test 9.2.2

MGMT_COMM_SET – section 9.10

Test 9.2.3

MGMT_ACTIVE_GET – section 9.7

Test 9.2.4

Start DUT as Router and configure it as commissioner – section 9.21

MGMT_ACTIVE_SET – section 9.8

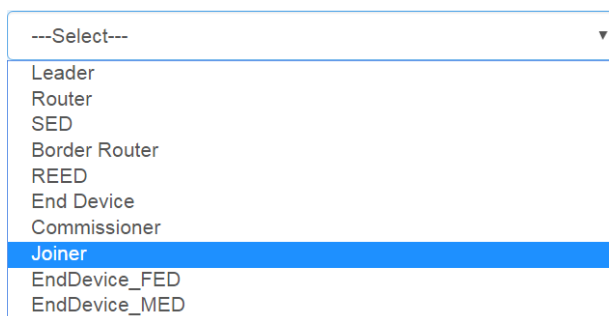
Test 9.2.14

Start DUT as Router and configure it as commissioner – section 9.21

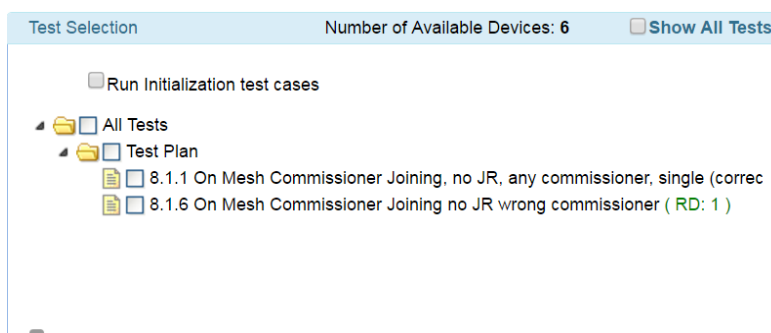
Send PANID Query – section 9.25

8.7. Running a DUT as Thread Joiner

To start testing Joiner test cases, first select the Joiner option at the DUT Selection within the Test Harness software.

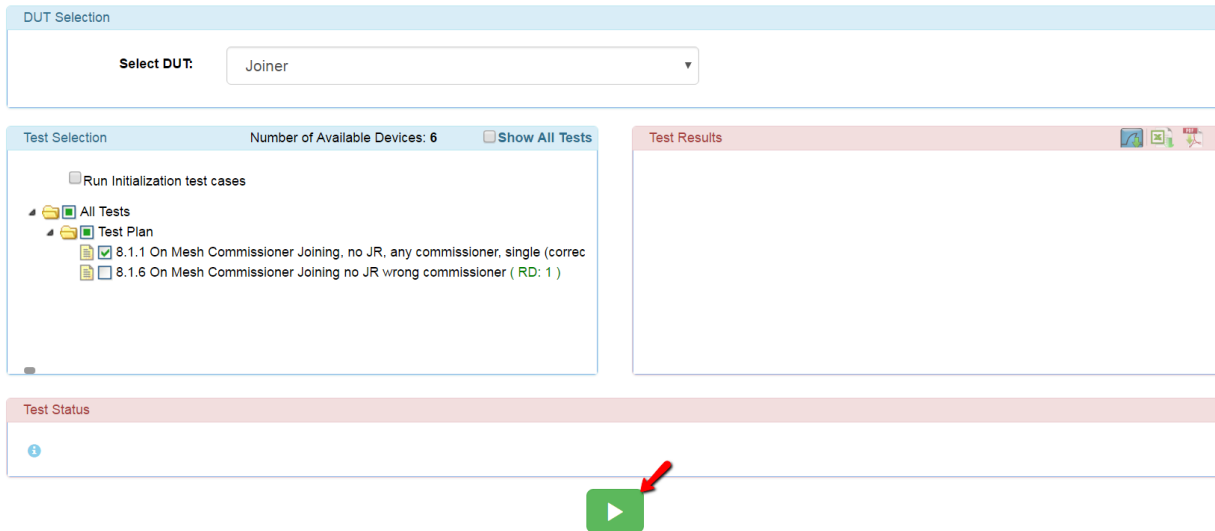


The Test Selection section is now populated with the runnable test cases available for the current Thread Harness setup.

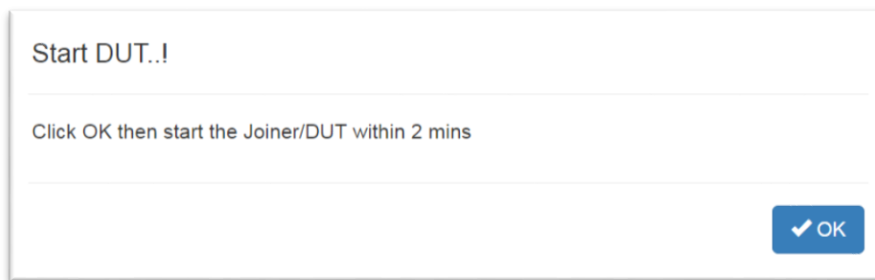


DUT Commands

Check one or more test cases, and click the green start button.

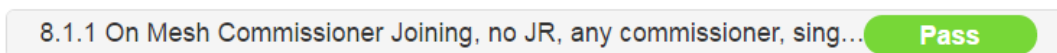


Follow the window pop-up to perform the test.



Follow the instructions in Section 9.20 to know how to start the DUT as a Joiner.

After the current test case execution finishes, the result will be displayed at the Test Results Section.



Some test cases will require user actions besides joining a network. The following lists all the Joiner cases that require an extra interaction, and a section link within this document on how to perform them.

Test 8.1.6

Set provisioning URL different than www.threadgroup.org – section 9.15

Test 8.2.2

Set provisioning URL equal to www.threadgroup.org – section 9.15

9. DUT Commands

Orange text represents input information to the Shell or Test Tool.

Violet text indicates information taken from the Shell or Test Tool.

9.1. Active TimeStamp

Set active timestamp to 10 seconds.

User Input Required

Set Activetimestamp on DUT to be 10 Seconds

OK

DUT Shell Command

```
$ thr set activetimestamp 167772160
```

```
$ thr set activetimestamp 167772160
Success!
```

DUT THCI Command

THR_SetAttrRequest (InstanceId = 0, AttributeId = **ActiveTimestamp**, Index = 0x00, AttrSize = **0x08**, ActiveSeconds = **0x0A0000000000**, ActiveTicks = 0x0000)

Command: THR_SetAttrRequest	
InstanceId[1]	0x00
AttributeId[1]	ActiveTimestamp
Index[1]	0x00
ActiveTimestamp	
AttrSize[1]	0x08
ActiveSeconds[6]	0x0A0000000000
ActiveTicks[2]	0x0000

9.2. Channel

DUT Shell Command

```
$ thr get channel
```

```
$ thr get channel
channel: 12
```

DUT THCI Command

THR_GetAttrRequest (InstanceId = 0, AttributeId = **Channel**, Index = 0x00)

```
TX: THR_GetAttr.Request 02 CE 17 03 00 00 04 00 DE
RX: THR_GetAttr.Confirm 02 CF 17 06 00 00 04 00 01 0C D7
  Sync      [1 byte] = 02
  OpGroup   [1 byte] = CF
  OpCode    [1 byte] = 17
  Length    [2 bytes] = 00 06
  InstanceId [1 byte] = 00
  AttributeId [1 byte] = 04 (Channel)
  Index     [1 byte] = 00
  Status    [1 byte] = 00 (Success)
  AttrSize  [1 byte] = 01
  Channel   [1 byte] = 0C
  CRC       [1 byte] = D7
```

Enter Channel

Enter the Leader/Commissioner channel.
Leave it blank if you don't know the channel (Scanning channel may take around 8 minutes)

✓ OK

9.3. Commissioner

Commissioner Start

DUT Shell Command

```
$ thr commissioner start
$ thr commissioner start
<Local> Commissioner Started
```

DUT THCI Command

MESHCOPI_StartCommissioner.Request (InstanceId = 0)

Command: MESHCOPI_StartCommissionerRequest

InstanceId[1] 0x00

Commissioner Stop

User Input Required

Unregister the Commissioner[DUT]

✓ OK

DUT Shell Command

```
$ thr commissioner stop
$ thr commissioner stop
```

DUT THCI Command

MESHCOPI_StopCommissioner.Request (InstanceId = 0)

Command: MESHCOPI_StopCommissionerRequest

InstanceId[1] 0x00

Commissioner Petition

User Input Required

Make Commissioner[DUT] sends a petition request to Leader

✓ OK

DUT Shell Command

```
$ thr commissioner start
```

```
$ thr commissioner start
<Local> Commissioner Started
```

DUT THCI Command

```
MESHCOPI_StartCommissioner.Request (InstanceId = 0)
```

Command: MESHCOPI_StartCommissionerRequest

InstanceId[1] 0x00

9.4. Factory MAC Address

Known as the IEEE EUI64, the factory address is composed of 8-bytes of a unique ID (3-bytes vendor IEEE OUI + 5-bytes vendor assigned mandatory to be unique).

DUT Shell Command

```
$ thr get eui
```

```
$ thr get eui
eui: 0x006037B861081F68
```



DUT Factory MAC Address Required !

Enter DUT Factory MAC Address.[Pass Hex Value]

0x006037B861081F68

✓ OK

Copy / Paste the eui value to the Thread Harness window, then click OK

DUT THCI Command

```
THR_GetAttrRequest (InstanceId = 0, AttributeId = IeeeExtendedAddr, Index = 0x00)
```

Command: THR_GetAttrRequest

InstanceId[1] 0x00

AttributeId[1] IeeeExtendedAddr

Index[1] 0x00

TX: THR_GetAttr.Request 02 CE 17 03 00 00 39 00 E3

RX: THR_GetAttr.Confirm 02 CF 17 0D 00 00 39 00 00 08 D9 6F A2 71 7F 37 60 00 A9

```

Sync      [1 byte] = 02
OpGroup   [1 byte] = CF
OpCode    [1 byte] = 17
Length    [2 bytes] = 00 0D
InstanceId [1 byte] = 00
AttributeId [1 byte] = 39 (IeeeExtendedAddr)
Index     [1 byte] = 00
Status    [1 byte] = 00 (Success)
AttrSize  [1 byte] = 08
IeeeExtendedAddr [8 bytes] = 00 60 37 7F 71 A2 6F D9
CRC       [1 byte] = A9

```



DUT Factory MAC Address Required !

Enter DUT Factory MAC Address.[Pass Hex Value]

0x0060377F71A26FD9

✓ OK

Copy / Paste the IeeeExtendedAddr value to the Thread Harness window pop-up, then press OK

9.5. Factory reset

DUT Shell Command

`$ factoryreset`

```
$ factoryreset
Reset Thread stack to factory defaults!
Reset MCU in: 499 milliseconds
SHELL starting...
MXP Thread v1.1.1.9
Enter "thr join" to join a network, or "thr create" to start new network
Enter "help" for other commands
```

DUT THCI Command

Command: `THR_FactoryResetRequest`

`THR_FactoryResetRequest`

```
▶ TX: THR_FactoryReset.Request 02 CE 1F 00 00 D1
▶ RX: THR_FactoryReset.Confirm 02 CF 1F 01 00 00 D1
▶ RX: THR_CpuReset.Indication 02 CF 22 05 00 01 F3 01 00 00 1B
▶ RX: THR_CpuReset.Indication 02 CF 22 2A 00 00 0A 46 52 44 4D 2D
45 4E 24 30 4E 58 50 00 91 11 07 31 2E 31 2E 31 00 00 8B
▶ RX: THR_EventGeneral.Confirm 02 CF 54 05 00 00 03 00 00 00 9D
  Sync      [1 byte] = 02
  OpGroup   [1 byte] = CF
  OpCode    [1 byte] = 54
  Length    [2 bytes] = 00 05
  InstanceId [1 byte] = 00
  EventStatus [2 bytes] = 00 03 (Reset to factory default)
  DataSize  [2 bytes] = 00 00
  CRC       [1 byte] = 9D
```

9.6. LL64 Address

Also known as the Link Local Address, the LL64 is used for direct neighbor communication.

LL64 Address

Enter DUT LL64 Address.

FE80:0000:0000:0000:146E:0A00:0000:0001

OK

DUT Shell Command

`$ ifconfig`

`$ ifconfig`

```
Interface 0: 6LoWPAN
  Link local address <LL64>: Fe80::d1c:1ff3:1c91:b105
  Mesh local address <ML16>: fd00:db8::ff:fe00:0
  Mesh local address <ML64>: fd00:db8::3da0:2e2f:673e:69f5
  Link local all Thread Nodes<MCast>: ff32:40:fd00:db8::1
  Realm local all Thread Nodes<MCast>: ff33:40:fd00:db8::1
```

Copy / Paste the Link local address (LL64) value from the shell terminal to the Thread Harness window pop-up, then press OK

DUT THCI Command

`THR_GetThreadIpAddrRequest` (InstanceId = 0, AddressType = `Link_Local_64`)

Command: `THR_GetThreadIpAddrRequest`

InstanceId[1] 0x00
AddressType[1] Link_Local_64

```

TX: THR_GetThreadIpAddr.Request 02 CE 19 02 00 00 00 D5
RX: THR_GetThreadIpAddr.Confirm 02 CF 19 14 00 00 00 00 01 7B 46 DD 34 B7 39 FF 21
  Sync      [1 byte] = 02
  OpGroup   [1 byte] = CF
  OpCode    [1 byte] = 19
  Length    [2 bytes] = 00 14
  InstanceId [1 byte] = 00
  Status     [1 byte] = 00 (Success)
  AddressType [1 byte] = 00
  NoOfIpAddr [1 byte] = 01
  AddressList [16 bytes] = IPV6[1] FE80:0000:0000:0000:021FF:39B7:34DD:467B
  CRC       [1 byte] = 39

```

LL64 Address

Enter DUT LL64 Address.

FE80:0000:0000:0000:021FF:39B7:34DD:467B

OK

Copy → Change the format → Paste the IPV6 address from the DUT THCI Test Tool to the Thread Harness window pop-up, then press OK

9.7. MGMT_ACTIVE_GET

The commissioner may send commands to the Border Agent over the Commissioning Session to get a view of the current network state.

Get All TLVs

DUT Shell Command

Send MGMT_ACTIVE_GET.req

Send MGMT_ACTIVE_GET.req to fd00:0db8:0000:0000:38aa:b607:95bf:11f8

CoAP Payload:
Get All TLVs

OK

```

$ thr mgmt activeget 0xfd000db80000000038aab60795bf11f8
$ Success!
$ ch: 12
pan: 0xface
xpan: 0x000DB80000000000
nwkname: GRL
pskc: 0x10B95765596AB9D0B86CEBDD0FA24DA3
masterkey: 0x00112233445566778899AABBCCDDEEFF
mlprefix: 0xFD000DB800000000
secpol: 672, 0xf8
activets: 1 sec, 0 ticks
chnsk: 0x00080000

```

DUT THCI Command

Send MGMT_ACTIVE_GET.req

Send MGMT_ACTIVE_GET.req to fd00:0db8:0000:0000:b160:d36e:5a89:ad10

CoAP Payload:
Get All TLVs

OK

Command: MESHCOPI_MgmtActiveGetRequest

InstanceId[1]	0x00
IPAddress[16]	fd00:0db8:0000:0000:b160:d36e:5a89:ad10
NumberOfTlvs[1]	0x00

Get Channel Mask, MLPrefix, and Network Name

DUT Shell Command

Send MGMT_ACTIVE_GET.req

Send MGMT_ACTIVE_GET.req to fd00:0db8:0000:0000:40c6:839c:90a8:4b7d

CoAP Payload:

1. Network Channel Mask
2. Network Mesh Local ULA prefix
3. Network Name

OK

DUT Commands

```
$thr mgmt. activeget fd00:0db8:0000:0000:40c6:839c:90a8:4b7d chmsk mlprefix nwname
```

```

$thr mgmt. activeget fd00:0db8:0000:0000:40c6:839c:90a8:4b7d chmsk mlprefix nwname
Success!
? nokname: CRL
mlprefix: 0xFD000DB800000000
chmsk: 0x00080000

```

DUT THCI Command

Send MGMT_ACTIVE_GET.req

Send MGMT_ACTIVE_GET.req to fd00:0db8:0000:0000:b160:d36e:5a89:ad10

CoAP Payload:

1. Network Channel Mask
2. Network Mesh Local ULA prefix
3. Network Name



OK

Command: MESHCOPI_MgmtActiveGetRequest	
InstanceId[1]	0x00
IPAddress[16]	FD00:0DB8:0000:0000:B160:D36E:5A89:AD10
NumberOfTlvids[1]	0x03
Tlvids[3]	
Tlvid[1]	ChannelMask
Tlvids_1[3]	
Tlvid[1]	MeshLocalUla
Tlvids_2[3]	
Tlvid[1]	NetworkName

Get Channel, MLPrefix, NetworkName, Scan Duratio, Energy List

DUT Shell Command

```
thr mgmt activeget fd00:0db8:0000:0000:40c6:839c:90a8:4b7d ch mlprefix nwname scan energylist
```

DUT THCI Command

Command: MESHCOPI_MgmtActiveGetRequest	
InstanceId[1]	0x00
IPAddress[16]	FD00:0DB8:0000:0000:B160:D36E:5A89:AD10
NumberOfTlvids[1]	0x05
Tlvids[5]	
Tlvid[1]	Channel
Tlvids_1[5]	
Tlvid[1]	MeshLocalUla
Tlvids_2[5]	
Tlvid[1]	NetworkName
Tlvids_3[5]	
Tlvid[1]	Channel
Tlvids_4[5]	
Tlvid[1]	Channel

Go to the Enter Command Section and hard-code:

MESHCOPI_MgmtActiveGet.Request 00 FD000DB800000000B160D36E5A89AD10 05 00 07 03 38 39

9.8. MGMT_ACTIVE_SET

A commissioner sends a MGMT_ACTIVE_SET message to change one or more Active Operational Datasets in the Leader.

Send MGMT_ACTIVE_Set.req

Send MGMT_ACTIVE_Set.req to leader

CoAP Payload:

1. Commissioner Session ID TLV (valid)

2. Active Timestamp TLV: 101, 0

3.Channel Mask TLV: 00:04:00:1f:ff:e04.Extended PAN ID TLV: 00:0d:b7:00:00:00:00:00

5.Network Name TLV: 'GRL'

6. PSKc TLV: 74:68:72:65:61:64:6a:70:61:6b:65:74:65:73:74:00

7. Security Policy TLV: 0e:10:c0

✓ OK

9.9. MGMT_COMM_GET

DUT gets Commissioner Dataset parameters while a commissioner is active.

Send MGMT_COMM_GET.req

Send MGMT_COMM_GET.req to fd00:0db8:0000:0000:309c:9be7:1b56:0de2
CoAP Payload:
Get all Tlvs

✓ OK

DUT Shell Command

```
$thr mgmt activeset sid 0x0002 activets 101 chmsk 0x001ffe0 xpan 0x000db70000000000 nwname GRL pskc
0x7468726561646a70616b657465737400 secpol 3600 0xc0
```

DUT THCI Command

Command: MESHOP_MgmtActiveSetRequest	
InstanceId[1]	0
IPAddress[16]	FD00:0DB8:0000:0000:0000:00FF:FE00:FC00
SessionIdEnable[1]	<input checked="" type="checkbox"/>
SessionId[1][2]	0x0002
BorderRouterLocatorEnable[1]	<input type="checkbox"/>
NewSessionIdEnable[1]	<input type="checkbox"/>
SteeringDataEnable[1]	<input type="checkbox"/>
ChannelEnable[1]	<input type="checkbox"/>
ChannelMaskEnable[1]	<input checked="" type="checkbox"/>
ChannelPage[1][1]	0
ChannelMaskLength[1][1]	4
ChannelMask[4][1]	0x00,0x1F,0xFF,0xE0
XpanIdEnable[1]	<input checked="" type="checkbox"/>
XpanId[1][8]	0x000DB70000000000
MLPrefixEnable[1]	<input type="checkbox"/>
MasterKeyEnable[1]	<input type="checkbox"/>
NwkNameEnable[1]	<input checked="" type="checkbox"/>
NwkNameSize[1][1]	3
NwkName[3][1]	GRL
PanIdEnable[1]	<input type="checkbox"/>
PSKcEnable[1]	<input checked="" type="checkbox"/>
PSKcSize[1][1]	15
PSKc[15][1]	threadpkttest
PolicyEnable[1]	<input checked="" type="checkbox"/>
RotationInterval[1][2]	3600
Policy[1][1]	
B bit[1]	<input checked="" type="checkbox"/>
C bit[1]	<input checked="" type="checkbox"/>
R bit[1]	<input type="checkbox"/>
N bit[1]	<input type="checkbox"/>
O bit[1]	<input type="checkbox"/>
ActiveTimestampEnable[1]	<input checked="" type="checkbox"/>
ActiveSeconds[1][6]	101
ActiveTicks[1][2]	0
PendingTimestampEnable[1]	<input type="checkbox"/>
DelayTimerEnable[1]	<input type="checkbox"/>
BogusTlvEnable[1]	<input type="checkbox"/>

9.10. MGMT_COMM_SET

A commissioner can set Commissioner Datasets values by sending MGMT_COMM_SET messages to the Leader

Send MGMT_COMM_SET.req

Send MGMT_Commissioner_Set.req to leader
CoAP Payload:
1.Commissioner Session ID TLV (0x0000)
2.Steering Data TLV (0xFF)

✓ OK

DUT Shell Command

```
$thr mgmt commset sid 0x000 steering 0xFF
```

DUT THCI Command

Command: MESHCOPI_MgmtCommissionerSetRequest

InstanceId[1]	0
IPAddress[16]	0000:0000:0000:0000:0000:0000:0000:0000
SessionIdEnable[1]	<input checked="" type="checkbox"/>
SessionId[1][2]	0x0000
BorderRouterLocatorEnable[1]	<input type="checkbox"/>
NewSessionIdEnable[1]	<input type="checkbox"/>
SteeringDataEnable[1]	<input checked="" type="checkbox"/>
SteeringDataSize[1][1]	1
SteeringData[1][1]	0xFF
ChannelEnable[1]	<input type="checkbox"/>
ChannelMaskEnable[1]	<input type="checkbox"/>
XpanIdEnable[1]	<input type="checkbox"/>
MLPrefixEnable[1]	<input type="checkbox"/>
MasterKeyEnable[1]	<input type="checkbox"/>
NwkNameEnable[1]	<input type="checkbox"/>
PanIdEnable[1]	<input type="checkbox"/>
PSKcEnable[1]	<input type="checkbox"/>
PolicyEnable[1]	<input type="checkbox"/>
ActiveTimestampEnable[1]	<input type="checkbox"/>
PendingTimestampEnable[1]	<input type="checkbox"/>
DelayTimerEnable[1]	<input type="checkbox"/>
BogusTlvEnable[1]	<input type="checkbox"/>

9.11. ML64 Address

The ML64 Address is highly recommended to be used by application software because it is randomly generated and it is persistent over reboots.

DUT Shell Command

```
$ ifconfig
```

```
$ ifconfig
Interface 0: 6LoWPAN
Link local address <LL64>: fe80::49e4:bfd6:6fca:c54c
Mesh local address <ML16>: fd00:db9::ff:fe00:c00
Mesh local address <ML64>: fd00:db9::bc17:6d1c:69ec:897
Link local all Thread Nodes<MCast>: ff32:40:fd00:db9::1
Realm local all Thread Nodes<MCast>: ff33:40:fd00:db9::1
$
```

ML64 Address

Enter DUT ML64 Address.

fd00:db9::bc17:6d1c:69ec:897

✓ OK

DUT Commands

Copy / Paste the Mesh local address (ML64) value from the shell terminal to the Thread Harness window pop-up, then click OK

DUT THCI Command

THR_GetThreadIpAddrRequest (InstanceId = 0, AddressType = **ML-EID**)

Command: THR_GetThreadIpAddrRequest

InstanceId[1] 0x00

AddressType[1] **ML-EID**
Link_Local_64
ML-EID
RLOC
Global

TX: THR_GetThreadIpAddr_Request 02 CE 19 02 00 00 01 D4

RX: THR_GetThreadIpAddr_Confirm 02 CF 19 14 00 00 00 01 01 33 EB 71 34 09

Sync [1 byte] = 02
OpGroup [1 byte] = CF
OpCode [1 byte] = 19
Length [2 bytes] = 00 14
InstanceId [1 byte] = 00
Status [1 byte] = 00 (Success)
AddressType [1 byte] = 01
NoOfIpAddr [1 byte] = 01
AddressList [16 bytes] = IPV6[1] **FD00:0DB8:0000:0000:3D2A:F809:3471:EB33**
CRC [1 byte] = F1

ML64 Address

Enter DUT ML64 Address.

FD00:0DB8:0000:0000:3D2A:F809:3471:EB33

OK

Copy the IPV6 value form Test Tool; change the format form hexadecimal to IPV6; paste it in the Thread Harness; then click OK

9.12. In-band-commissioning device

DUT Shell Command

`$ thr set iscomissioned 0`

```
$ thr set iscomissioned 0
Success!
```

DUT THCI Command

THR_SetAttrRequest (InstanceId = 0, AttributeId = **IsDevCommissioned**, Index = 0x00, AttrSize = 0x01, Value = **FALSE**)

Command: THR_SetAttrRequest

InstanceId[1] 0x00

AttributeId[1] **IsDevCommissioned**

Index[1] 0x00

IsDevCommissioned

AttrSize[1] 0x01

Value[1] ☐

9.13. Out-of-band commissioning device

DUT Shell Command

`$ thr set iscomissioned 1`

```
$ thr set iscomissioned 1
Success!
```

DUT THCI Command

THR_SetAttrRequest (InstanceId = 0, AttributeId = **IsDevCommissioned**, Index = 0x00, AttrSize = 0x01, Value = **TRUE**)

Command: THR_SetAttrRequest

InstanceId[1] 0x00

AttributeId[1] **IsDevCommissioned**

Index[1] 0x00

IsDevCommissioned

AttrSize[1] 0x01

Value[1] ☒

9.14. Prefix on DUT

2001::



Figure 5. Prefix 2001:: Thread Harness pop-up

DUT Shell Command

```
$ thr nwldata add slaac -p 2001::0 -len 64 -t 60500
$thr sync nwldata
```

```
$ thr nwldata add slaac -p 2001::0 -len 64 -t 60500
Success!
$ thr sync nwldata
Success!
```

DUT THCI Command

THR_BrPrefixAddEntry.Request(InstanceId = 0x00, prefixLength = 0x40, PrefixValue = 2001:0000:0000:0000:0000:0000:0000:0000, PrefixReservedFlags = 0x00, **P_Default = TRUE**, P_Configure = FALSE, P_Dhcp = FALSE, **P_Slaac = TRUE**, **P_Prefered = TRUE**, P_preference = Medium, prefixLifetime = 0xFFFFFFFF, Prefix Advertised = TRUE)

Command: THR_BrPrefixAddEntryRequest	
InstanceId[1]	0x00
prefixLength[1]	0x40
PrefixValue[16]	2001:0000:0000:0000:0000:0000:0000:0000
PrefixFlagsReserved[1]	0x00
PrefixFlags[1]	
P_On_Mesh[1]	<input checked="" type="checkbox"/>
P_Default[1]	<input checked="" type="checkbox"/>
P_Configure[1]	<input type="checkbox"/>
P_Dhcp[1]	<input type="checkbox"/>
P_Slaac[1]	<input checked="" type="checkbox"/>
P_Prefered[1]	<input checked="" type="checkbox"/>
P_Preference[2]	Medium(Default)
prefixLifetime[4]	0xFFFFFFFF
prefixAdvertised[1]	<input checked="" type="checkbox"/>
ExternalRouteFlags[1]	
R_preference[2]	Medium(Default)
ExternalRouteLifetime[4]	0xFFFFFFFF
ExternalRouteAdvertised[1]	<input type="checkbox"/>

THR_BrPrefixSync.Request(InstanceID = 0x00)

Command: THR_BrPrefixSyncRequest	
InstanceId[1]	0x00

2002::

Configure Prefix on DUT

Prefix 2: P_Prefix= 2002::/64, P_stable=0, P_default=1, P_slaac_preferred=1, P_on_Mesh=1

OK

DUT Shell Command

```
$ thr nwldata add slaac -p 2002::0 -len 64 -t 1
$thr sync nwldata
```

```
$ thr nwldata add slaac -p 2002::0 -len 64 -t 1
Success!
$ thr sync nwldata
Success!
```

DUT THCI Command

THR_BrPrefixAddEntry.Request(InstanceId = 0x00, prefixLength = 0x40, PrefixValue = 2002:0000:0000:0000:0000:0000:0000:0000, PrefixReservedFlags = 0x00, **P_Default = TRUE**, P_Configure = FALSE, P_Dhcp = FALSE, **P_Slaac = TRUE**, **P_Prefered = TRUE**, P_preference = Medium, prefixLifetime = 0x00000030, PrefixAdvertised = TRUE)

Command: THR_BrPrefixAddEntryRequest

InstanceId[1]	0x00
prefixLength[1]	0x40
PrefixValue[16]	2002:0000:0000:0000:0000:0000:0000:0000
PrefixFlagReserved[1]	0x00
PrefixFlags[1]	
P_On_Mesh[1]	<input checked="" type="checkbox"/>
P_Default[1]	<input checked="" type="checkbox"/>
P_Configure[1]	<input type="checkbox"/>
P_Dhcp[1]	<input type="checkbox"/>
P_Slaac[1]	<input checked="" type="checkbox"/>
P_Prefered[1]	<input checked="" type="checkbox"/>
P_Preference[2]	Medium(Default)
prefixLifetime[4]	0x00000030
prefixAdvertised[1]	<input checked="" type="checkbox"/>
ExternalRouteFlags[1]	
R_preference[2]	Medium(Default)
ExternalRouteLifetime[4]	0xFFFFFFFF
ExternalRouteAdvertised[1]	<input type="checkbox"/>

THR_BrPrefixSync.Request(InstanceID = 0x00)

Command: THR_BrPrefixSyncRequest

InstanceId[1]	0x00
---------------	------

2003::

Configure Prefix on DUT

Prefix 3: P_Prefix= 2003::/64, P_stable=1, P_default=0, P_slaac_preferred=1, P_on_Mesh=1

OK

DUT Shell Command

```
$ thr nwdata add slaac -p 2003::0 -len 64 -t 60500
$thr sync nwdata
```

```
thr nwdata add slaac -p 2003::0 -len 64 -t 60500
Success!
$ thr sync nwdata
Success!
```

DUT THCI Command

THR_BrPrefixAddEntryRequest (InstanceId = 0, prefixLength = 40, PrefixValue = 2003:0000:0000:0000:0000:0000:0000:0000, P_On_Mesh = TRUE, P_Default = FALSE, P_Configure = FALSE, P_Dhcp = FALSE, P_Slaac = TRUE, P_Preferred = TRUE, P_Preference = Medium, prefixLifetime = 0xFFFFFFFF, prefixAdvertised = TRUE, R_preference = Medium, ExternalRouteLifetime = 0xFFFFFFFF, ExternalRouteAdvertised = FALSE)

Command: THR_BrPrefixAddEntryRequest	
InstanceId[1]	0x00
prefixLength[1]	0x40
PrefixValue[16]	2003:0000:0000:0000:0000:0000:0000:0000
PrefixFlagsReserved[1]	0x00
PrefixFlags[1]	
P_On_Mesh[1]	<input checked="" type="checkbox"/>
P_Default[1]	<input type="checkbox"/>
P_Configure[1]	<input type="checkbox"/>
P_Dhcp[1]	<input type="checkbox"/>
P_Slaac[1]	<input checked="" type="checkbox"/>
P_Preferred[1]	<input checked="" type="checkbox"/>
P_Preference[2]	Medium(Default)
prefixLifetime[4]	0xFFFFFFFF
prefixAdvertised[1]	<input checked="" type="checkbox"/>
ExternalRouteFlags[1]	
R_preference[2]	Medium(Default)
ExternalRouteLifetime[4]	0xFFFFFFFF
ExternalRouteAdvertised[1]	<input type="checkbox"/>

THR_BrPrefixSync.Request(InstanceId = 0x00)

Command: THR_BrPrefixSyncRequest	
InstanceId[1]	0x00

9.15. Provisioning URL

Clear existing value

DUT Shell Command

```
$ thr set provisioningurl
$ thr set provisioningurl
Success!
```

DUT THCI Command

THR_SetAttr.Request (InstanceId = 00, AttributeId = ProvisioningURL, Index = 00, AttrSize = 0)

Command: THR_SetAttrRequest	
InstanceId[1]	0x00
AttributeId[1]	ProvisioningURL
Index[1]	0x00
ProvisioningURL	
AttrSize[1]	0x00

www.threadgroup.org

Set Provisioning URL

Set Provisioning URL on DUT to www.threadgroup.org

OK

DUT Shell Command

```
$ thr set provisioningurl www.threadgroup.org
$ thr set provisioningurl www.threadgroup.org
Success!
```

DUT THCI Command

THR_SetAttr.Request (InstanceId = 00, AttributeId = **ProvisioningURL**, Index = 00, AttrSize = **0x13**, Value = www.threadgroup.org)

Command: THR_SetAttrRequest	
InstanceId[1]	0x00
AttributeId[1]	ProvisioningURL
Index[1]	0x00
ProvisioningURL	
AttrSize[1]	0x13
Value[19][1]	www.threadgroup.org

www.nxp.com/thread

Set Provisioning URL

Set Provisioning URL on DUT to other than www.threadgroup.org

OK

DUT Shell Command

```
$ thr set provisioningurl www.nxp.com/thread
$ thr set provisioningurl www.nxp.com/thread
Success!
```

DUT THCI Command

THR_SetAttr.Request (InstanceId = 00, AttributeId = **ProvisioningURL**, Index = 00, AttrSize = **0x12**, Value = www.nxp.com/thread)

Command: THR_SetAttrRequest	
InstanceId[1]	0x00
AttributeId[1]	ProvisioningURL
Index[1]	0x00
ProvisioningURL	
AttrSize[1]	0x12
Value[18][1]	www.nxp.com/thread

9.16. PSKd

PSKd is a device password set by the manufacturer. This can be a unique alphanumeric string printed on a product label.

PSKd required..!

Enter PSKd of device

threadjaketest

✓ OK

DUT Shell Command

```
$ thr get pskd
$ thr get pskd
pskd: threadjaketest
```

DUT THCI Command

THR_GetAttrRequest (InstanceId = 0, AttributeId = **Security_PSKd**, Index = 0x00)

Command: THR_GetAttrRequest

InstanceId[1]	0x00
AttributeId[1]	Security_PSKd
Index[1]	0x00

TX: THR_GetAttr.Request 02 CE 17 03 00 00 19 00 C3

RX: THR_GetAttr.Confirm 02 CF 17 14 00 00 19 00 0F 74 68 72 65 61 64 6A 70 61 6B 65 74 65 7 B7

Sync [1 byte] = 02

OpGroup [1 byte] = CF

OpCode [1 byte] = 17

Length [2 bytes] = 00 14

InstanceId [1 byte] = 00

AttributeId [1 byte] = 19 (Security_PSKd)

Index [1 byte] = 00

Status [1 byte] = 00 (Success)

AttrSize [1 byte] = 0F

Security_PSKd [15 bytes] = 74 68 72 65 61 64 6A 70 61 6B 65 74 65 73 74 threadjaketest

CRC [1 byte] = B7



PSKd required..!

Enter PSKd of device

threadjaketest

✓ OK

Copy the alphanumeric PSKd value from Test Tool into the Thread Harness window pop-up, then press OK

9.17. Random Extended MAC Address

Known as the Random IEEE 802.15.4 Extended Address, it consists of 8 bytes randomly generated using TRNG. Its purpose is to guarantee unique device identity in the same network.

DUT Random Extended MAC Address Required !

Enter DUT Random Extended MAC Address.[Pass Hex Value]

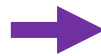
✓ OK

DUT Shell Command

```
$ thr get randomaddr
```

DUT Random Extended MAC Address Required !

Enter DUT Random Extended MAC Address.[Pass Hex Value]




```
$ thr get randomaddr
randomaddr: 0x0F1C1FF31C91B105
```

✓ OK

Copy the randomaddr value from the shell terminal; paste it in the Thread Harness window pop-up, then press OK

DUT THCI Command

```
THR_GetAttrRequest (InstanceId = 0, AttributeId = RandomAddress, Index = 0x00)
```

9.18. RLOC Address

The Mesh Local Address ML16 embeds the Router Locator or RLOC, because of its nature this is an address intended to be used internally by the Thread Stack.

RLOC Address

Enter DUT RLOC Address.

✓ OK

DUT Shell Command

```
$ ifconfig
```



```
$ ifconfig
Interface 0: 6LoWPAN
Link local address <LL64>: fe80::156:a19e:f1b1:252c
Mesh local address <ML16>: fd00:db8::ff:fe00:0
Mesh local address <ML64>: fd00:db8::adde:2802:3b1:db8
Global address: 2001::8100:f499:4c33:1ef6
Link local all Thread Nodes<MCast>: ff32:40:fd00:db8::1
Realm local all Thread Nodes<MCast>: ff33:40:fd00:db8::1
```

RLOC Address

Enter DUT RLOC Address.

✓ OK

Copy the ML16 address from the shell terminal; paste it in the Thread Harness window pop-up, then click OK

DUT THCI Command

THR_GetThreadIpAddrRequest (InstanceId = 0, AddressType = **RLOC**)

Command: THR_GetThreadIpAddrRequest	
InstanceId[1]	0x00
AddressType[1]	RLOC
	Link_Local_64
	ML_EID
	RLOC
	Global

9.19. Short Address

2 bytes assigned specifically with the Routing Locator (RLOC) to facilitate node location within the thread mesh.

Short Address

Enter DUT ShortAddress Address.

✓ OK

DUT Shell Command

\$ thr get shortaddr

```
$ thr get shortaddr
shortaddr: 0x0400
```

DUT THCI Command

THR_GetAttrRequest (InstanceId = 0, AttributeId = **ShortAddress**, Index = 00)

Command: THR_GetAttrRequest	
InstanceId[1]	0x00
AttributeId[1]	ShortAddress
Index[1]	0x00

9.20. Start DUT – Joiner Router

Start DUT..!

Click OK then start the Joiner/DUT within 2 mins

✓ OK

DUT Shell Commands

```
$ thr set iscommissioned 0
$ thr set provisioningurl
$ thr join
```

```
$ thr set iscommissioned 0
Success!
$ thr set provisioningurl
Success!
$ thr join
Joining network...
```

DUT THCI Command

THR_SetAttr.Request (Instanceid = 0, Attributeid = **IsDevCommissioned**, Index = 0, AttrSize = 1, Value = **FALSE**)

Command: THR_SetAttrRequest	
Instanceid[1]	0x00
Attributeid[1]	IsDevCommissioned
Index[1]	0x00
IsDevCommissioned	
AttrSize[1]	0x01
Value[1]	<input type="checkbox"/>

THR_SetAttr.Request (Instanceid = 0, Attributeid = **ProvisioningURL**, Index = 0, AttrSize = 0)

Command: THR_SetAttrRequest	
Instanceid[1]	0x00
Attributeid[1]	ProvisioningURL
Index[1]	0x00
ProvisioningURL	
AttrSize[1]	0x00

THR_Join.Request (InstanceID = 0, discoveryMethod = **gUseThreadDiscovery_c**)

Command: THR_JoinRequest	
InstanceID[1]	0x00
discoveryMethod[1]	gUseThreadDiscovery_c
	gUseMACBeacon_c
	gUseThreadDiscovery_c

9.21. Start DUT – Router Commissioner

The DUT will join the network knowing the credential to then request to the leader to become the active commissioner.

Start DUT as Router..!

Boot Device as Router and configure it as commissioner

☒ OK

DUT Shell Commands

```
$ thr set iscommissioned 1
$ thr set provisioningurl
$ thr join
```

```
$ thr set iscommissioned 1
Success!
$ thr set provisioningurl
Success!
$ thr join
```

Wait until the device is a REED....

```
Joining network...
$
Joining a Thread network...
Attached to network with PAN ID: 0xface
[]
```

```
$thr commissioner start
```



```
$ thr commissioner start
<Local> Commissioner Started
```

DUT THCI Command

THR_SetAttr.Request (InstanceId = 0, AttributeId = **IsDevCommissioned**, Index = 0, AttrSize = 1, Value = **TRUE**)

Command: THR_SetAttrRequest

InstanceId[1]	0x00
AttributeId[1]	IsDevCommissioned
Index[1]	0x00
IsDevCommissioned	
AttrSize[1]	0x01
Value[1]	<input checked="" type="checkbox"/>

THR_SetAttr.Request (InstanceId = 0, AttributeId = **ProvisioningURL**, Index = 0, AttrSize = 0)

Command: THR_SetAttrRequest

InstanceId[1]	0x00
AttributeId[1]	ProvisioningURL
Index[1]	0x00
ProvisioningURL	
AttrSize[1]	0x00

THR_Join.Request (InstanceID = 0, discoveryMethod = **gUseThreadDiscovery_c**)

Command: THR_JoinRequest

InstanceID[1]	0x00
discoveryMethod[1]	gUseThreadDiscovery_c

Wait until the device is a REED....

```

RX: THR_EventGeneral_Confirm 02 CF 54 05 00 00 0C 00 00 00 92
  Sync      [1 byte] = 02
  OpGroup   [1 byte] = CF
  OpCode    [1 byte] = 54
  Length    [2 bytes] = 00 05
  InstanceId [1 byte] = 00
  EventStatus [2 bytes] = 00 0C (Device is REED)
  DataSize   [2 bytes] = 00 00
  CRC       [1 byte] = 92

```

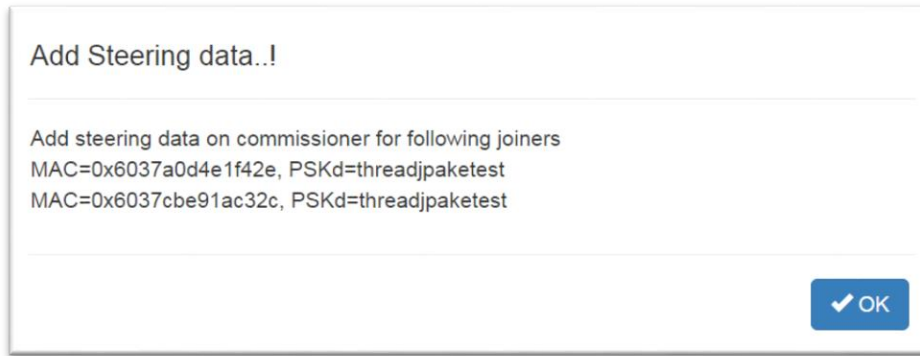
MESHCOPI_StartCommissioner.Request (InstanceID = 0)

Command: MESHCOPI_StartCommissionerRequest

InstanceId[1]	0x00
---------------	------

9.22. Steering

Configure joiners to be accepted by the Thread Commissioner. To add a new joiner use the add command with the joiner PSKd and MAC address information, then call sync steering to add the joiner to the bloom filter.



DUT Shell Commands

```
$ thr joiner add threadjpacketest 0x006037a0d4e1f42e
$ thr joiner add threadjpacketest 0x006037cbe91ac32c
$ thr sync steering
```

DUT THCI Commands

MESHCOPI_AddExpectedJoiner.Request (InstanceId = 0, Selected = TRUE, EuiType = LongEUI, LongEUI = 0x006037cbe91ac32c, PSKdSize = 0x0F, PSKd = threadjpacketest)

Command: MESHCOPI_AddExpectedJoinerRequest

InstanceId[1]	0x00
Selected[1]	<input checked="" type="checkbox"/>
EuiType[1]	LongEUI
LongEUI[8]	0x006037CBE91AC32C
PSKdSize[1]	0x0F
PSKd[15][1]	threadjpacketest

MESHCOPI_AddExpectedJoiner.Request (InstanceId = 0, Selected = TRUE, EuiType = LongEUI, LongEUI = 0x006037a0d4e1f42e, PSKdSize = 0x0F, PSKd = threadjpacketest)

Command: MESHCOPI_AddExpectedJoinerRequest

InstanceId[1]	0x00
Selected[1]	<input checked="" type="checkbox"/>
EuiType[1]	LongEUI
LongEUI[8]	0x006037A0D4E1F42E
PSKdSize[1]	0x0F
PSKd[15][1]	threadjpacketest

MESHCOPI_SyncSteeringData.Request (InstanceId = 0, EuiMask = ExpectedJoiners)

Command: MESHCOPI_SyncSteeringDataRequest

InstanceId[1]	0x00
EuiMask[1]	ExpectedJoiners

9.23. GUA Address

GUA Address

Enter DUT GUA 2001:: Prefix Address.

2001:0000:0000:0000:D4BF:DFDE:81B9:5669

✓ OK

DUT Shell Commands

```
$ ifconfig
```

DUT THCI Commands

```
NWKU_IfConfigAll.Request(Instanceld = 0)
```

9.24. Security Key Switch Disable

Some test cases test that MLE packets are secured with a key index incremented by 1. Setting the security key switch to 0 allows this change to occur within the Thread Harness timings.

DUT Shell Command

```
$ thr set keyswitchguard 0
```

```
$ thr set keyswitchguard 0
Success!
```

DUT THCI Command

```
THR_SetAttrRequest (Instanceld = 0, AttributeId = Security_KeySwitchGuardTime, Index = 0x00, AttrSize = 0x04, PrefixData = 0x00, 0x00, 0x00, 0x00)
```

Command: THR_SetAttrRequest	
Instanceld[1]	0x00
AttributeId[1]	Security_KeySwitchGuardTime
Index[1]	0x00
Security_KeySwitchGuardTime	
AttrSize[1]	0x04
Value[4]	0x00000000

9.25. MGMT_SET_PANID

Set MGMT_PANID_QUERY.qry !

Send MGMT_PANID_QUERY to FF33:0040:FD00:0DB8::1

✓ OK

```
$ thr mgmt query FF33:0040:FD00:0DB8::1
Success!
```

DUT THCI Command

```
$ thr mgmt. query FF33:0040:FD00:0DB8::1
```

DUT THCI Command

Revision history

MESHCOPI_MgmtSendPanIdQuery.Request (ThrInstanceID = 0, ScanChannelMask = 00 20 08 00, PanId = 00 00, IPAddress = FF 33 00 40 FD 00 0D B8 00 00 00 00 00 00 00 01)

Command: MESHCOPI_MgmtSendPanIdQueryRequest	
ThrInstanceID[1]	0x00
ScanChannelMask[4]	0x00200800
PanId[2]	0x0000
IPAddress[16]	FF33:0040:FD00:0DB8:0000:0000:0000:0001

9.26. MLPrefix

The Mesh Local Prefix MLP is consistent for a Thread Network, its value is generated per IPv6 RFC4193. To be able to correlate values with the Thread Test Harness, the MLP needs to be hard-coded

DUT THCI Command

```
$ thr set mlprefix 0xFD000DB800000000
```

```
$ thr set mlprefix 0xFD000DB800000000  
Success!
```

DUT THCI Command

THR_SetAttrRequest (InstanceID = 0, AttributeID = **MLPrefix**, Index = 0x00, AttrSize = **0x11**, PrefixData = **0xFD,0x00,0x0D,0xB8,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00**, Prefix Length = **0x10**)

Command: THR_SetAttrRequest	
InstanceID[1]	0x00
AttributeID[1]	MLPrefix
Index[1]	0x00
MLPrefix	
AttrSize[1]	0x11
Value	
PrefixData[16][1]	0xFD,0x00,0x0D,0xB8,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
PrefixLength[1]	0x10

10. Filling the Thread Certification application document

Download the zip file AN5428SW.zip from www.nxp.com for an example of the NXP Kinetis Thread software library filling information.

11. Revision history

Table 2. Revision history

Revision number	Date	Substantive changes
0	04/2017	Initial release

How to Reach Us:

Home Page:

nxp.com

Web Support:

nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including "typicals," must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address:

nxp.com/SalesTermsandConditions.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, and Tower, are trademarks of NXP B.V. All other product or service names are the property of their respective owners.

ARM, the ARM Powered logo, and Cortex are registered trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved.

© 2017 NXP B.V.

Document Number: AN5428

Rev. 0

04/2017

