Document identifier: UM11324
Rev. 1, 10 March 2020

OpenThread Demo Applications

User's Guide

Contents

Chapter 1
Introduction

This document is an overview of the deployment and operation of wireless network applications created using the OpenThread network protocol stack running on JN5189 platform.

This document is also a user's guide for the sample applications included in the JN5189 OpenThread SDK.

1.1 Audience

This document is for firmware and system developers who create OpenThread-enabled products. The document also provides high level descriptions of OpenThread application scenarios which can be deployed on JN5189 development boards.

Chapter 2
Thread Stack and Technology Overview

Thread is an IPv6 wireless mesh networking protocol for integration into consumer products. At the link layer, Thread is based on IEEE® 802.15.4 MAC and PHY operating in the 2.4 GHz open radio frequency band.

Thread networks provide a mesh communication fabric for seamless user-to-device and device-to-device application interaction scenarios for the connected home. The technology also provides IP connectivity and addressability to products such as coin-cell battery operated sensors and simple appliances and actuators making them accessible within the mesh network, the home area IP infrastructure, and Internet.

Compared to other wireless protocols, Thread is focused on providing low latency, reliability, redundancy, security, simple and autonomous network configuration, and low-power consumption.

2.1 Thread Device Types

The image below shows the different categories of connected home devices and their respective power profiles.

Figure 1. Connected home devices

The Thread applications provided with the JN5189 OpenThread stack contain preconfigured stack and application layer configurations organized around the major categories of Thread device types and capabilities reflecting the power and complexity requirements shown in the image above.

Chapter 3
OpenThread Stack Applications Overview

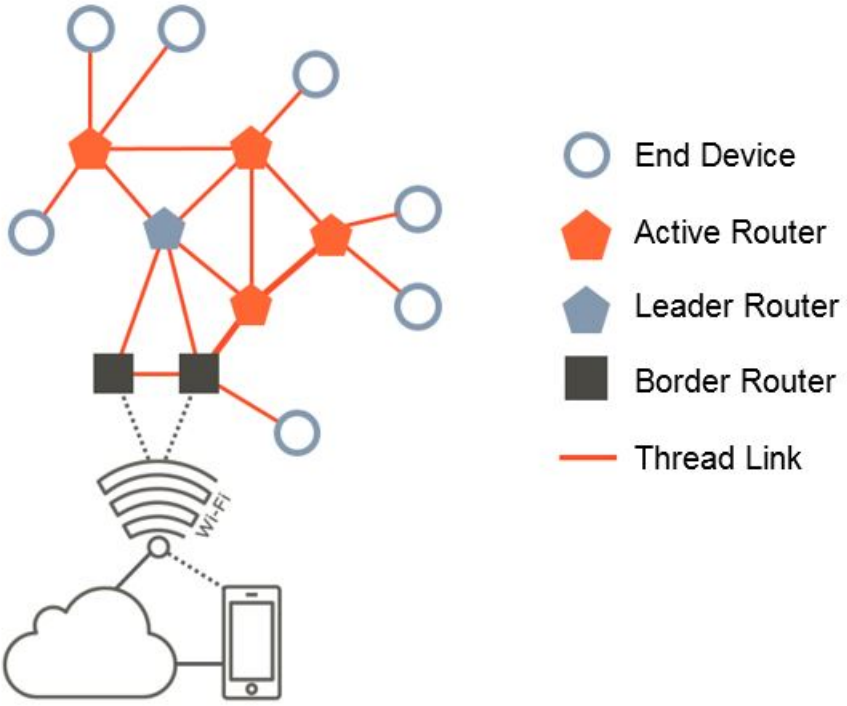


Figure 2. Typical Thread network devices and roles

3.1 Thread router eligible devices

A Router Eligible Device is a node which initially joins the network as an End Device, but can adaptively become a mesh Router. Such a device may also have capabilities to initialize, create, and bootstrap a new Thread Network for the user or a management entity.

The distinct roles this device can play during network operation are:

- Router Eligible End Device(REED)
- Active Router, also referred to as a Promoted Router, or a Router

The two roles are dynamic and can adaptively change from the End Device role when the node does not actively participate in routing, does not forward data transmissions, and cannot accept to be a “parent” for other End Devices to the Active Router role which actively performs all these functionalities.

Typically up to 16-32 Router Eligible Devices on a Thread Network can automatically activate the routing functionality and request a Router Identifier (Router ID). When a Router ID is assigned, the devices become Active Routers.

A Router Eligible Device can also assume a Leader role, acting as a point of decision for Router ID assignments and dissemination of routing and network information for a network partition. A Thread network starts as a single network partition. However, multiple distinct partitions can form if routing segments of the same network become disconnected. The partitions are still part of the same Thread network, and can share network credentials and merge back to a single partition if inter-connectivity between the routing segments is re-established. Each partition has a single Leader router. The Leader role is transitional and can be taken over by any other Active Router in the partition if the current Leader becomes unavailable.

The application example for a Router Eligible Device is a template for developing mains-powered and always-on Thread devices such as smart power outlets, appliances, network range extenders, and control panels.

Other mains-powered devices such as smart light bulbs or lamps are not restricted by power constraints and can act as routers. However, because users can disconnect power from light bulbs using room power switches, the network routing capacity can become diminished when onlylighting fixtures act as routers.

When deploying Thread devices and networks, a base routing infrastructure consisting of always-on Router Eligible Devices is recommended.

3.2 Thread end devices

An End Device is a node which does not have routing eligibility or network creation capabilities. The main characteristic of an End Device is that it always communicates to the rest of the network by having data relayed to and from an Active Router which becomes a "parent" for the End Device. The End Device at its turn acts as the "child" of the Active Router and expects the Active Router to act on its behalf for forwarding data transfers.

An End Device can be battery-powered by using a high-capacity battery. However, the end device is usually a mains powered device which is not expected to be always-on or which otherwise cannot fulfil the Router Eligible role described above due to its limited memory or processing resources available.

3.3 Thread low-power end devices

The Low-power End Device (also called Sleepy End Device (SED)) is an End Device which is meant to remain in a low-power, dormant state for the majority of its lifetime. It is usually battery-powered with a limited battery capacity.

The Low-power End Device in most use cases becomes active for communication only for very short periods, either “waking up” automatically based on a predefined periodic timer or by means of user interaction.

The device can become active, for example when a user presses a button and generates a wake up interrupt. As soon as they become active, Low-power End Devices can poll their parent Active Router for any data addressed to them and transmit any outgoing data they have to send. The devices should return as quickly as it is allowed by the application to the previous inactive radio state to preserve their battery.

Other devices communicating with a Low-power End Device must assume there may be some latency when sending data towards the end device. On the other hand, an interrupt-based wake up and associated transmissions which originate from the Low-power End Device can be very quick. These devices make good candidates for light switches, remote controls, or sensors.

3.4 Thread border routers

A Thread Network is a native IPv6 subnet, where each node has assigned one or more IPv6 addresses as end points and where IP protocols are used for all communications and network management.

Border Routers are Thread devices which have at least another IP interface on board besides the Thread radio transceiver. As a result, Border Routers can internally forward data at the IP layer from the Thread network to the other network interfaces over IP subnet boundaries. This allows end-to-end IP connectivity between Thread devices and application running on computers and smartphones, other devices on home local area Wi-Fi networks and even to an Internet server or a Virtual Private Network (VPN).

The Border Router example templates are recommended for devices meant to have a Thread radio on board as well as Wi-Fi or Ethernet capability and allow inter-connection to the external IP networks and the Internet.

3.5 Demo Applications Overview

This chapter shows an overview of the features enabled by default within the demo applications configurations.

Table 1. Demo Applications overview

Example App	Available board functionalities
border_router (template for **products where a UDOO Neo SBC device** is running the IP stack on multiple interfaces**, including Thread,** using a **JN5189 device as radio interface**)	**CoAP**: led, temp, data sink **UART**: shell **USB**: N/A **Lib capability**: leader, router, reed, rx on ed, nd **Commission**: auto-start collapsed commissioner on leader
router_eligible_device (template for **mains powered, always-on products driven entirely by JN5189**: security control panels, standalone sensor hubs, range extenders, smart plugs, some thermostats, wall light switches, some light fixtures, some appliances)	**CoAP:** led, temp, data sink **UART:** shell **USB:** N/A **Commission:** auto-start collapsed commissioner on leader **Lib capability:** leader, router, reed
end_device (template for **mains powered** or **high-cpapacity battery products driven entirely by JN5189** which are **NOT intended to be always-on**: light fixtures, appliances, some door locks, some thermostats, some resource constrained devices)	**CoAP:** led, temp, data sink **UART:** shell **USB:** N/A **Other:** rx on ed defaults **Lib capability:** rx on ed
low_power_end_device (template for **low-capacity battery JN5189 products**: sensors, remote controls, door locks)	**CoAP:** temp, data sink **UART:** N/A **USB:** N/A **LP:** LP mode 3 **Lib capability:** sed

• **CoAP: led** -- The app has a CoAP callback which controls the LEDs on the board.

• **CoAP: temp** -- The app has a CoAP API which sends the chip temperature over the air.

• **CoAP: data sink --** The app has a CoAP API which advertises via multicast the node as a sensor data data sink, enabled LED and temperature data to be then sent unicast to the node.

• **UART: shell --** Shell is enabled over UART (via OpenSDA chip on DK6 Carrier Main Board or via module pins)

• **Lib Capability: leader --** Can become a Leader at runtime

• **Lib Capability: router --** Can become a Router at runtime

• **Lib Capability: reed --** Can act as a Router Eligible End Device at runtime.

• **Lib Capability: sed --** Includes low-power end device capabilities (ability to enter low-power)

• **Lib Capability: rx on ed --** Includes the MAC rx-on mode for an end device at runtime

• **Lib Capability: nd --** Includes IPv6 Ethernet/Wi-Fi Neighbor Discovery features

Chapter 4
Deploying OpenThread Stack and Applications Software

The JN5189 OpenThread software package includes the components necessary to begin Thread wireless mesh network application development on the JN5189 platform. These components include:

- OpenThread Internet Protocol (IP) based mesh network software libraries
- Example consisting in demo application firmware projects corresponding to different Thread node or device categories
- Precompiled examples firmware

The JN5189 OpenThread software package includes the JN5189 peripheral drivers, platform startup code, other generic JN5189 platform software as provided by the SDK.

4.1 Downloading the JN5189/K32W061 Connectivity Software

To download the Thread Stack software, perform the following steps:

- In a web browser, access the MCUXpresso home page mcuxpresso.nxp.com.
- Choose Select Development Board menu option
- Perform the user authentication
- Use the Search by Name for JN5189DK development board
- Press Build MCUXpresso SDK button
- Select the Toolchain/IDE and Host OS preferred options
- Press Request SDK button
- Download the generated SDK package
- Copy the package into C:\NXP**SDK_2.6.0_JN5189DK6** folder **for example.**

4.2 Supported integrated development environments

The IAR® Embedded Workbench for Arm® (IAR EWARM) Integrated Development Environment (IDE) and MCUXpresso IDE and toolchain are required to customize, compile, and debug the example demo applications included with the JN5189 OpenThread SDK. IAR EWARM or MCUX are also required to develop similar applications using the given examples as templates.

To download an evaluation version or to acquire a full version of IAR EWARM, visit the web page https://www.iar.com/iar-embedded-workbench/partners/nxp.

To download the MCUXpresso Integrated Development Environment (IDE), see the NXP page: MCUX IDE download.

4.3 Prerequisites for deploying the OpenThread projects

Before building an OpenThread project from the IAR® Embedded Workbench for Arm® (IAR EWARM) or from the MCUXpresso IDE, please install the following toolchains and utilities:

- GNU ARM Embedded Toolchain.
- Edit the System Path environment variable and add the path to the GNU ARM Embedded Toolchain.

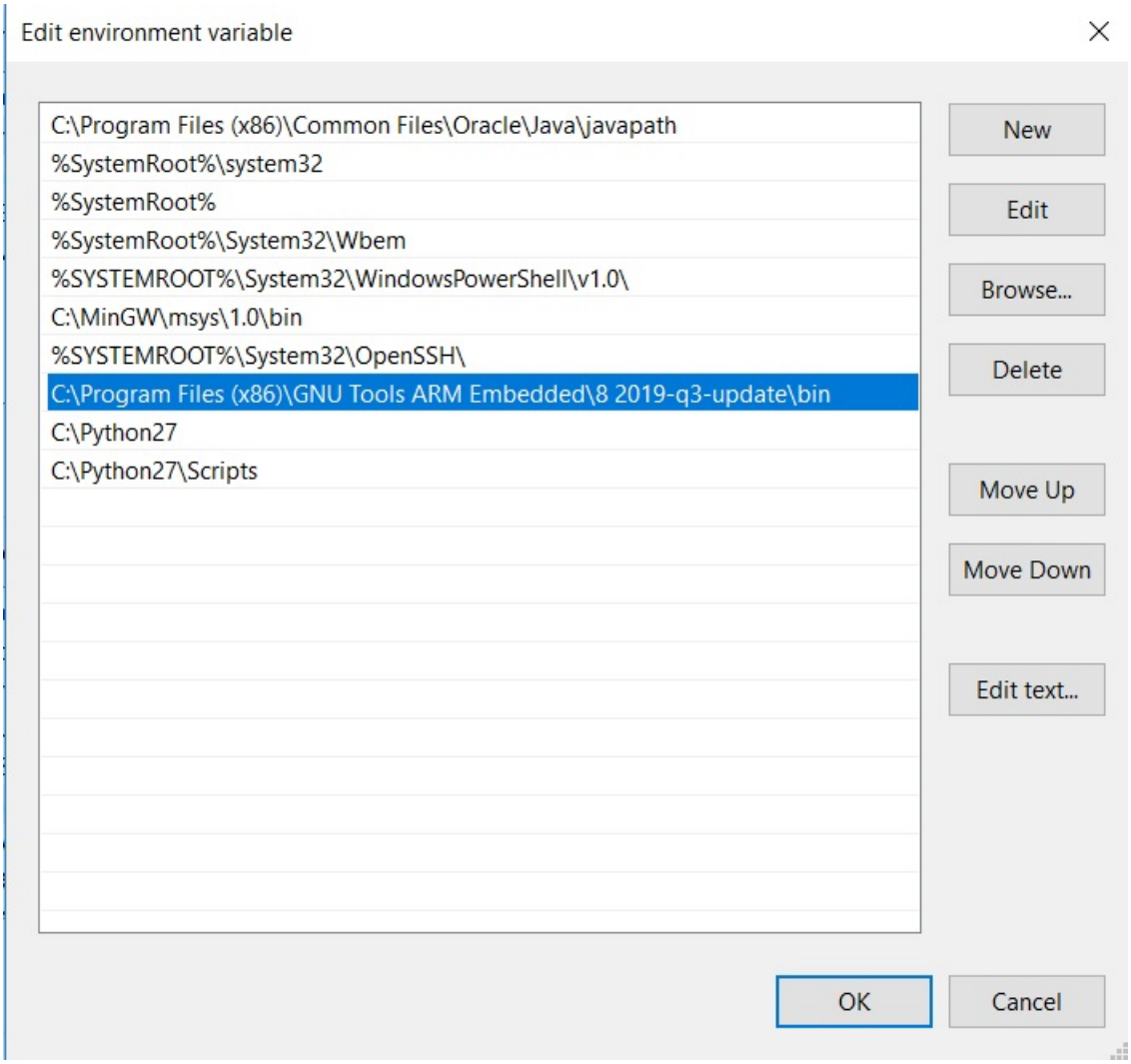


Figure 3. Edit Environment Variable

- Install Python 2.7 and the pycryptodome module
    1. Download Python 2.7 and install it
    2. Open a CMD prompt
    3. > pip list
    4. > python -m pip install --upgrade pip
    5. > pip install pycryptodome
- Edit the System Path environment variable and add the paths to the Python and Python scripts folders.

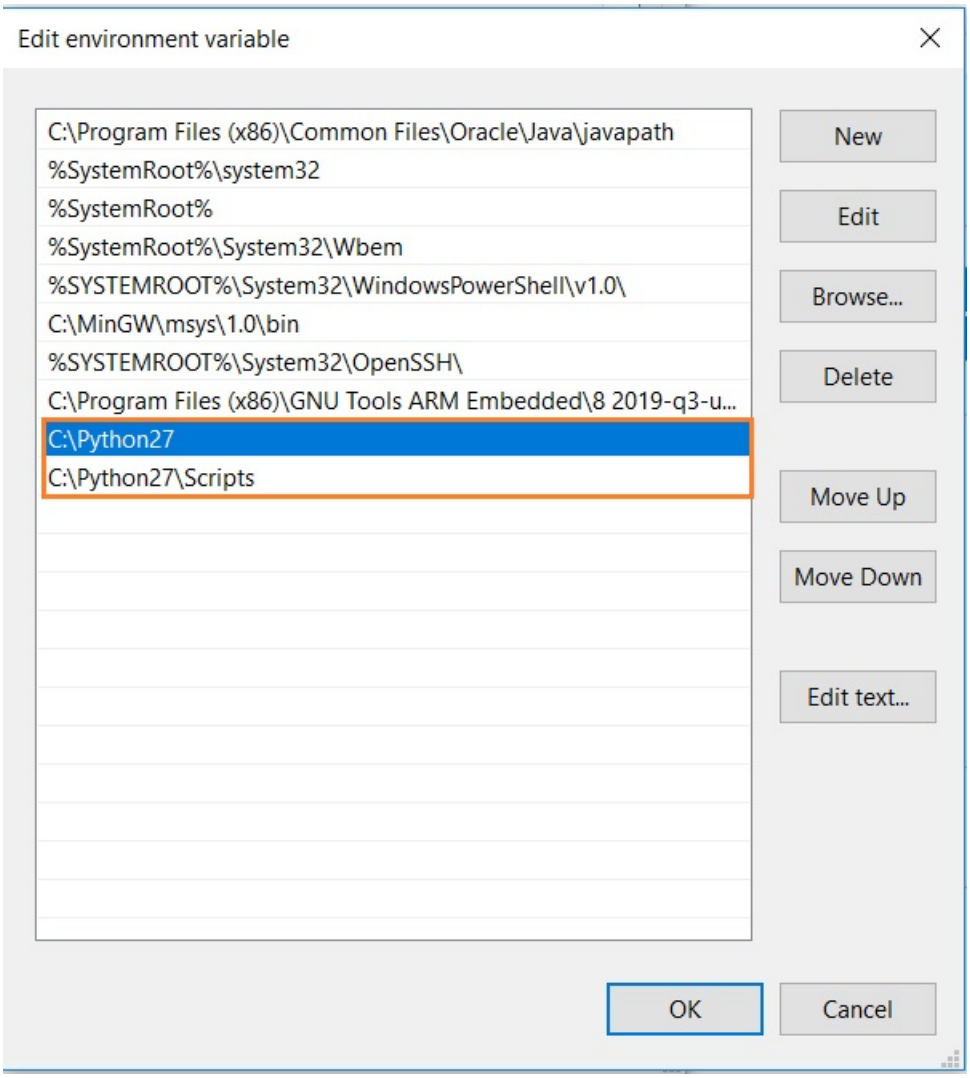


Figure 4. Python System Path Variables

Chapter 5
OpenThread Hardware Platforms

The OpenThread Stack version 1.1.1 supports the following platform which have radio frequency transceiver capabilities compatible with the IEEE 802.15.4 standard used by Thread:

- JN5189 Wireless MCU family - integrates an Arm Cortex ® -M4 core with an IEEE 802.15.4 transceiver

Chapter 6
Deploying Applications with IAR EWARM

6.1 Project launch files

Each of the example demo projects provide a set of separate IAR EWARM launch files. The launch files consist in:

- <*.eww> workspace configuration file
- <*.ewp> project configuration file
- <*.ewd> debug configuration file

The launch files for each of the configurations are contained in the folder structure starting at the boards subfolder in the root of the JN5189 OpenThread SDK installation.

For instance, to access the launch files for the **Thread Router Eligible End Device** example to deploy on the **JN5189DK6** development platform, navigate to the following subfolder:

<Connectivity_Software_Installation, for example, C:\NXP\SDK_2.6.0_JN5189DK6>

```
\boards\jn5189dk6\wireless_examples\openthread\reed\bm\iar
```

Each configuration subfolder has the following generic structure:

<Connectivity_Software_Installation, for example, C:\NXP\SDK_2.6.0_JN5189DK6>:

```
\boards\<board>\wireless_examples\openthread\ <example application> \<RTOS>\ <toolchain>
```

Each configuration subfolder contains the set of workspace, project, and debug configuration files, for instance:

- reed_bm.ewd
- reed_bm.ewp
- reed_bm.eww

6.2 Opening a workspace

To open an example OpenThread demo application into IAR EWARM for development: double-click or press Enter to launch the **<*.eww>** workspace file - such as **reed_bm.eww**

IAR EWARM opens and display the projects referenced in the workspace file.

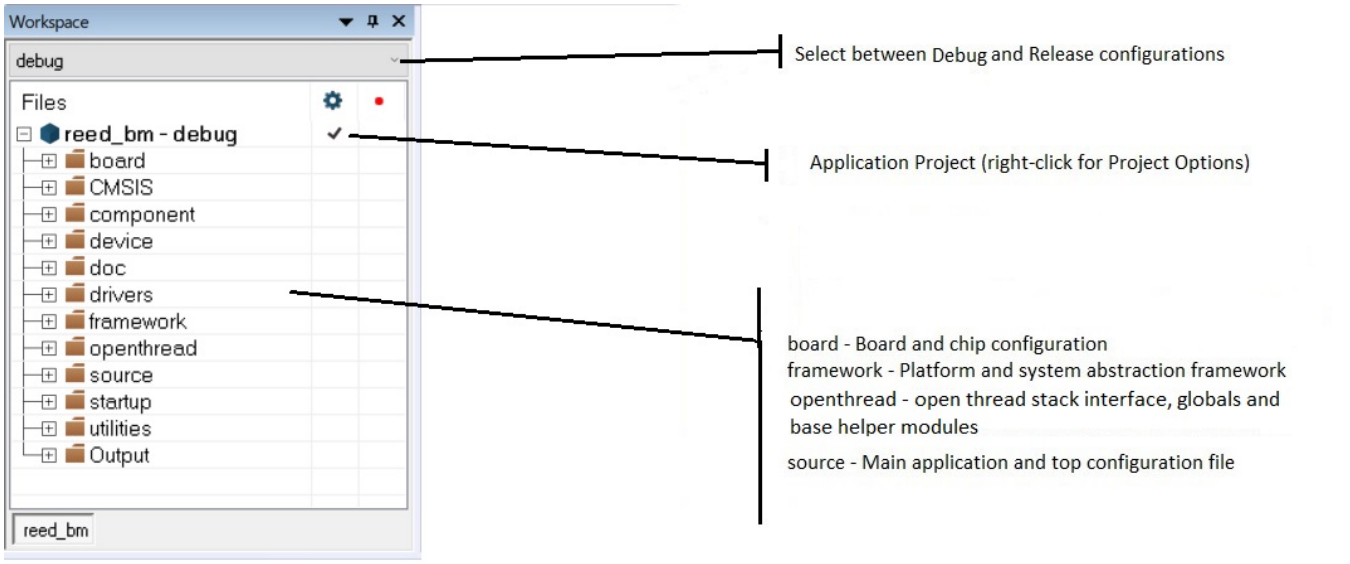


Figure 5. IAR EWARM workspace for REED

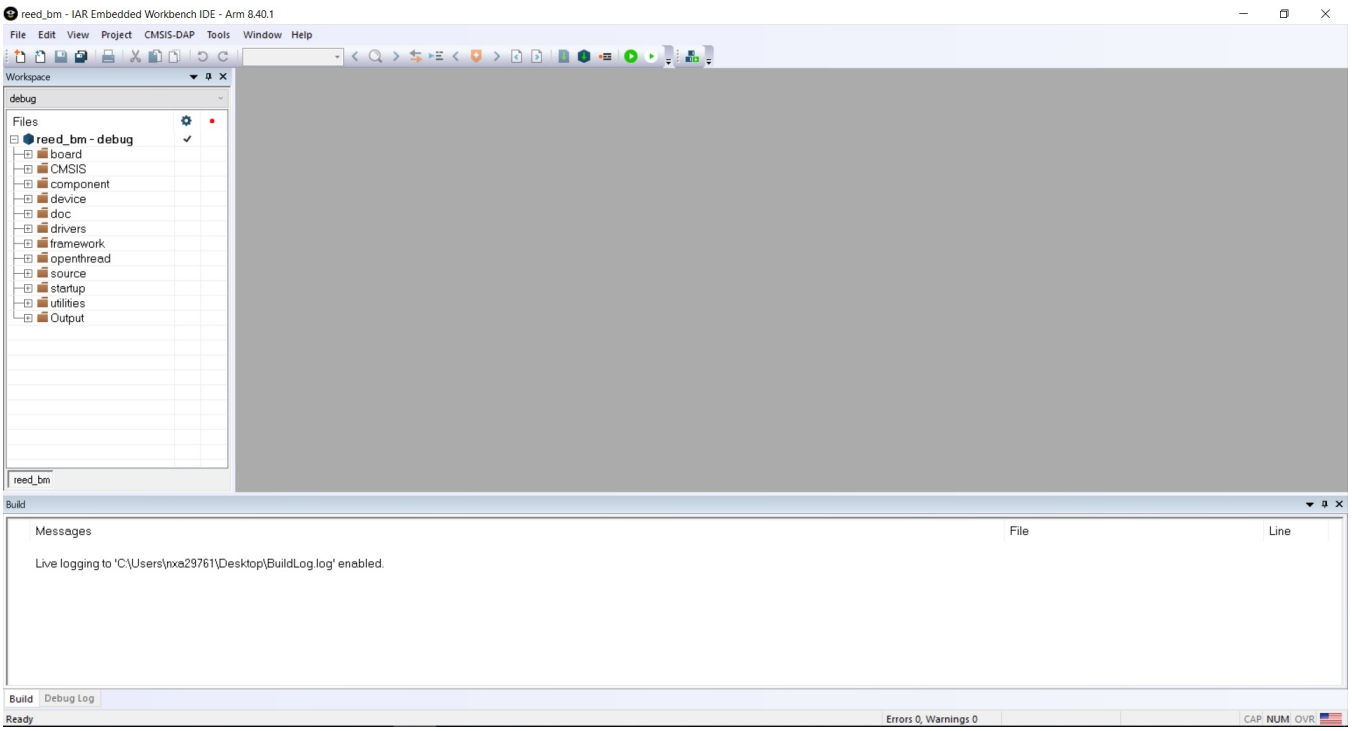


Figure 6. IAR EWARM workspace

6.3 Project configurations

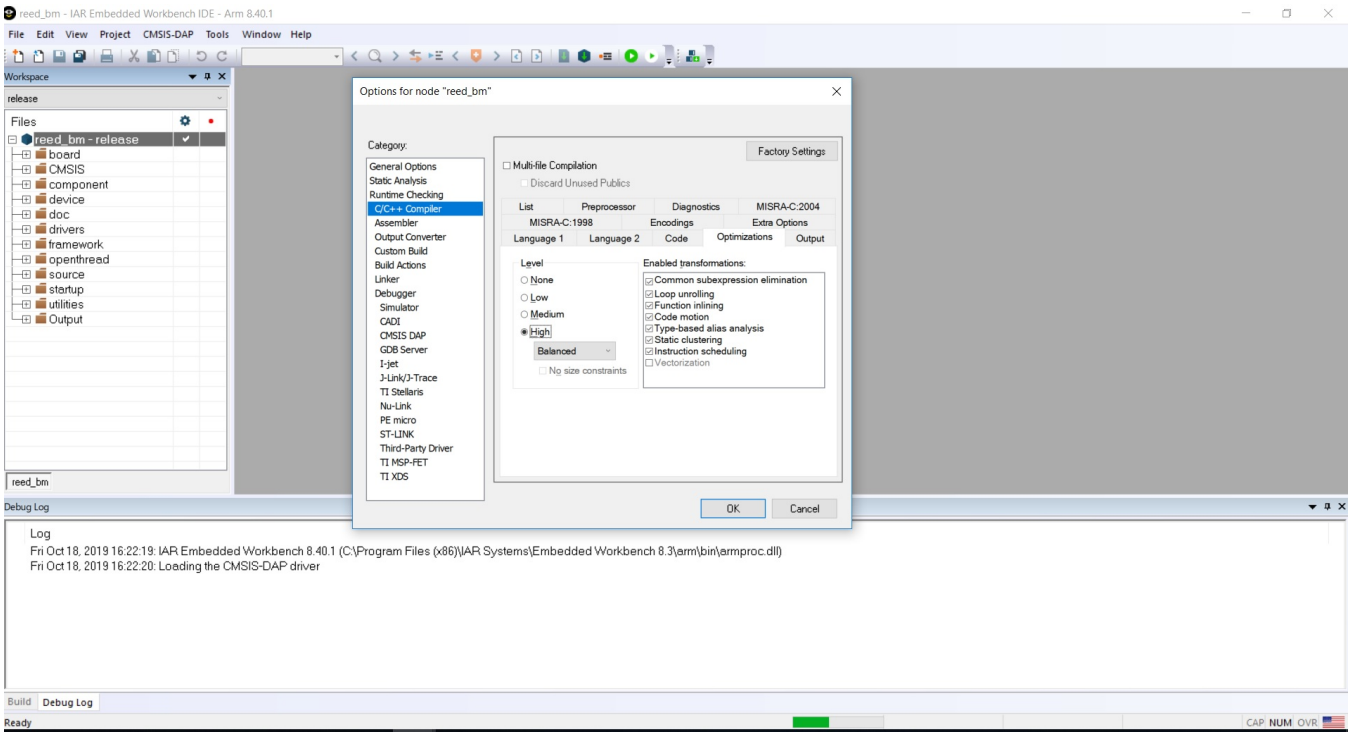


Figure 7. EWARM Release Configuration Options

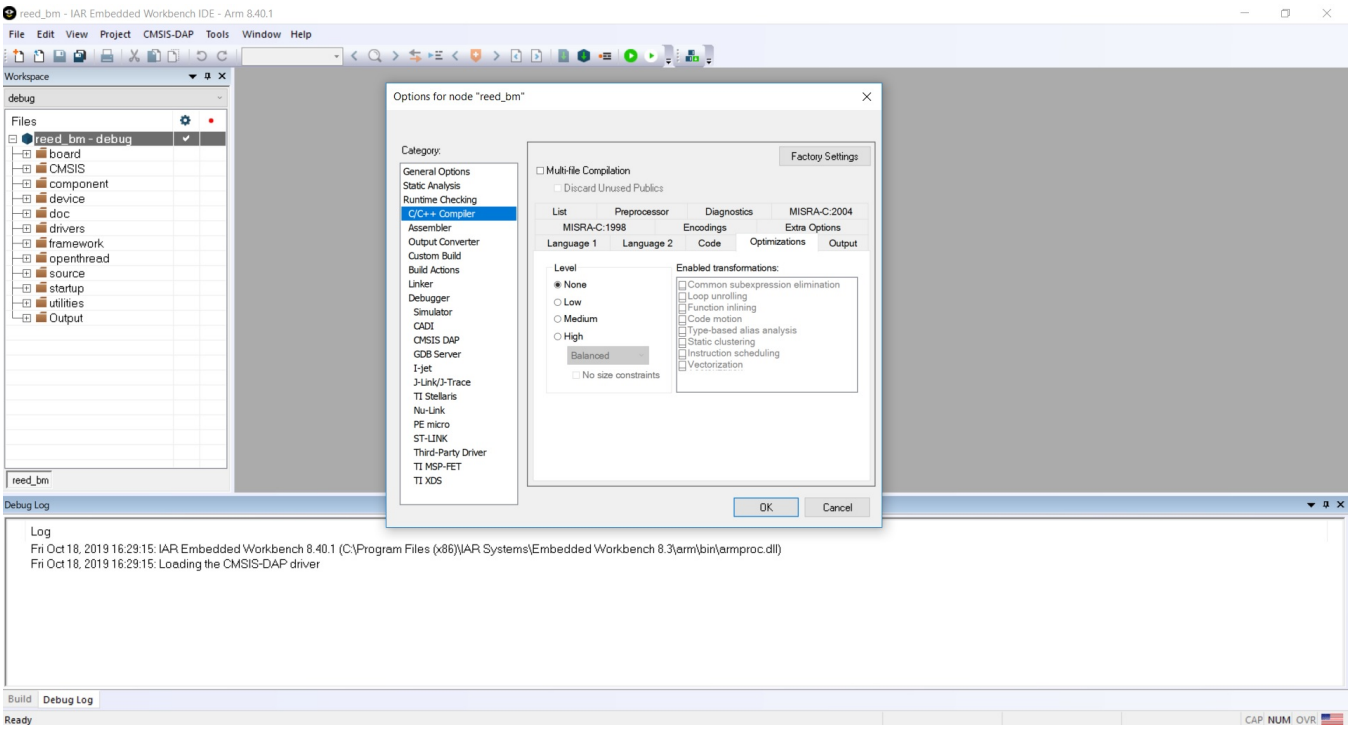


Figure 8. EWARM Debug Configuration Options

6.4 Building the application executable

The Bare Metal IAR project of the OpenThread Router Eligible End Device is available in:

```
<install_path>\boards \jn5189dk6\wireless_examples\openthread\reed\bm\iar
```

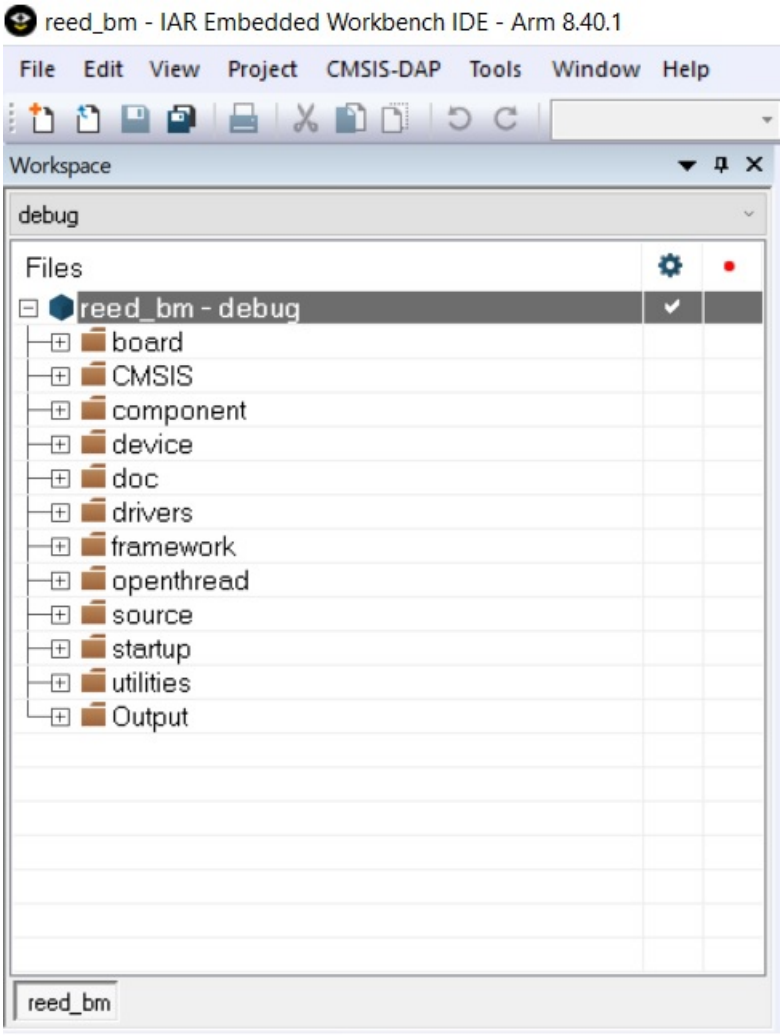


Figure 9. REED BM IAR Project

To build the application executable firmware: in the EWARM Workspace navigator, right click the application name and select **Make**.

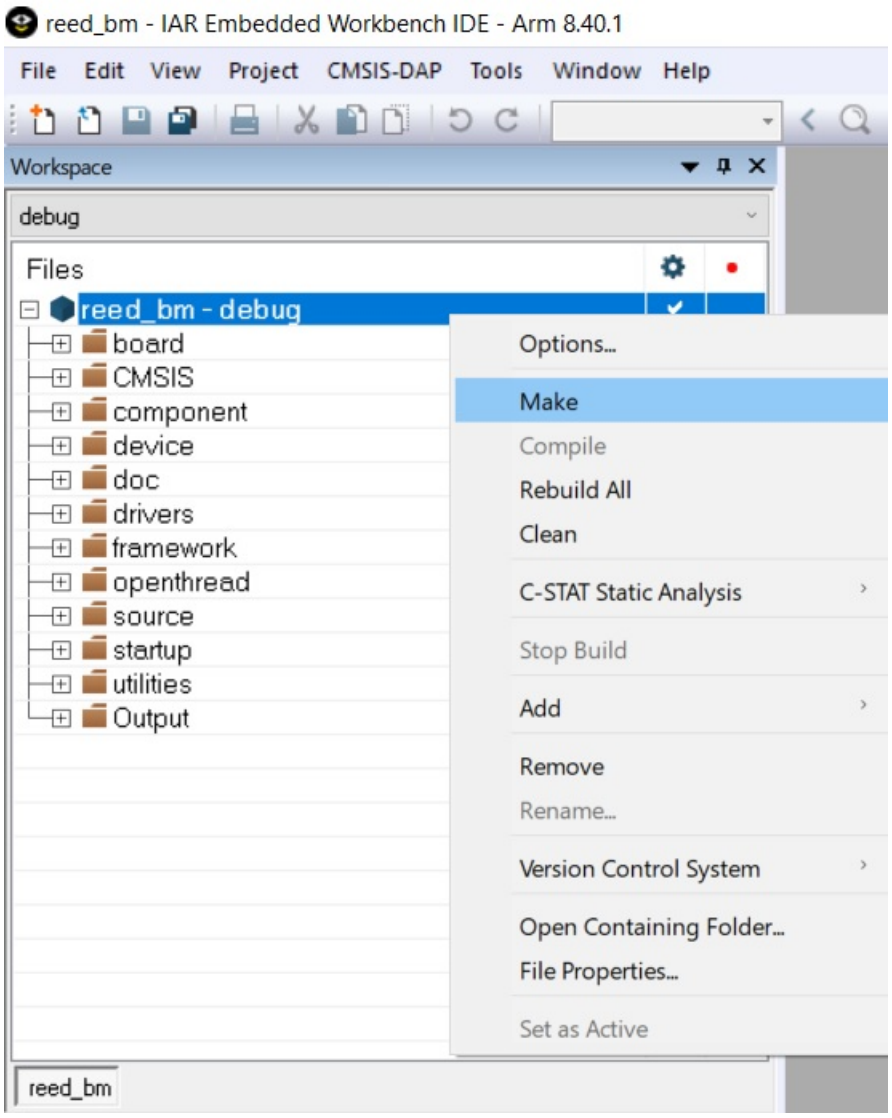


Figure 10. IAR EWARM Building Firmware

Build the project and observe the .bin file generated in `<install_path\boards\jn5189dk6\wireless_examples\openthread\reed\bm\iar\debug` for this particular project configuration, after a successful build.

6.5 Deploying the firmware using the debugger connection

After building the executable, connect the development board using the OpenSDA USB connection.

Note that the IAR project **Debugger Driver** option matches the board and interface used. To change the Debugger option, right click the application name, then select **Options -> Debugger-> Driver**.

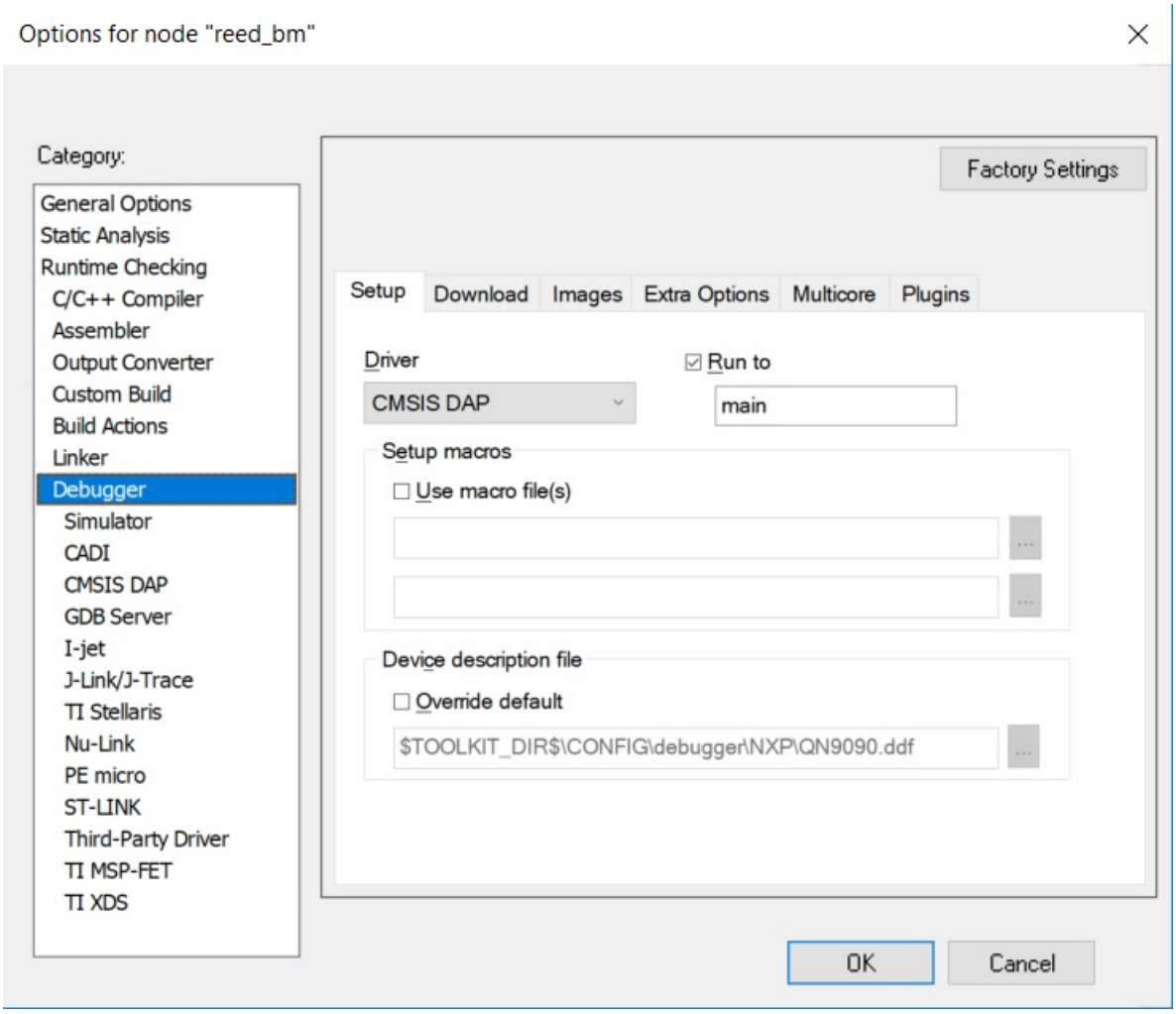


Figure 11. Setting Debugger Options CMSIS DAP

To deploy the application:

- Ensure the application project is active in the Workspace navigator
- Click **Download and Debug** in the toolbar.

6.6 Using EWARM batch build

EWARM allows users to build multiple projects and configurations in a batch. To use batch build with a JN5189 OpenThread SDK configuration workspace:

- Press F8
- In the **Batch Build** window select the configurations for batch build: Debug, Release or all (Debug and Release)
- Choose one of the **Make**, or **Rebuild All** options at the bottom of the window to build the selected batch configuration, or **Clean** to remove build artifacts

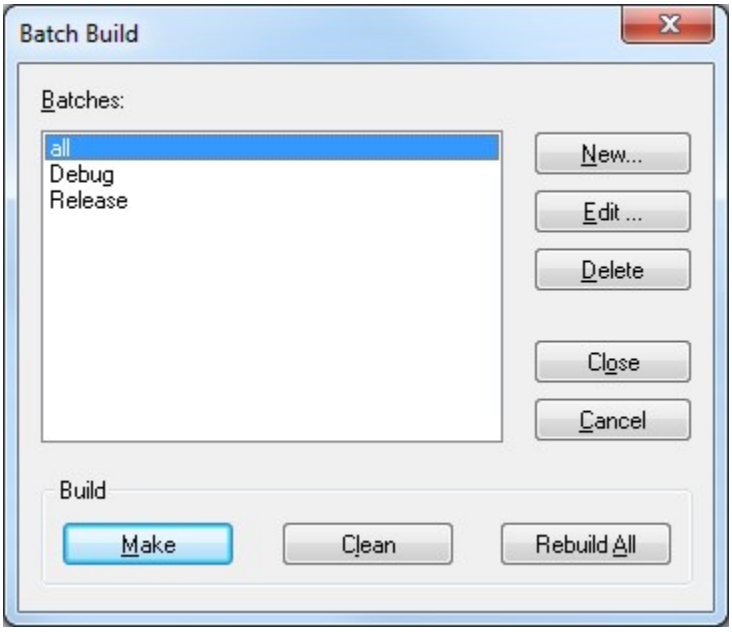


Figure 12. EWARM Batch Build

Chapter 7
Deploying Applications with MCUXpresso IDE

7.1 Project launch files

Each of the example demo projects provides a separate MCUX launch file. The launch file consist of the following.

- <.xml> MCUX Project configuration file

The launch files are contained in the folder structure starting at the **boards** subfolder in the root of the OpenThread SDK installation.

For example, to access the launch files for the **Router Eligible End Device** example to deploy on the JN5189DK6 development platform, navigate to the following subfolder: `\boards\jn5189dk6\wireless_examples\openthread\reed\bm\`

- reed_bm.xml

7.2 Opening a workspace

To open an example OpenThread demo application in the MCUX IDE for development, the first step is to install the JN5189 OpenThread SDK. To do that, drag and drop the .zip file into the Installed SDKs tab.

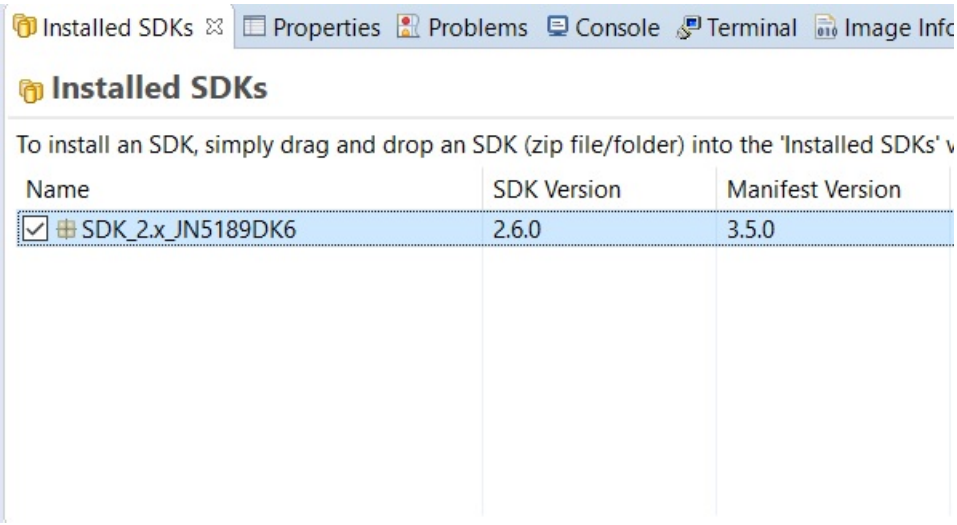


Figure 13. MCUXpresso Install SDK

When the Import SDK files finishes, select “Import SDK example(s)…” menu option.

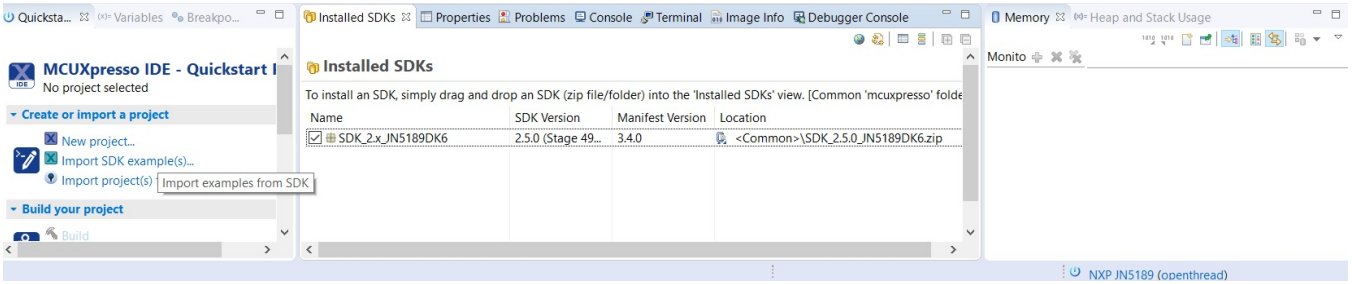


Figure 14. MCUXpresso Import Selected Project

Select the Connectivity Software, for example: `<C:\NXP\SDK_2.6.0_JN5189DK6>: \boards\jn5189dk6\wireless_examples\openthread\reed\bm\`, then click OK.

Select the appropriate board (for example jn5189) and then click "Next".

Select the desired example (for example: `wireless_examples\openthread\reed\bm`).

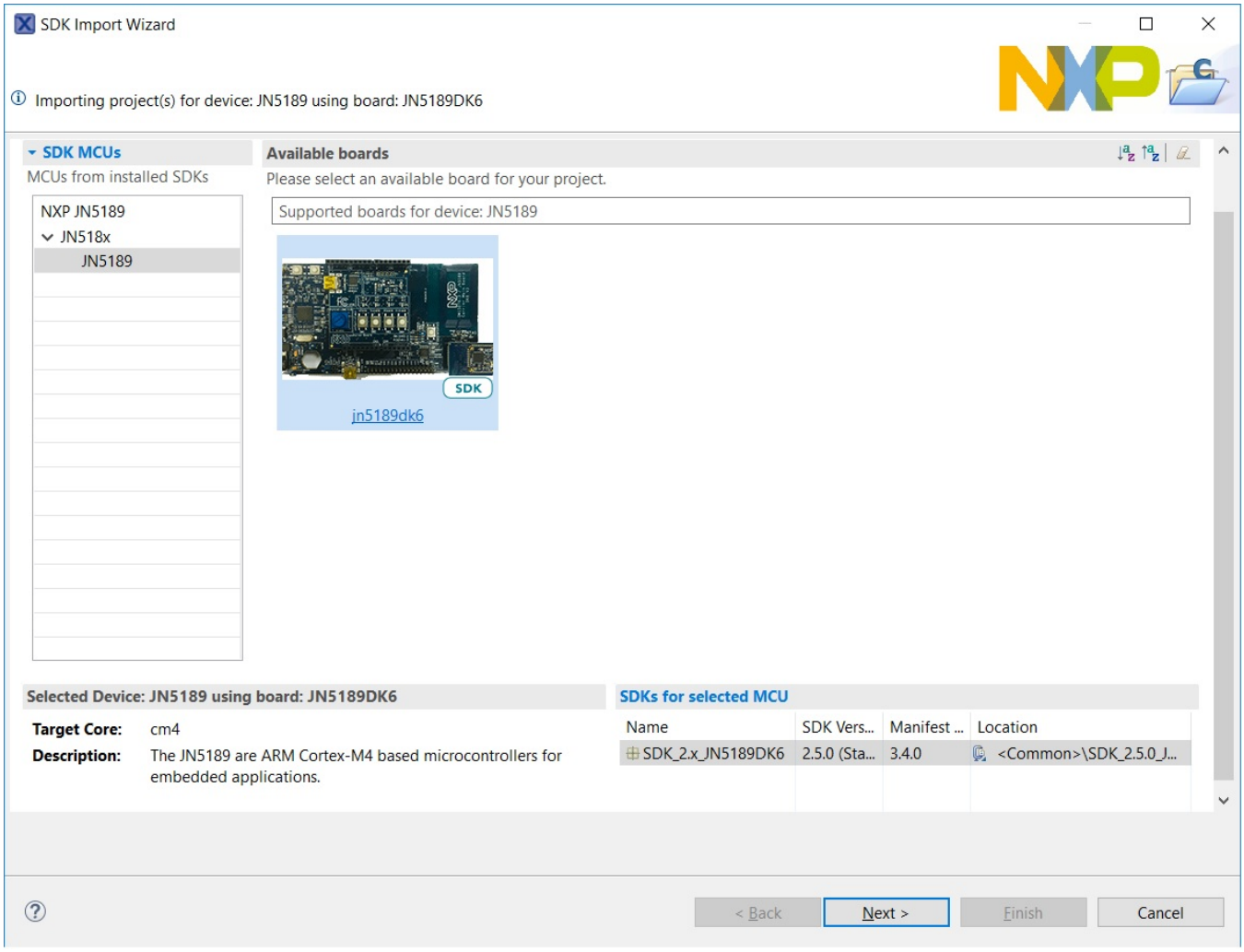


Figure 15. MCUXpresso Select Board

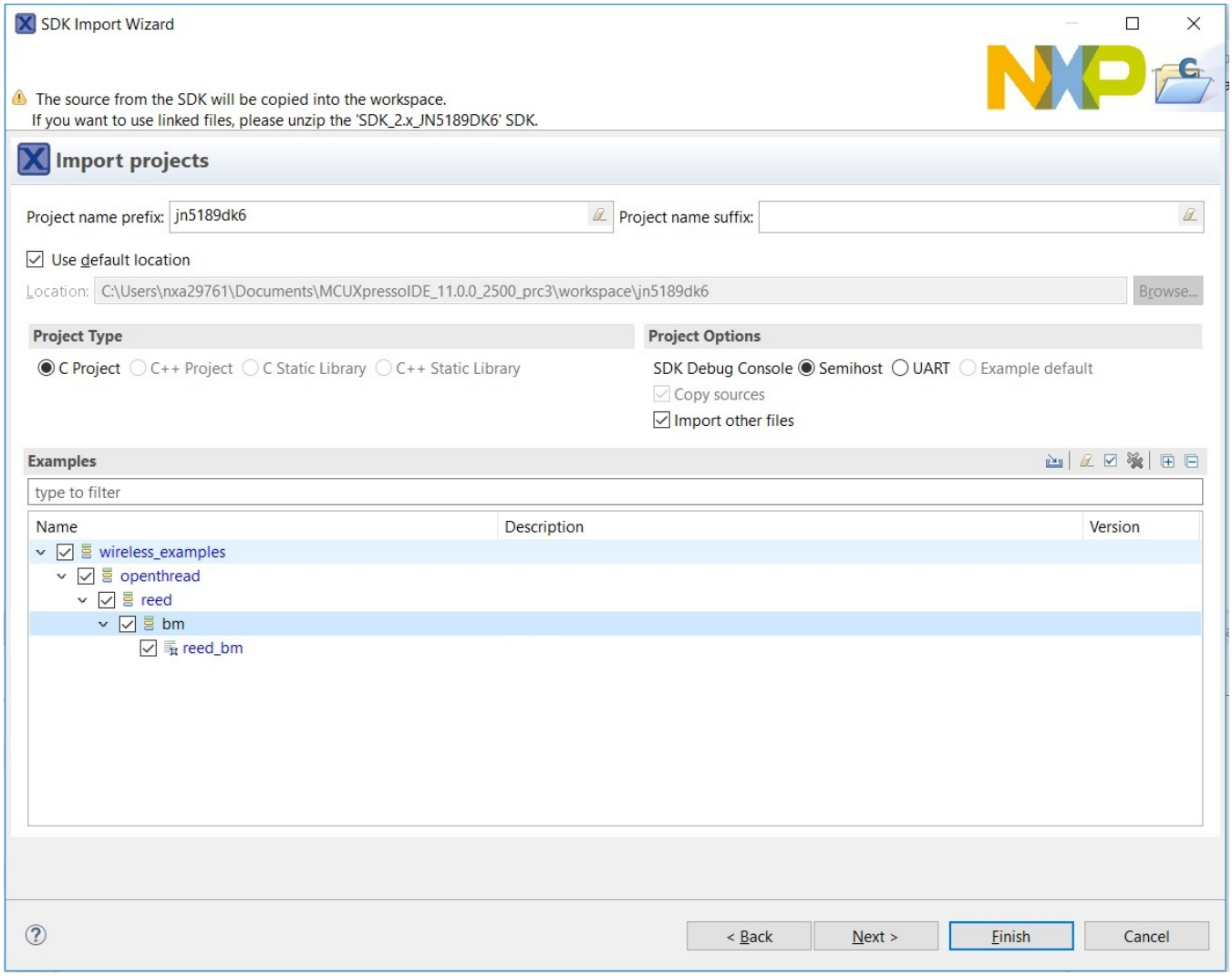


Figure 16. MCUXpresso Selected Project Imported

Click Finish on the Import Projects dialog box. The project is successfully imported into MCUX IDE.

7.3 Workspace contents

Once opened, the project for the main application **reed_bm** is displayed first.

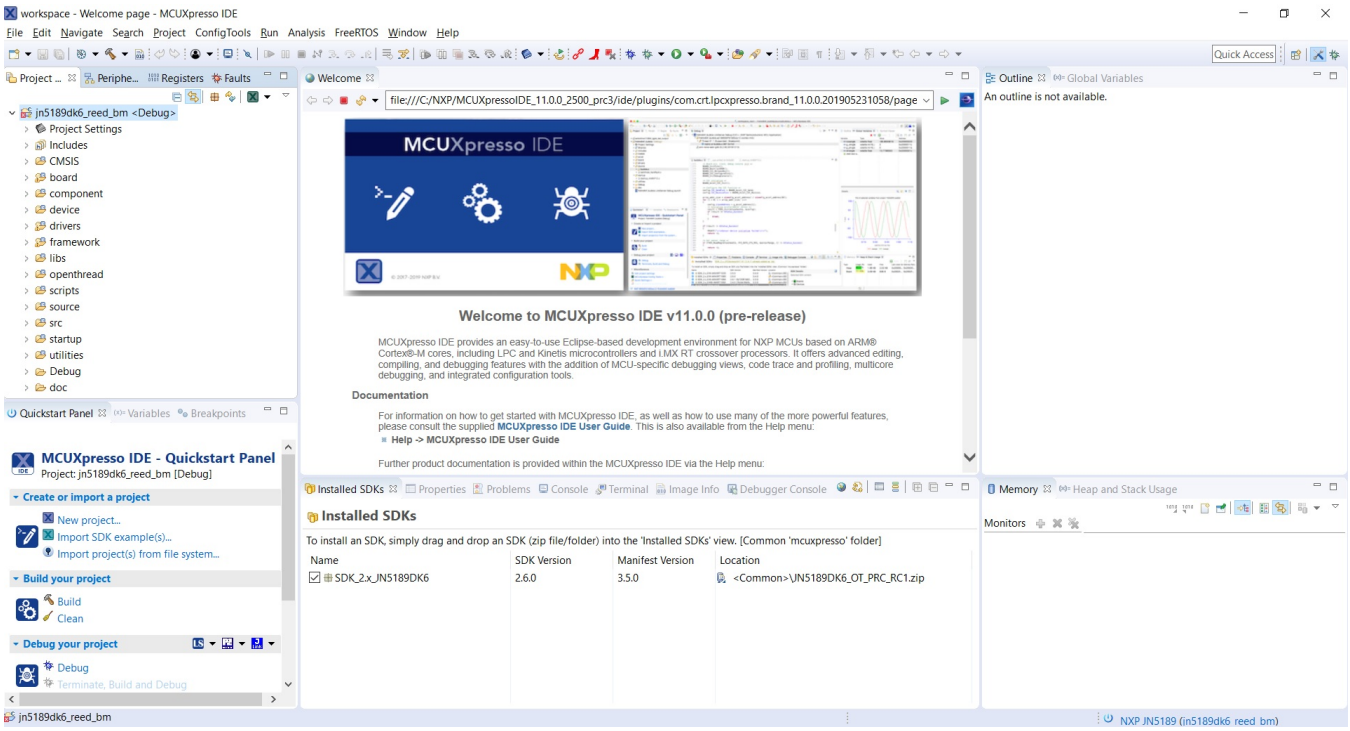


Figure 17. MCUXpresso Workspace

7.4 Project configurations

The optimization level for the Release configuration is set to Optimize Size (-Os)Size(-O0)

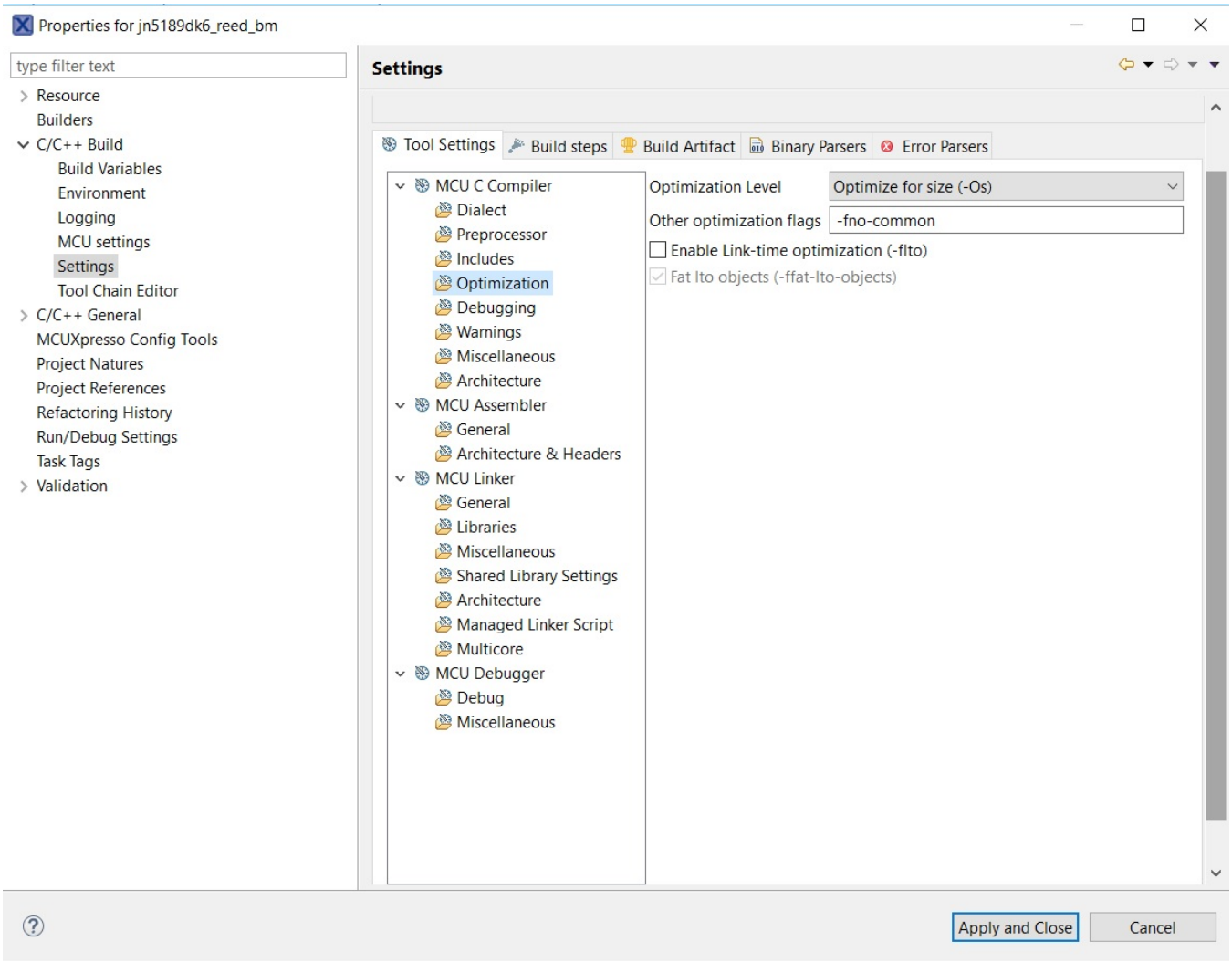


Figure 18. MCUXpresso Release Configuration Options

The optimization level for the Debug configuration is set to Optimize for debug (-Og).

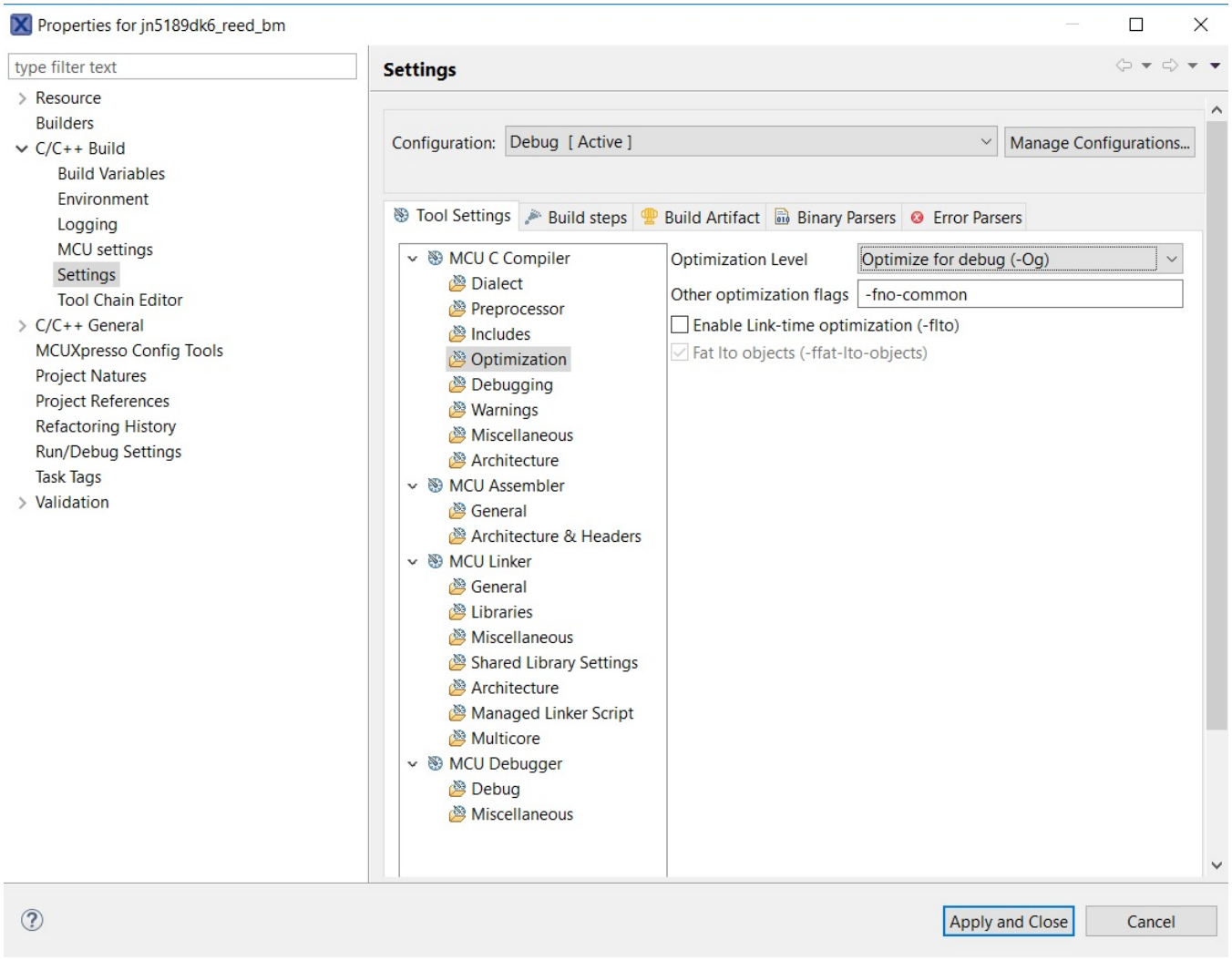


Figure 19. MCUXpresso Debug Configuration Options

7.5 Building the application executable

To build the application executable firmware in the MCUXpresso Workspace navigator, right click the application name of the project and select **Build Project**.

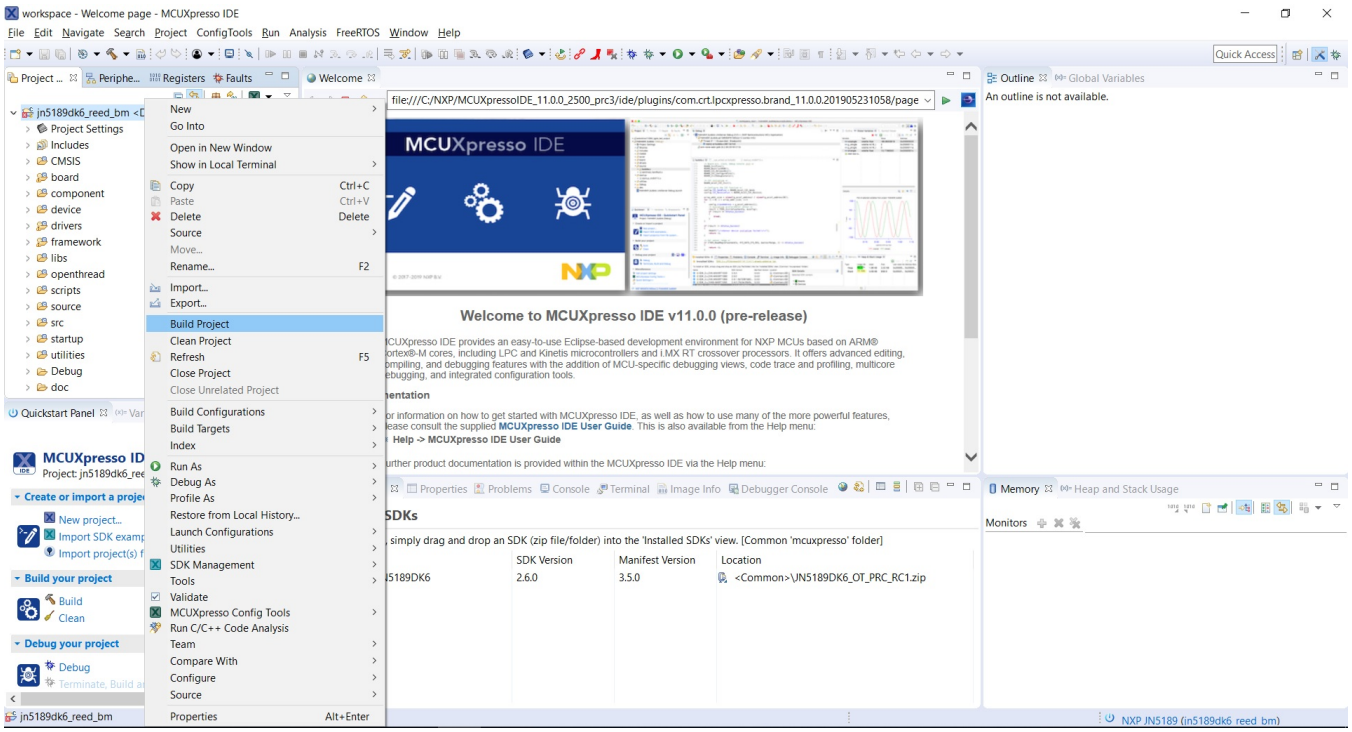


Figure 20. MCUXpresso Building Firmware

According to the selected project configuration (Debug or Release), a .bin file is generated in the corresponding …/Debug/Release folder, after the project is built.

7.6 Deploying the firmware using the debugger connection

After building the executable, connect the PC to the development board using the OpenSDA USB connection.

The first step is to create a debug configuration. The MCUX Debug Configuration dialog box is available using the Run->Debug Configuration menu option.

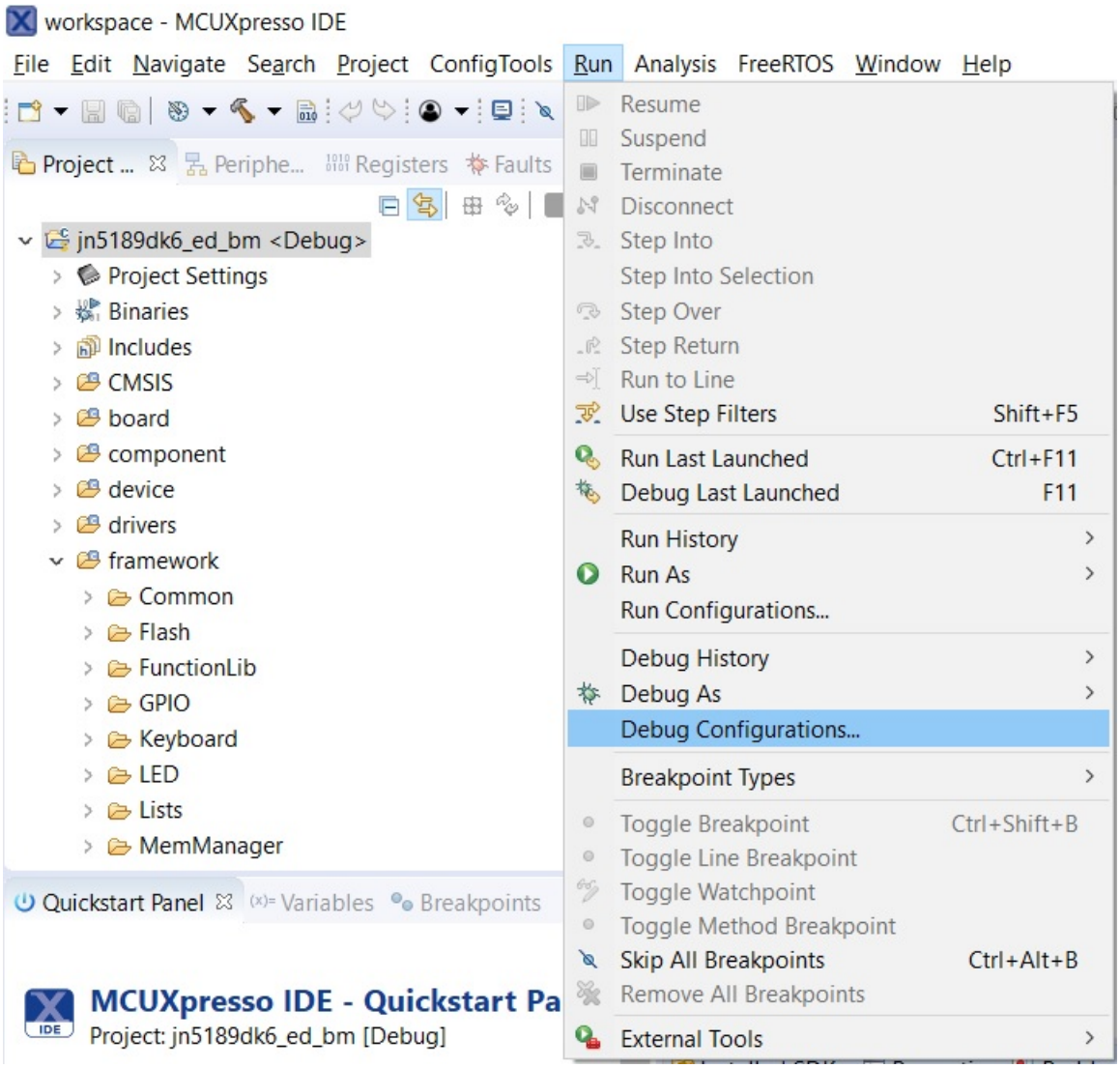


Figure 21. MCUXpresso Debug Configuration Section

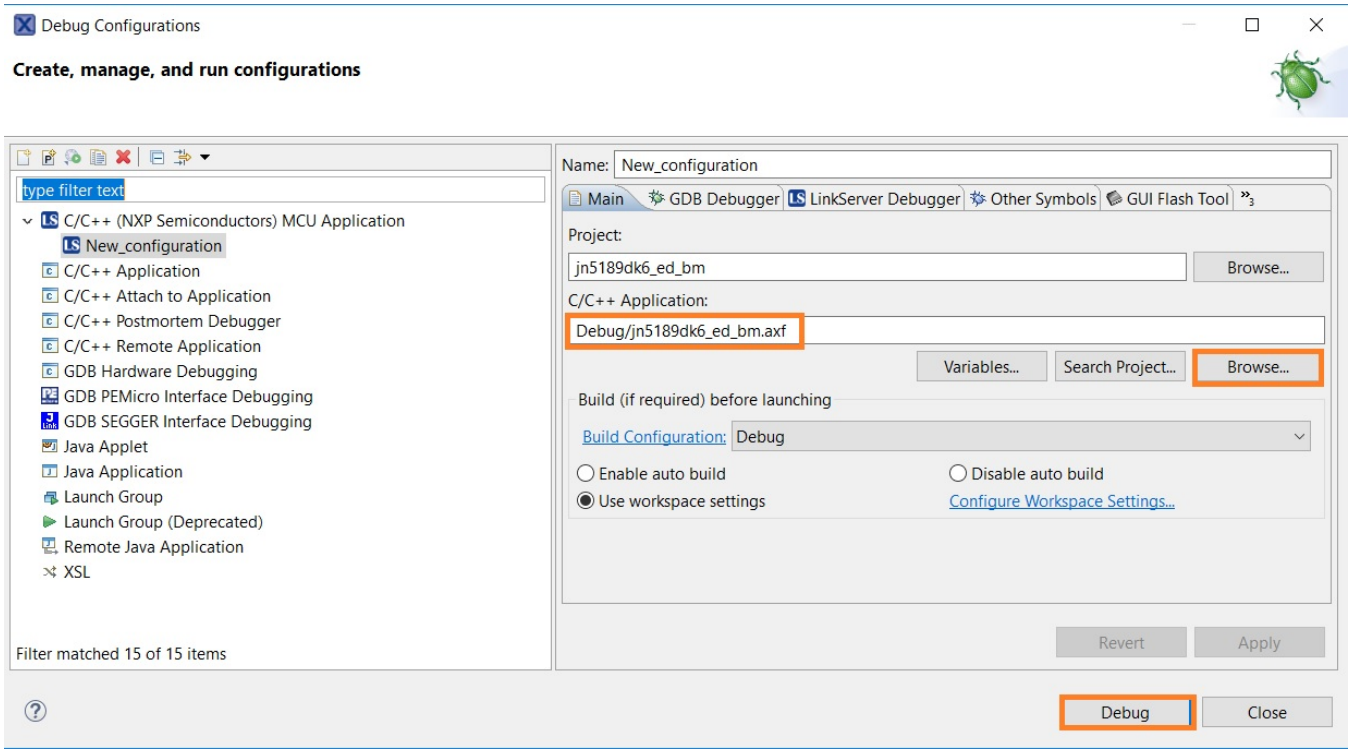


Figure 22. MCUXpresso Debug Configuration Creation

- A new window will appear, during the process to find the debug target. After successfully attaching to the board and loading the firmware, the debug can start:

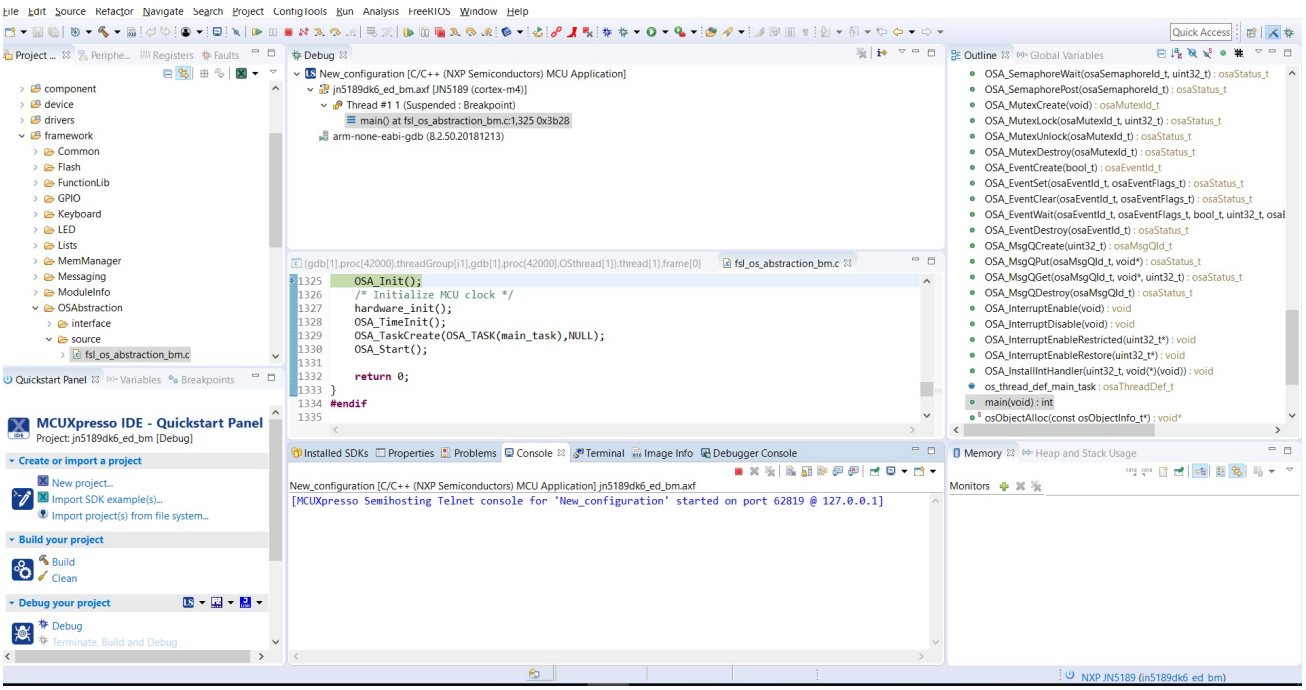


Figure 23. Start Debugging the Example Application

Chapter 8
Demo Functionality Overview

The example projects show the following Thread network and application layer functionalities:

- **Creating (bootstrapping) new OpenThread networks**
- **Joining existing OpenThread networks** of new devices based on predefined or customized network security
- **Configuring commissioning parameters** for joining of new devices including both the Joiner device configuration as well as the Commissioner configuration
- **Inspecting IPv6 Address Assignment** for Thread device interfaces
- **Using ICMP Ping** to test basic IP layer inter-connectivity
- **Sending Unicast and Multicast Application Data** using the **Constrained Application Protocol (CoAP)** message format over **UDP**
- **Inspecting and configuring network parameters** such as routing and neighbor tables, link-layer addresses, or radio channel parameters
- **IPv6 Border Routing** to applications and devices operating on different IP subnets and link layer technologies
- Noting **the differences in network behavior** between Routers, End Devices, and Low-power End Devices
- Facilitate **integration of JN5189 OpenThread devices in multiple chip configuration** driven by an external Host application processor

Chapter 9
Running Thread Network Scenarios

9.1 Board setup and provisioning

The network scenarios described in the following sections go through Thread network use cases and functionalities shown by the demo applications.

These scenarios assume availability of a minimum of 2 (two) OpenThread development boards in the following supported board set.

- JN5189DK6
- USB-JN5189 (limited board user interface)

Availability of more than 2 boards is useful in verifying application behavior in larger mesh networks consisting in multiple nodes or for deployment of end device multihop scenarios.

Different types of development boards can be mixed within the mesh network depending on availability.

For some scenarios, sequential identification of boards is useful. As such, the boards used in each scenario may be referred to in order based on their device type such as Node A, Node B, Node C, etc., such as below:

Table 2. Node and Application configuration

Node	Application
Node A	OpenThread Router Eligible Device
Node B	OpenThread End Device
Node C	OpenThread Border Router

9.2 Factory default state

9.2.1 Factory default

A device is placed a **Factory Default** state after initial application firmware loading as well as when it has been purposefully reset to this state as shown in the **Factory Reset** section.

In the Factory Default state, the device is **idle** from a OpenThread network perspective: it is not actively connected to any network and is provisioned with the original default firmware settings. In the Factory Default state, the device is ready for a user initiated action such as creation of a new Thread network or joining an existing network.

The **Factory Default** state is indicated by the development boards **blinking LEDs**.

Once devices create or join a Thread network, they save the network configuration to Non-Volatile Memory (NVM). As a result, the devices maintain their network settings even after a power-off reset cycle.

To ensure consistency, most of the network scenarios in the sections below assume the devices are reset to their Factory Default state before starting to run each scenario.

9.2.2 Factory reset

To reset an application to Factory Default state using board switches.

- **Press and HOLD FOR OVER 8 SECONDS any development board button switch** (with the exception of the Reset/RST switch)

NOTE

For low-power end devices, before holding any development board button switch for eight seconds, click any button once to wake the device up.

- After the button hold is released, the current Non-Volatile Memory settings are erased and the microcontroller powers-down/powers-up the rest of the system
- Board LEDs blink to show Factory Default state as indicated above

9.3 Overview

9.3.1 Overview

This scenario shows how to create (also known as: start, form, bootstrap) a new Thread network using a board button switch action when a device is in Factory Default mode and has the default firmware settings.

9.3.2 Board and application configuration

Table 3. Board and application configuration

Nodes Needed: at least 1	
Node	**Application**
Node A	OpenThread Router Eligible Device

9.3.3 Running the scenario

9.3.3.1 Initiating network creation

To **create (start, form, bootstrap) a new OpenThread network** with a user-initiated button switch action:

- **Short press any board button switch 2 (two) times** – any button switch initiates the action to create the new network on the double press except for the Reset/RST switch
- The 1st button press initiates network join attempts based on active network discovery requests; the 2nd button press indicates that the application stops the attempts to join a new network with the next opportunity (when the stack indicates a join failure process) and create a new network instead
- Verify the device has started as a network **Leader** role as indicated below

Note: with default settings, the OpenThread Router Eligible Device demo also auto-starts an initial network **Commissioner** application module once it starts-up as the initial leader. By default the active Commissioner allows joining of any devices which have their **device passphrase (PSKd)** set to default value.

9.3.3.2 Verifying network creation and leader role

Once the network is created as shown above, the **Router Eligible Device** also becomes an **Active Router** to accept new children nodes to join. At the same time it also self-promotes as a network partition **Leader**.

To verify **the node has created a new network and started successfully as a Leader**, note the state of the board LEDs as shown below:

- **JN5189DK6 + OM15082**
  - Solid red LED D1 on OM15082 board indicates Leader role
  - Solid red LED D2 on OM15082 board indicates Router role

9.4 Joining a router eligible device to an existing network

9.4.1 Overview

This scenario shows how to join an OpenThread Router Eligible Device in Factory Default state to an existing Thread mesh network using a board button switch action.

9.4.2 Board and application configuration

Table 4. Board and application configuration

Nodes Needed: at least 2	
Node	**Application**
Node A	OpenThread Router Eligible Device
Node B	OpenThread Router Eligible Device

9.4.3 Steps to create a new network and join a new device

Note: the images used in the detailed steps assume using a JN5189DK6 platform with OM15082 expansion board for Node A and Node B.

To **join a new Thread network** with a user initiated button switch action:

- Generally before joining the device, users must ensure there is an existing Thread network in range having with the following pre-conditions:
  - There is at least 1 (one) **Active Router** in the network allowing new devices to join
  - There is an active **Commissioner** for the network permitting joining from devices provisioned with device password (PSKd) *THREAD* and any EUI64 device address
- **Creating a new network usingNode A** and having the default settings of the Router Eligible Device application ensures the conditions above are met: Node A fulfills both the **Active Router** and **Commissioner** roles
- On **Node A** and **Node B**: ensure nodes are in **Factory Default** state (LEDs blink)
- On **Node A**: as shown in section **Steps to Create a New Thread Network** short press any board button switch **2 (two) times**
- On **Node A**: verify network creation succeeds as shown in section **Verifying Network Creation and Leader Role** (solid red LED D1 on OM15082 board indicates Leader role and solid red LED D2 on OM15082 board indicates Router role)
- On **Node B**: **Short press any board button switch 1 (one) time** – any button switch initiates the action to discover and join an existing network
- On **Node B**: as the user initiates joining, the board LED state cycles to display ongoing progress for several seconds as Node B discovers networks in range, negotiates security with the Commissioner and the device attaches to the network after having received its credentials after successful commissioning
- On **Node B**: after a few seconds, if attaching to the network succeeds, Node B starts and requests a Router ID from the network Leader (Node A). If this succeeds and Node B becomes an Active Router in the same network as Node B, its LED state stabilizes
- On **Node B**: if the initial discovery and joining attempt to an existing network does NOT succeed, the node retries joining indefinitely (shown by LED cycle) until a network becomes open for joining or the user resets the node

9.4.4 Joining an end device or low-power end device to an existing network

9.4.4.1 Overview

This scenario shows how to join an OpenThread End Device or Low-power End Device in Factory Default state to an existing Thread mesh network using a board button switch action.

9.4.4.2 Board and application configuration

Table 5. Board and application configuration

Nodes Needed: at least 2	
Node	**Application**
Node A	OpenThread Router Eligible Device
Node B	OpenThread End Device or OpenThread Low-power End Device

9.4.4.3 Running the scenario

NOTE

The images used in the detailed steps assume using a JN5189 platform with OM15082 expansion board for Node A and Node B.

To **join a Thread network** with a user initiated button switch action:

- Generally before joining the device, users must ensure there is an existing Thread network in range having with the following pre-conditions:
  - There is at least 1 (one) **Active Router** in the network allowing new devices to join
  - There is an active **Commissioner** for the network permitting joining from devices provisioned with device password (PSKd) *THREAD* and any EUI64 device address
- **Creating a new network using Node A** and having the default settings of the Open Router Eligible Device application ensures the conditions above are met: Node A fulfills both the **Active Router** and **Commissioner** roles

On **Node A** and **Node B**: ensure nodes are in **Factory Default** state (LEDs blink)

On **Node A**: as shown in section **Steps to Create a New Thread Network** short press any board button switch **2 (two)times**

- On **Node A**: verify network creation succeeds as shown in section **Verifying Network Creation and Leader Role** (solid red LED D1 on OM15082 board indicates Leader role and solid red LED D2 on OM15082 board indicates Router role)
- On **Node B**: **Short press any board button switch 1 (one) time** – any button switch initiates the action to discover and join an existing network
- On **Node B**: as the user initiates joining, the board LED state cycles to display ongoing progress for several seconds as Node B discovers networks in range, negotiates security with the Commissioner and the device attaches to the network after having received its credentials after successful commissioning
- On **Node B**: after a few seconds, if attaching to the network succeeds, LEDs turn off.
- On **Node B**: if the initial discovery and joining attempt to an existing network does NOT succeed, the node retries joining indefinitely (shown by LED cycle) until a network becomes open for joining or the user resets the node

9.5 Sending multicast LED control CoAP messages

9.5.1 Overview

This scenario shows how to use board button switch actions to send an **LED ON** or **LED OFF** command to the other devices on the Thread network and notice the LED control taking effect.

The LED control command is:

- encoded using a **CoAP frame format** at the application layer
- carried over **UDP** at the transport layer
- multicast using the **<All Thread Nodes>** realm-local multicast destination address at the IPv6 layer indicating it should be passed on to all nodes
- passed through via **6LoWPAN** and **MPL** by the upper IPv6 protocol stack to and from the link layer
- broadcast over-the-air at the **link layer** for all non-Low-power End Device destinations
- unicast over-the-air at the **link layer** on a MAC Data Request poll sequence for Low-power End Device destinations by the Routers which are respective parents of the End Devices

9.5.2 Board and application configuration

Table 6. Board and application configuration

Nodes Needed: at least 2	
Node	**Application**
Node A	OpenThread Router Eligible Device
Node B	OpenThread Router Eligible Device

9.5.3 Steps to send multicast LED control CoAP messages

NOTE

The images used in the detailed steps assume using JN5189DK6OM15082 platform for Node A and Node B. Tables at the end of the section show images of initiating the multicast action also for the other supported boards.

To **send a multicast LED control CoAP message** using a board button switch action:

- LED control works only if devices are already joined to the network. All following steps assume the steps of joining Node B to the network created by Node A have been performed as shown in section **Steps to Create a Network and Joining a New Device.** Node A and Node B should be Active Routers.
- On **Node B: press button switch for LED OFF** (SW2 on OM15082 board for JN5189) to ensure the specific LED used to show control actions turns off
- On **Node A**: note that an LED turns off if it was on (LED D3 on OM15082 board for JN5189)
- On **Node B**: **press button switch for LED ON** (SW1 on OM15082 board for JN5189)
- On **Node A**: note LED turns on
- On **Node B**: **press switch button for LED OFF again** (SW2 on OM15082 board for JN5189)
- On **Node A**: note LED turns back off
- On **Node A**: repeat the same steps above using board switches for LED OFF and LED ON and note the effects in the other direction taking place on Node B
- If 3 or more boards are used, note the effect of the LED control takes place on all the devices on the network indicating multicast messaging is being used

9.6 Announcing a data sink and sending unicast LED control CoAP messages

This scenario shows how to use board button switch actions to:

- set a node to announce itself as an application **Data Sink –** representing a single concentrator destination node on the network for application messages
- set a node to announce releasing (ceasing) the Data Sink role
- send unicast **LED ON** or **LED OFF** CoAP confirmable commands addressed to the Data Sink node exclusively and notice the LED control taking effect

The commands to **announce setting a Data sink** and **announce releasing a data sink** are sent as multicast CoAP messages. Once a Data Sink is set, LED commands sent to it are sent unicast, on an optimal route and are confirmable, being retransmitted if not acknowledged by the Data Sink node.

9.6.1 Board and application configuration

Table 7. Board and application configuration

Nodes Needed: at least 3	
Node	**Application**
Node A	OpenThread Router Eligible Device
Node B	OpenThread Router Eligible Device
Node C	OpenThread Router Eligible Device

9.6.2 Steps to announce and send unicast messages to a data sink

Note: the images used in the detailed steps assume using JN5189DK6OM15082 platform for Node A and Node B.

To **announce a node as a Data Sink** using a board button switch action take the following steps:

- Data sink announcements work only if devices are already joined to the network. All following steps assume the steps of joining Node B and Node C to the network created by Node A have been performed as shown in section **Steps to Create a Network and Joining a New Device.** Node A, Node B, and Node C should be Active Routers.
- Check that a data sink is not active and LED control messages are multicast:
- On **Node A: press button switch for LED ON** (SW1 on OM15082 board for JN5189)
- On **Node B and C**: note that the LED D3 on OM15082 board turn on
- On **Node B: Short press button switch to Create Data Sink** (SW3 on OM15082 board for JN5189)
- On **Node A: press button switch for LED OFF** (SW2 on OM15082 board for JN5189)
- On **Node B**: note LED D3 on OM15082 board has gone of

On **Node C**: note LED has NOT gone off

A Data Sink has been created with Node B as a concentrator. Node C and Node A no longer receive LED control messages from other devices which are now unicast to Node B

- On **Node C**: check that the LED ON and LED OFF buttons only control Node B LED, but not Node A
- On **Node B**: **Long press (press and HOLD 2-3 seconds) switch button to Release Data Sink** (SW3 on OM15082 board for JN5189)
- On **Node A** and **Node C:** check that the LED ON and LED OFF buttons now have reverted back to multicast behavior controlling all other nodes.

9.7 Sending data from end devices

Nodes B and Node C in the **Sending Multicast LED Control CoAP messagesand Announcing a Data Sink and Sending Unicast LED Control CoAP messages** can also be exercised via **End Device** and **Low-power End Device** nodes with similar behavior.

NOTE

When using a **Low-power End Device** the device only wakes up from low-power state by default at **3 seconds** to receive data, or otherwise on a button press on a subset of board button switches as detailed below. After a wake up, the low power end device sends the CoAP message and goes back to sleep until the next poll. For obtaining the best low power results it is recommended to remove the OM15082 expansion board. In this case there are only 2 buttons available as described in #unique_58:

Table 8. Low-power wake up switches

Board	Low-power Wake Up Switches
JN5189DK6	User interface / ISP switches on DK6 carrier board

9.8 Network partitioning and merging

9.8.1 Overview

This scenario shows how to visualize the OpenThread network states of **partition creation** and **partition merging** using the board LEDs.

A Thread routing segment is a group of Routers and their associated children End Devices which can reach each other on direct radio links or via a multihop path (through messages forwarded through a set of other routers in the segment).

When two or more routing segments of a Thread network become disconnected - for instance due to nodes being moved out of range or intermediary routers being turned off - then each routing segment creates a distinct network partition having its own Leader node.

When Routers detect loss of connectivity to the current Leader node due to that being in a different network partition, they advertise their state and a new Router in the current routing segment emerges as a Leader of the new network partition which got disconnected from the segment containing the previous Leader.

In a worst case scenario, even a single Router can be part of a distinct network partition.

When Routers in distinct partitions come back into connectivity range, their respective partitions merge back into a common partition having a single Leader.

9.8.2 Board and application configuration

Table 9. Board and application configuration

Nodes Needed: at least 2	
Node	**Application**
Node A	OpenThread Router Eligible Device
Node B	OpenThread Router Eligible Device

9.8.3 Using partitioning and merging

Note: the images used in the detailed steps assume using JN5189DK6OM15082 platform for Node A and Node B. Tables at the end of the section show images of initiating the multicast action also for the other supported boards.

To **create a partitioning situation**:

- All following steps assume the initial joining of Node B to the network created by Node A has been performed as shown in section **Steps to Create a Network and Joining a New Device.** Node A and Node B should be Active Routers and in connectivity range.
- Create a loss of connectivity situation so that Node B is no longer in radio range with Node A. To achieve that either:
  - **Temporarily power off Node A**
  - **Take one of the nodes physically out of range** from the other – for instance by moving the nodes to different floors for a multistoried building or increasing the physical distance between the nodes significantly (100 feet or longer)
- Note that if **Node B** is not power-off reset, then in 1-2 minutes after losing connectivity, it indicates via its LED it has become a new partition Leader (solid red LED D1 on OM15082 board indicates Leader role).

This indicates Node B is part of a distinct network partition.

- If a **power-off reset takes place onNode B** as the nodes are out of range, the partitioning happens more quickly as Node B starts back up as an Active Router without detecting other neighbor Routers.
- **Bring the 2 nodes back into range** (for example, power Node A back on)
- Note the network reverts back to having one Leader, indicating the **distinct partitions have merged**. Note that Node B can also emerge as the single Leader when the partitions are merged back.

Chapter 10
Running Thread Network Scenarios Using the Shell Interface

10.1 Board setup and provisioning for shell usage

The following OpenThread Stack examples are provisioned by default with a shell command line interface accessible via a Terminal application such as PuTTY:

- OpenThread Router Eligible Device
- OpenThread End Device
- OpenThread Border Router

10.2 Shell provisioning in Windows® OS

To connect to a board shell in Windows OS:

- Plug an USB cable attached to a host PC into each device and power on the board.
- Expand section **Ports (COM&LPT)** and check if a COMxx port entry appears as show in the figure below.

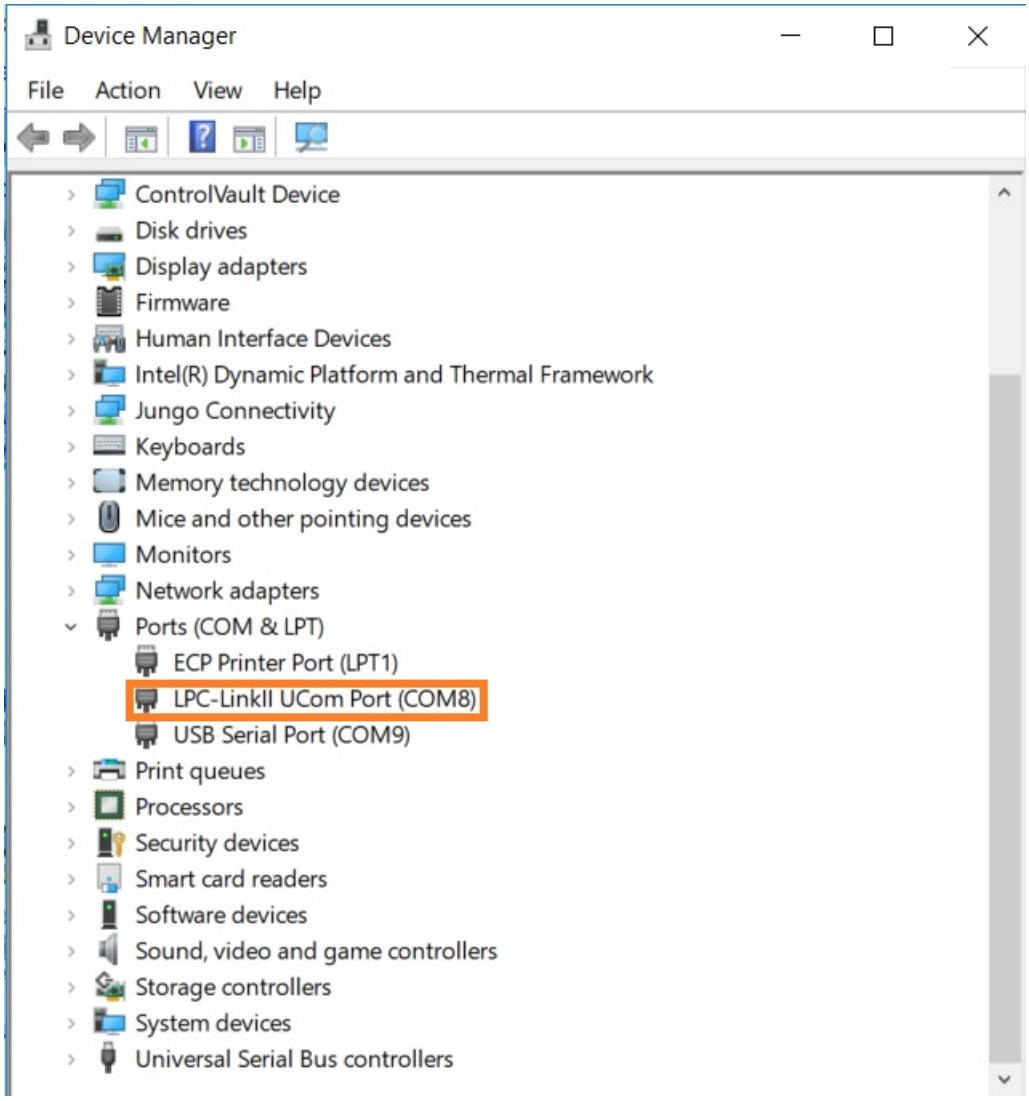


Figure 24. Using Device Manager to determine COM port numbers of USB connections

- To open a shell terminal from the PuTTY application, use the COM port number identified above and a 115200 bps baud rate be used in the port settings

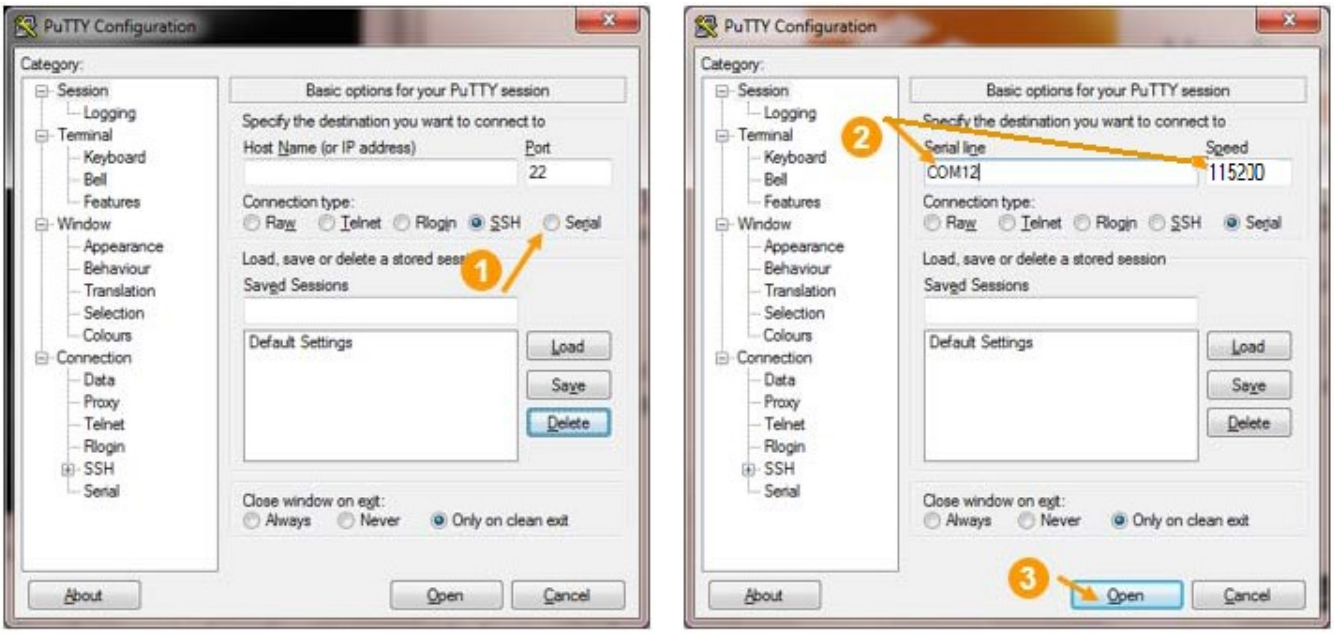


Figure 25. Starting PuTTY

10.3 Shell provisioning in MAC® OS X

To connect to a board shell interface in MAC OS X:

- An USB modem driver should automatically be loaded by MAC OS X when plugging in the board.
- To check the board is recognized by the operating system lunch Spotlight and type: **Terminal**
- Select and launch the terminal application shown in the Spotlight results
- At the terminal prompt type:

ls /dev/tty.*

- A /dev file system entry such as below is shown for the board:

/dev/tty.usbmodem0000001

- The COM port can be accessed via a terminal application such as **screen** with the 115200bps open port baud rate

10.4 Creating a new Thread network and commissioning a device

10.4.1 Overview

This scenario shows how to Factory Reset a device, proceed to create a new Thread network using the shell interface, and join a new device to the network by commissioning it based on a customized device pass phrase (PSKd).

10.4.2 Board and application configuration

Table 10. Board and application configuration

Nodes Needed: at least 2	
Node	**Application**
Node A	OpenThread Router Eligible Device
Node B	OpenThread Router Eligible Device

10.4.3 Running the scenario

- On both nodes, factory reset the devices device using the shell by entering:

```
> factoryreset
```

Note: when using the USB connection, the device needs to be re-inserted and re-opened in PuTTY or the terminal application.

- On **Node A** shell, to create a new Thread network enter:

```
> panid 0xabcd
Done
> ifconfig up
Done
> thread start
Done
```

Note the status messages in the shell indicating the network parameters. Note a Local Commissioner instance also starts on Node A

- On **Node B** shell, initiate joining with the default PSKd:

```
> panid 0xabcd
Done
> ifconfig up
Done
> thread start
Done
```

- Note the status messages in the shell indicating the joining results.

10.5 Inspecting IP address assignment and testing connectivity

10.5.1 Overview

This scenario shows how to inspect the IP address assigned to the Thread interface and use the ICMP Ping to test basic connectivity.

10.5.2 Board and application configuration

Table 11. Board and application configuration

Nodes Needed: at least 3	
Node	Application
Node A	OpenThread Router Eligible Device
Node B	OpenThread Router Eligible Device
Node C	OpenThread Router Eligible Device

10.5.3 Running the scenario

- On **All Nodes**, factory reset the devices device using the shell by entering:

> factoryreset

- On **Node A** shell, to create a new Thread network enter:

> panid 0xabcd

Done

> ifconfig up

Done

> thread start

Done

Note the status messages in the shell indicating the network parameters. Note a Commissioner instance also starts on Node A

- On **Node B** shell initiate joining with the default PSKd:

> panid 0xabcd

Done

> ifconfig up

Done

> thread start

Done

Note the status messages in the shell indicating the joining results.

- On **Node C** shell initiate joining with the default PSKd:

> panid 0xabcd

Done

> ifconfig up

Done

> thread start

Done

Note the status messages in the shell indicating the joining results.

- On **Node A**, enter the ipaddr command to view IP address information**:**

> ipaddr

fdde:ad00:beef:0:0:ff:fe00:fc30

fdde:ad00:beef:0:0:ff:fe00:fc00

fdde:ad00:beef:0:0:ff:fe00:d400

fdde:ad00:beef:0:14f4:5afc:30ba:2019

fe80:0:0:0:e41d:8e0e:af4d:3f4e

Done

- On **Node B**, enter the ipaddr command to view IP address information:

> ipaddr

fdde:ad00:beef:0:0:ff:fe00:d401

fdde:ad00:beef:0:603d:f969:d2a8:b584

fe80:0:0:0:802e:8673:6937:22a2

Done

- On **Node C**, enter the ipaddr command to view IP address information.
- Using the mesh local ML64 (ML-EID) addresses of the destination noted via ipaddr command to ping the other nodes over the Thread network.
- For instance, **on Node A**, to ping Node B:

> ping fdde:ad00:beef:0:0:ff:fe00:d401

16 bytes from fdde:ad00:beef:0:0:ff:fe00:d401: icmp_seq=1 hlim=64 time=28ms

> ping stop

Done

10.6 Sending application data CoAP messages using the shell

10.6.1 Overview

This scenario shows how to send and view the receive indication for CoAP messages using the shell interface.

10.6.2 Board and application configuration

Table 12. Board and application configuration

Nodes Needed: at least 2	
Node	Application
Node A	OpenThread Router Eligible Device
Node B	OpenThread Router Eligible Device

10.6.3 Running the scenario

- On All Nodes, factory reset the devices device using the shell by entering:

> factoryreset

- On **Node A** shell, to create a new Thread network enter:

> panid 0xabcd

Done

> ifconfig up

Done

> thread start

Done

Note the status messages in the shell indicating the network parameters. Note a Commissioner instance also starts on Node A

- On **Node B** shell initiate joining with the default PSKd:

> panid 0xabcd

Done

> ifconfig up

Done

> thread start

Done

Note the status messages in the shell indicating the joining results.

- On **Node A**, enter the ipaddr command to view IP address information**:**

> ipaddr

fdde:ad00:beef:0:0:ff:fe00:fc30

fdde:ad00:beef:0:0:ff:fe00:fc00

fdde:ad00:beef:0:0:ff:fe00:d400

fdde:ad00:beef:0:14f4:5afc:30ba:2019

fe80:0:0:0:e41d:8e0e:af4d:3f4e

Done

- On **Node B**, enter the ipaddr command to view IP address information**:**

> ipaddr

fdde:ad00:beef:0:0:ff:fe00:d401

fdde:ad00:beef:0:603d:f969:d2a8:b584

fe80:0:0:0:802e:8673:6937:22a2

Done

- Use the addresses noted via ipaddr command to send CoAP LED control messages.

For instance on **Node A** enter the command to send to Node B a LED ON command:

> coap post fe80:0:0:0:802e:8673:6937:22a2 led con on

Done

> coap response from fe80:0:0:0:802e:8673:6937:22a2

- If Node B has an LED, it turns on.
- On **Node A** to send to Node B a LED OFF command enter:

> coap post fe80:0:0:0:802e:8673:6937:22a2 led con off

Done

> coap response from fe80:0:0:0:802e:8673:6937:22a2

- On **Node B** press “Report Temperature” button switch on Node B (SW4 on OM15082 board for JN5189)
- On **Node A** note how the indication of the temperature (in degrees Celsius) is displayed in the shell as coming from the IP address of Node B:

Temp:27.37 From IPv6 Address: fe80:0:0:0:802e:8673:6937:22a2

Chapter 11
Running Border Router Application Scenarios

11.1 Border routers overview

The role of a Border Router is to connect a Thread Network to other IP-based networks.

The NXP OpenThread Border Router hardware solution is composed of a UDOO Neo Board and a JN5189 Dongle Device that plugs in the USB port of the Border Router.

The NXP OpenThread Border Router software solution consists of:

- An Ubuntu 18.04 OS image for UDOO Neo: *udoo_neo_os_nxp.img*. The image can be download from the MCUXpresso production server; In order to create this image, we started from Ubuntu 18.04. 4.0 Beta 1 Desktop image (available on the official UDOO website) then modified its Kernel:
  - Added full iptables kernel-support such that the IP-packets can be NAT'ed between the Thread interface and the Ethernet interfaces;
  - Added required Kernel modules for running a Docker instance.
- A Docker image that encapsulates all the Border Router functionalities: *udoo_neo_container_nxp*; The most important capabilities of the Border Router are:
  - Web GUI for configuration and management;
  - Thread Border Agent to support external commissioning;
  - DHCPv6 Prefix Delegation to obtain IPv6 prefixes for a Thread network;
  - NAT64 for connecting to IPv4 networks;
  - DNS64 to allow Thread devices to initiate communications by name to an IPv4-only server;
  - Thread interface driver using wpantund.
- A Bash script, *run_br_container.sh*, required for launching in execution the Docker container;
- A Bash script, *restart_br_agent.sh*, required for restarting the web agent service running in the Docker container.
- Both Docker image and bash scripts can be downloaded from the MCUXpresso production server.
- An MCU/IAR project/binary file for the JN5189 USB Dongle: *rcp-bin*

This software solution is based on the OpenThread Radio Co-Processor Architecture where the host communicates with the controller using the Spinel Protocol through a wpantun interface .

Steps for assembling the NXP Border Router Solution:

- flash rcp-bin file on the JN5189 Dongle using the NXP Flash Programmer or the Firmware Loader capability enclosed in the NXP Test Tool.
- plug the JN5189 Dongle in the USB port of the UDOO Board;
- flash *udoo_neo_os_nxp.img on* an SD card using a tool like Win32DiskImages (example:https://codeyarns.com/2013/06/21/how-to-write-a-disk-image-using-win32-disk-imager/);

NOTE

The user needs an SD card with a storage size of at least 8Gb.

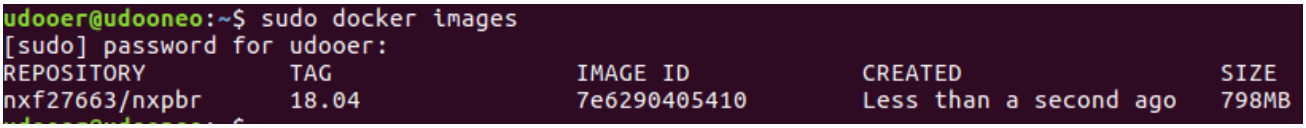
- boot the Udoo Board and connect to it using a tool like ssh (user and pass are udooer:udooer);
- install Docker: curl -sSl https://get.docker.com | sh;

NOTE

Please note that before installing the Docker package you may need to setup the Linux DNS server and set the date and time of the system.

- copy Docker image and Bash scripts to the Udoo Board;
- load the Docker image: sudo docker load -i udoo_neo_container_nxp;
- Check that the Docker image was successfully loaded: sudo docker images (see the attached docker_images.png);

Figure 26. Docker images



- Launch a Docker container: sudo ./run_br_container.sh;

NOTE

There are times when the Thread Commissioning App cannot discover/connect to the mDNS services exposed by the Border Router. In this case, please restart the web agent service by running the restart_br_agent.sh bash script (sudo ./restart_br_agent.sh). The role of this script is to connect to the running Docker container and restart the otbr-agent service.

11.2 External routing via Ethernet

11.2.1 Overview

This scenario shows how to test IP connectivity between an OpenThread Node and an Internet node.

11.2.2 Board and application configuration

Table 13. Board and application configuration

Nodes Needed: at least 2	
Node	**Application**
Node A (UDOO Neo + JN5189 USB Dongle)	OpenThread Border Router POSIX on UDOO Neo with JN5189 RCP configuration
Node B (JN5189 Module + JN5189 DK6 Board)	OpenThread Router Eligible Device

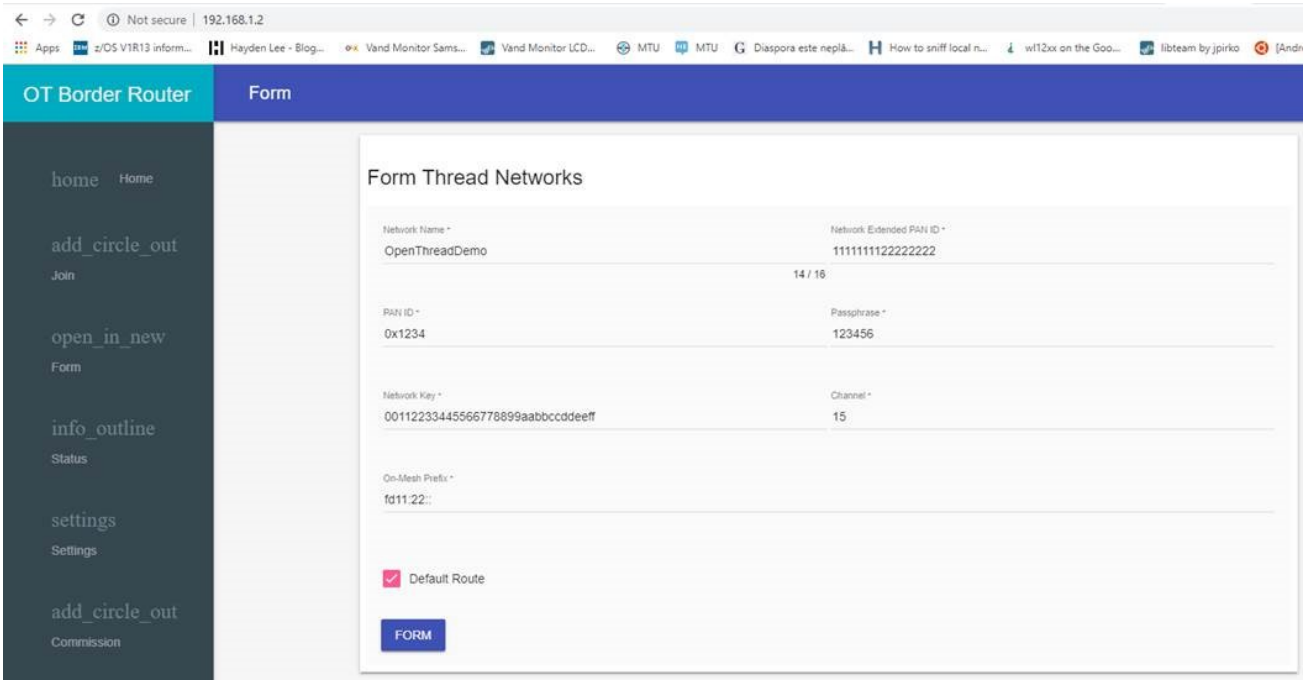
NOTE

The NXP Open Thread Border Router must be connected to the Internet using the Ethernet connection.

11.2.3 Running the scenario

- For **Node A**, connect an Ethernet cable directly to the Ethernet port of UDOO Neo board:
  - check that an IPv4-DHCP address was assigned (e.g.: by sniffing the Ethernet traffic with Wireshark);
  - use the above address for connecting through SSH to the Border Router shell using the username **udooer** and the password **udooer**. In case of an SSH error/warning, follow the instructions from the prompt for regenerating a new pair of SSH keys;

  - once connected to the Border Router shell, make sure that a ping to an Internet address (e.g.: 8.8.8.8) is successful;
- For **Node B**, flash the REED Router app, then plug the dongle in the USB port of the UDOO Neo device;
- The Border Router Web service advertises the “BorderRouter-AP” SSID. Please connect to this Wi-Fi Network then access the Web Server main page located at 192.168.255.1. Using the left menus navigate to the “Form” tab which allows the creation of a Thread Network. You may keep the default values then click on the “Form” button.



NOTE

All the shell commands will be executed inside the docker container, so the first step is to get a shell to the running container:

```
sudo docker exec container_id /bin/bash
```

The id of the running container can be obtained using the command:

```
sudo docker container ls
```

- Start a 802.15.4 sniffer on the above selected channel and observe the MLE advertisements triggered by the creation of this network; Also, the status of the Form command can be found using the Border Router shell:
-

```
sudo wpanctl status
wpan0 => [
    "NCP:State" => "associated"
    "Daemon:Enabled" => true
    "NCP:Version" => "OPENTHREAD/g89558fd-id; JN5189; Nov 04 2019 09:24:19"
    "Daemon:Version" => "0.08.00d (0.07.01-124-g038e8b0/0.07.01-153-g6752c40; Nov  4 2019
17:55:05)"
    "Config:NCP:DriverName" => "spinel"
    "NCP:HardwareAddress" => [CCEED0CA9B5A12DB]
    "NCP:Channel" => 14
    "Network:NodeType" => "leader"
    "Network:Name" => "OpenThread"
    "Network:XPANID" => 0xDEAD1111DEAD2222
    "Network:PANID" => 0xDEAD
```

```
    "IPv6:LinkLocalAddress" => "fe80::bc53:88e7:f6bf:9046"
    "IPv6:MeshLocalAddress" => "fdde:ad11:11de:0:8f4:2c2f:e2de:d3e1"
    "IPv6:MeshLocalPrefix" => "fdde:ad11:11de::/64"
    "com.nestlabs.internal:Network:AllowingJoin" => false
]
sudo wpanctl getprop Thread:OnMeshPrefixes
Thread:OnMeshPrefixes = [
    "fd11:22::              prefix_len:64   origin:user     stable:yes flags:0x33 [on-mesh:1 def-
route:1 config:0 dhcp:0 slaac:1 pref:1 prio:med]"
]
sudo wpanctl getprop Network:PSKc
Network:PSKc = [198886F519A8FD7C981FEE95D72F4BA7]
```

- On **Node A**, in the shell, start the Border Router internal Commissioner and allow any Thread Node that is in the knowledge of the preshared key to join the network:
  - sudo wpanctl commissioner start;
  - sudo wpanctl commissioner joiner-add “*” 120 J01NME;
- On **Node B**:
  - Ifconfig up;
  - Joiner start J01NME
- Note the status messages in the shell indicating the joining results.
  - Once the Joining is complete attach to the Border Router:
    - thread start
  - Check the state after a few moments to confirm. It may initially start as a child, but within two minutes it should upgrade to a router.
    - state
  - Also check the device's IPv6 addresses. It should have a Global address using the On-Mesh Prefix specified during formation of the Thread network:

    ```
    ipaddr
    fdde:ad11:11de:0:0:ff:fe00:9400
    fd11:22:0:0:3a15:3211:2723:dbe1
    fe80:0:0:0:6006:41ca:c822:c337
    fdde:ad11:11de:0:ed8c:1681:24c4:3562
    ```

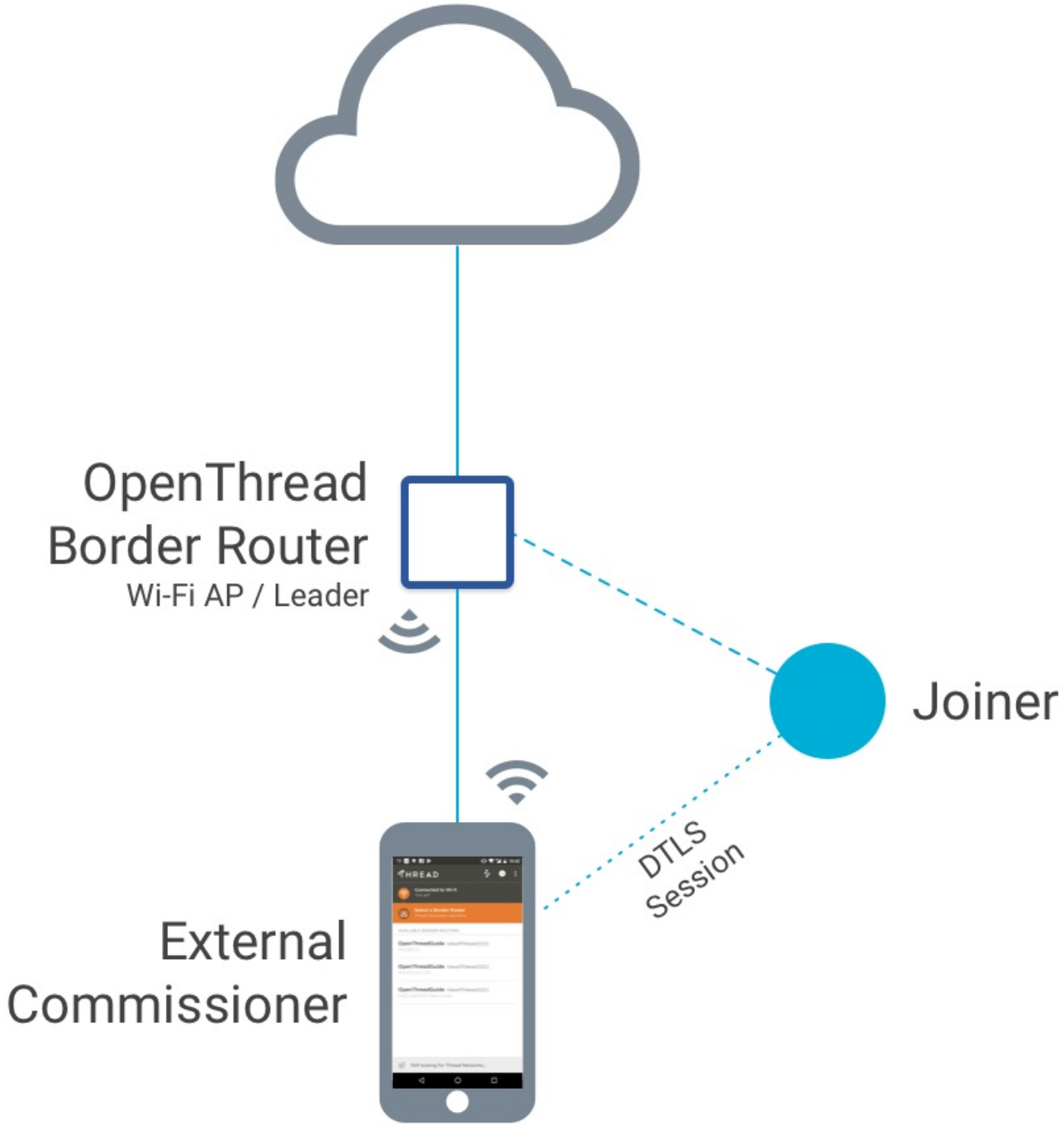
  - Ping the External Internet (For example, the Well-Known Prefix of 64:ff9b::/96 and an IPv4 address of 8.8.8.8 combine to form an IPv6 address of 64:ff9b::808:808.):
    - Ping 64:ff9b::808:808

11.3 External Thread Commissioning

The purpose of this use case is to explain how to join a new JN5189 OpenThread node into an already existing network using an external commissioner (a mobile phone application).

The UDOO Board serves as a Wi-Fi Access Point, having the SSID “BorderRouter-AP”.

OpenThread Border Router (OTBR) features a Thread Border Agent, which supports external Thread Commissioning. In external Thread Commissioning, a device outside of the Thread network (for example, a mobile phone) commissions new devices onto the network.The Thread Commissioner serves to authenticate a user (external Commissioner) or a Thread device onto the Thread network. After authentication, the Commissioner instructs the Border Router to transfer Thread network credentials, such as the master key, to the device directly.This is an example of in-band commissioning, where Thread network credentials are transferred between devices over the radio.



- **Form the Thread Network**
  - For more details, please check the specific section and the step by step instructions available on the openthread.io website.
- **Prepare the Joiner Device**
  - Once the Joiner device is ready, obtain its factory-assigned IEEE EUI-64. Use the eui64 command in the OpenThread CLI:
    - **> eui64**
    - **000b57fffe15d68**

    - Done

- **Download the Thread Commissioning Application**
  - For more details, please check the specific section and the step by step instructions available on the openthread.io website.
- **Connect to the Border Router**
  - For more details, please check the specific section and the step by step instructions available on the openthread.io website.
- **Commission the Joiner**
  - For more details, please check the specific section and the step by step instructions available on the openthread.io website.
- **Join the Network**
  - For more detials, please check the specific section and the step by step instructions available on the openthread.io website.
- Ping the External Internet
  - For more details, please check the specific section and the step by step instructions available on the openthread.io website.

11.4 Over the Air Updates (OTA)

This chapter provides an overview of the Over the Air (OTA) process and usage for the JN5189 OpenThread stack. The transfer of a new OTA image can be sent from a server (the OpenThread Border Router) to the clients via multicast messages. At the end of the process, the clients that still need image fragments to complete the new OTA image will receive those fragments via unicast messages from the server. Also, the server supports the transfer via unicast messages. After advertising that a new image binary is available, the clients request image blocks until they receive the whole binary file.

The user can also initiate or terminate the OTA process using the Web Interface of the OpenThread Border Router.

11.4.1 OTA file format

The OTA file format is composed of a header followed by a number of sub-elements. The header describes general information about the file such as version, the manufacturer that created it, and the device it is intended for. Sub-elements in the file may contain upgrade data for the embedded device, certificates, configuration data, log messages, or other manufacturer-specific pieces.

Table 14. OTA file format using CRC

Name	Header	ImageTag	Image	Bitmap tag	Bitmap	CRC tag	CRC
Size	60 bytes	6 bytes	Variable	6 bytes	32 bytes	6 bytes	4 bytes

The bitmap sub-element of an OTA cluster image file indicates the sectors that should or should not be erased and reprogrammed in the internal flash. Setting a bit to 1 allows the erasure of the corresponding flash sector. If a bit is set to 0, the corresponding flash sector is protected.

11.4.2 OTA Commands

11.4.3 General considerations

- The initial network must be deployed with the OTA client feature enabled (define in config.h and linker flag set):

— the initial image that is on the REED/ED device needs to have gEnableOTAClient_d = 1.

— the linker flag gUseInternalStorageLink_d needs to be set to 1 for the initial OTA client deployment with internal flash. For external flash usage, see chapter 11.5.

11.4.4 gOtaCmd_ImageNotify_c (/otaserver) OTA Image Notify

- Sent unicast or multicast, the Image Notify command represents the way by which the server alerts clients that a new image is available.
- URI-Path: NON POST coap://<dest>/otaserver

11.4.5 gOtaCmd_QueryImageReq_c (/otaclient) OTA Query Image Request

- Automatically sent when the client receives an Image Notify Command.
- URI-Path: NON GET coap://<dest>/otaclient

11.4.6 gOtaCmd_QueryImageRsp_c (/otaserver) OTA Query Image Response

- The server response for Query Image Req command. Based on the status parameter, the OTA client decides whether or not to start downloading the new image.
- URI-Path: NON POST coap://<dest>/otaserver

11.4.7 gOtaCmd_BlockReq_c (/otaclient) OTA Block Request

- This command is used to download the new image, by requesting a specific block. This command uses a socket open on a port that is announced by the Server on a ML_EID address.
- Default port: 0xF0BE

11.4.8 gOtaCmd_BlockRsp_c (/otaserver) OTA Block Response

- This is the server response for Block Req command. Based on the status parameter, the OTA client decides whether or not to continue the download procedure. This command uses a socket open on a port that is announced by the Server on a ML_EID address. The response is sent on the port on which the command is received.
- Default port: 0xF0BE

11.4.9 gOtaCmd_UpgradeEndReq_c (/otaclient) OTA Upgrade End Request

- This command is used to complete the transfer
- URI-Path: NON GET coap://<dest>/otaclient

11.4.10 gOtaCmd_UpgradeEndRsp_c (/otaserver) OTA Upgrade End Response

- This is the server response for Upgrade End Req command and contains the delay used by the client before a re-flash/reboot to the new image (only if it detects the transfer procedure has been successful).
- URI-Path: NON POST coap://<dest>/otaserver

11.4.11 gOtaCmd_ServerDiscovery_c (/otaclient) OTA_Server_discovery

- This command is sent multicast and represents the way to discover a server by requesting a specific manufacturer and image type. A server will respond with an Image Notify unicast if success. The OTA Server discovery feature is disabled if the data regarding the OTA server (like the server's short address) is contained in a Service TLV in Network Data packets.
- URI-Path: NON POST coap://<dest>/otaclient

11.4.12 OTA Process Diagrams

Figure 27. OTA Server Discovery

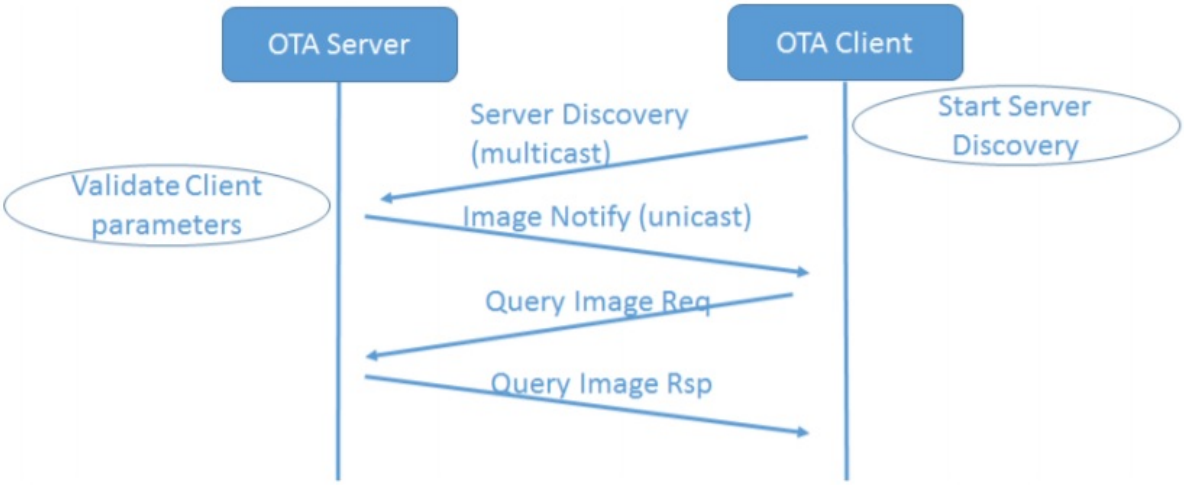


Figure 28. OTA Multicast

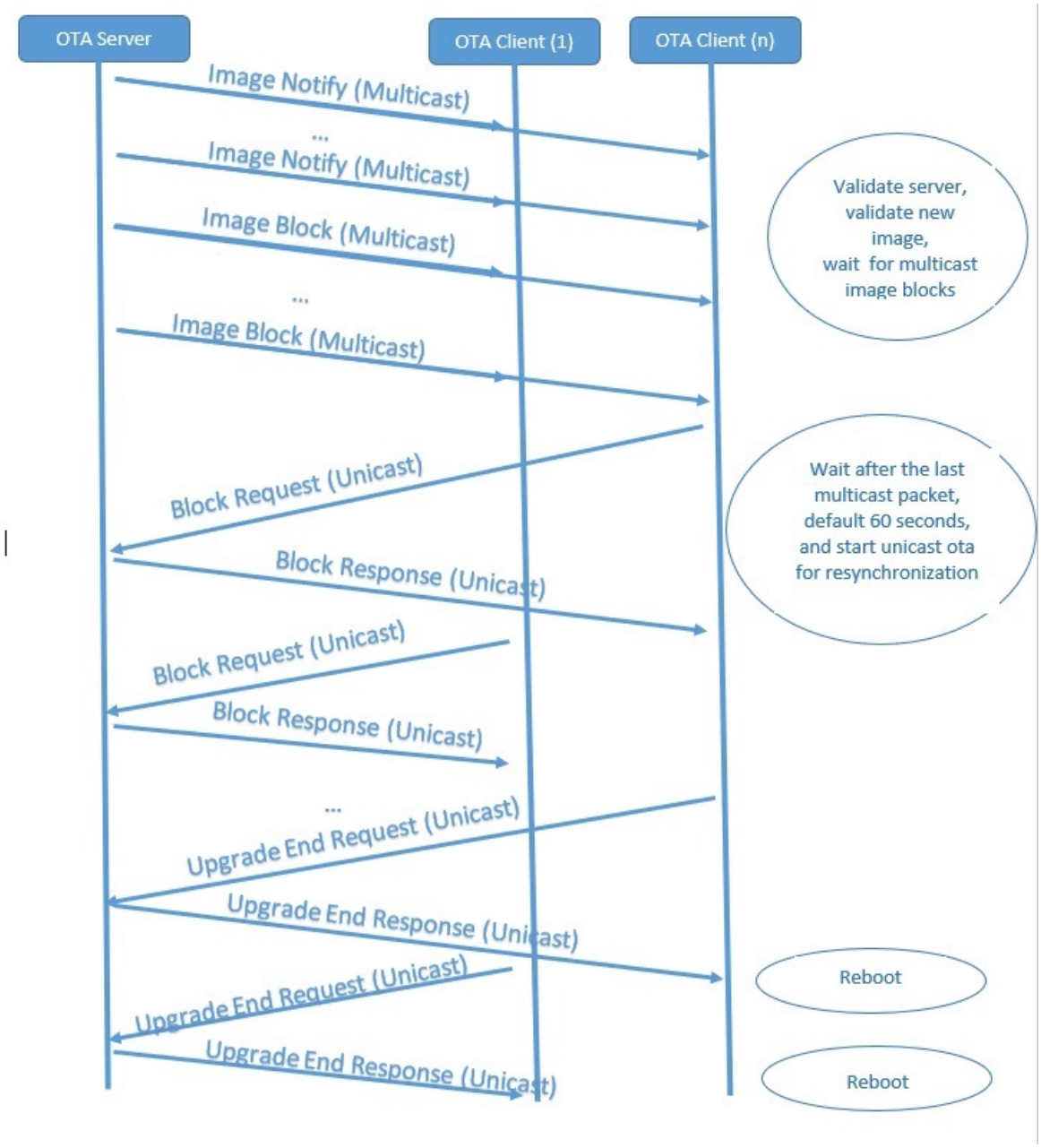
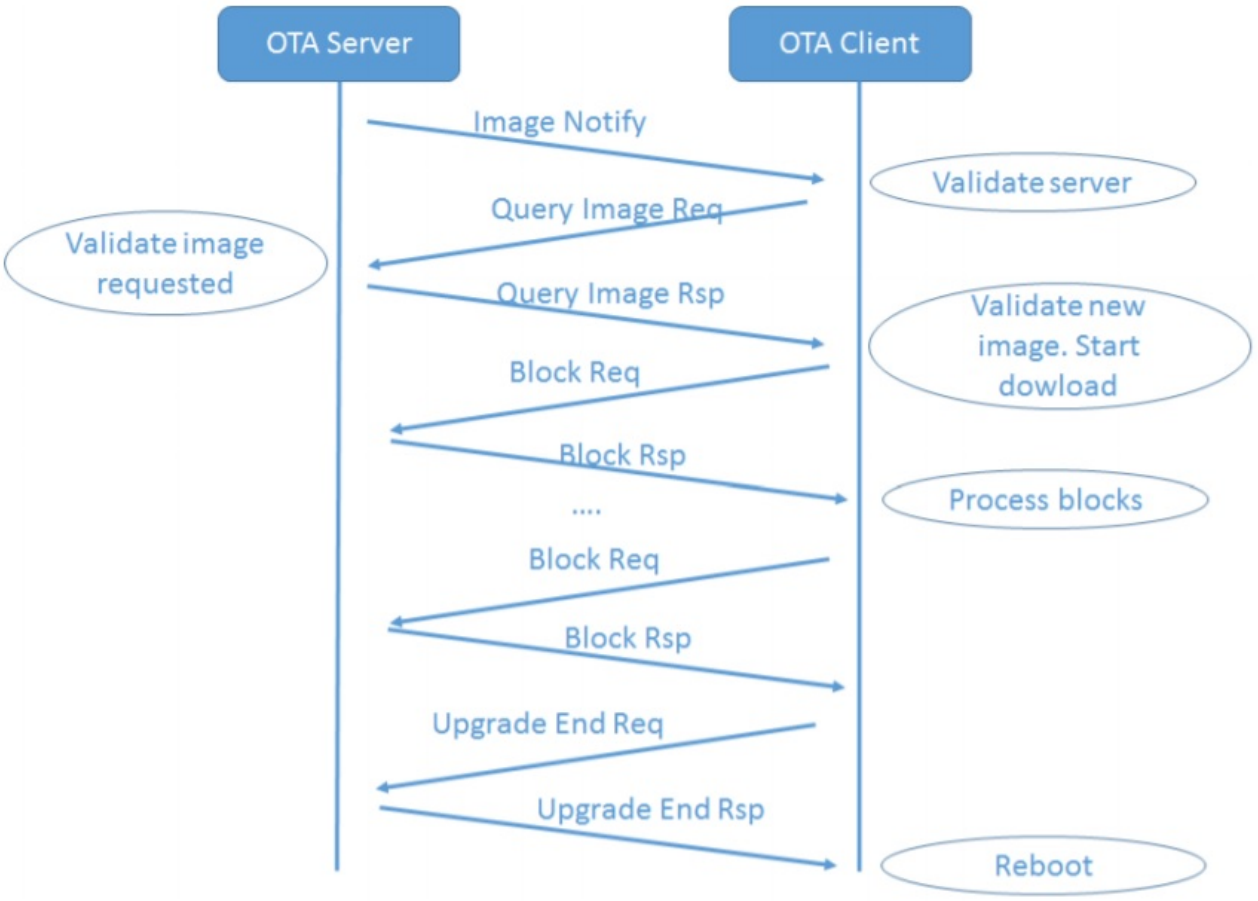


Figure 29. OTA Unicast



11.5 Step by step instructions for OTA upgrade

The current OTA implementation uses the JN5189 DK6 board's external flash, the 8 Mbit ultra-low power flash memory from Macronix, MX25R8035F.

The OTA client application with external flash uses also a second stage bootloader (SSBL). The binary ssbl.bin can be found in `<sdk_package>\boards\jn5189dk6\wireless_examples\framework\ssbl\binary`.

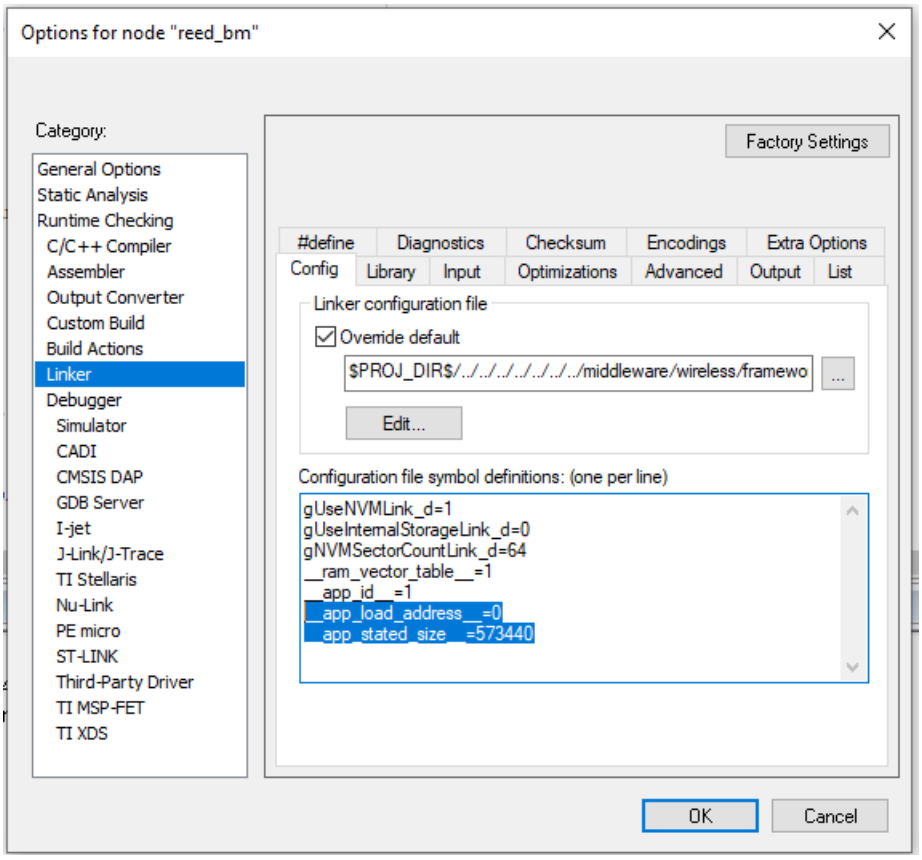
Support for OTA updates in heterogeneous Thread networks for REEDs and EDs is available. This feature is configured in the binary through the config.h file. The following defines control this feature (example is for REED project):

```
#define gOtaCurrentImageType_c       0x00, 0x00
#define gOtaCurrentImageTypeNo_c     0x0000
```

In order to build the OTA client application, the user can use IAR or MCUXpresso IDE. The following steps must be performed:

- In the project's config.h file, change the gEnableOTAClient_d to 1
- The project's linker symbol definitions must be changed in order to have the application written at the correct offset, after the SSBL:
  - In IAR, right click on the project's name in the workspace and go to Options. Go to Runtime Checking -> Linker (as in the figure below) and do the following changes:
    - Erase the "__app_load_address__=0" line
    - Replace "__app_stated_size__" with "__app1_size__"

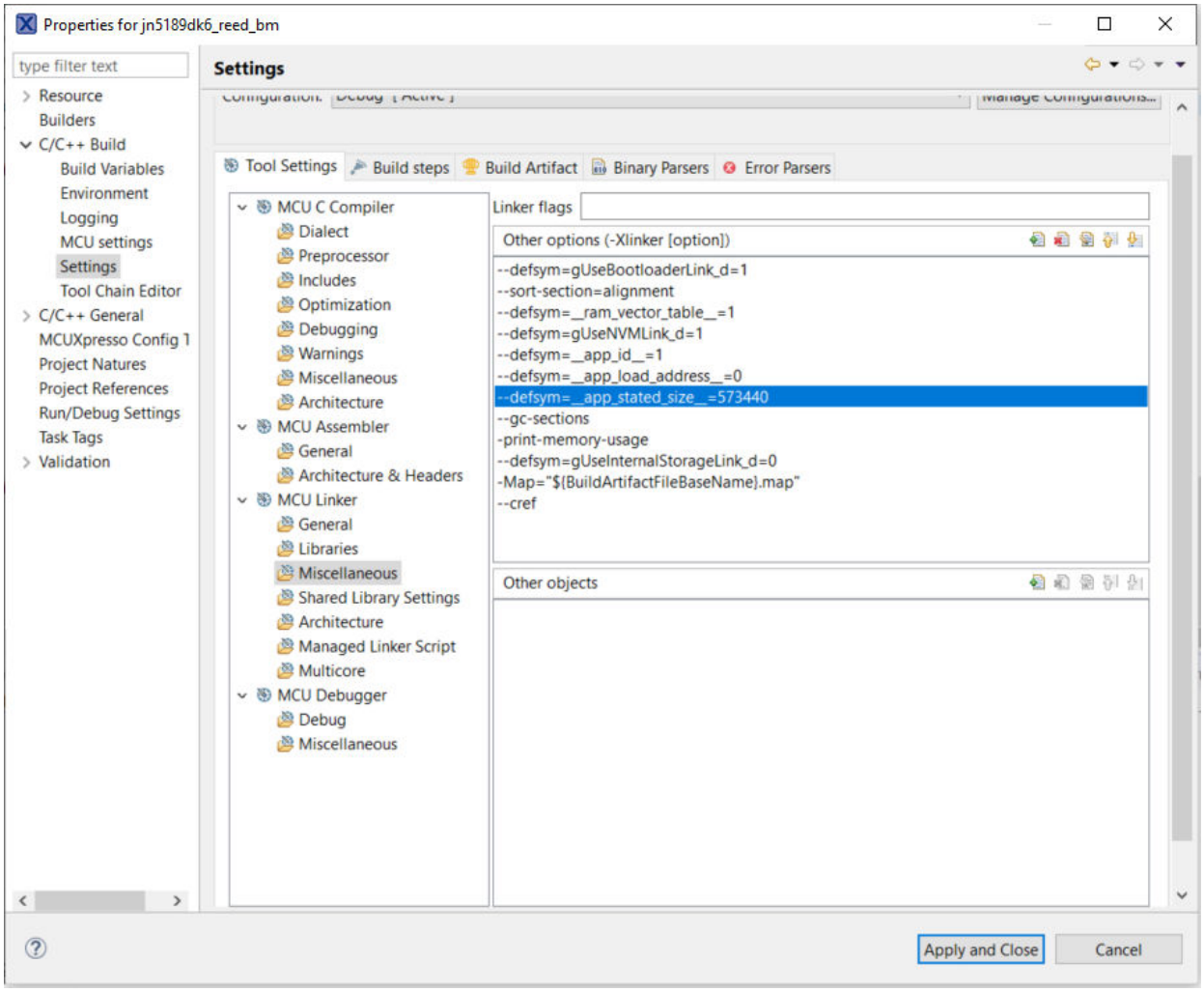
Figure 30. IAR Linker Options



— In MCUXpresso IDE, right click on the project's name in the Project Explorer and go to Properties. Go to C/C++ Build->Settings->MCU Linker->Miscellaneous (as in the figure below) and do the following changes:

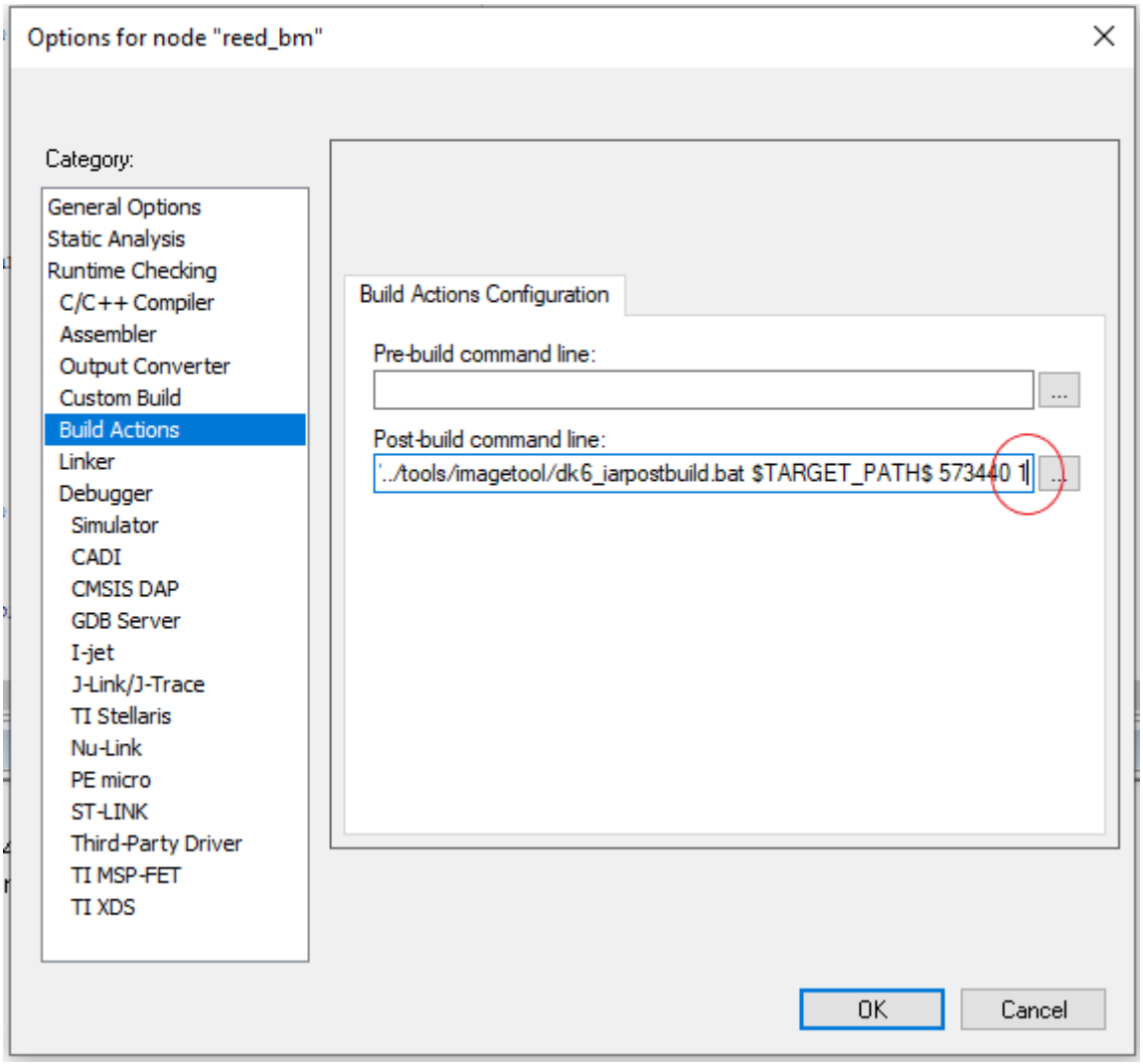
- Erase the "--defsym=__app_load_address__=0" line
- Replace "__app_stated_size__" with "__app1_size__"

Figure 31. MCUXpresso Linker Options



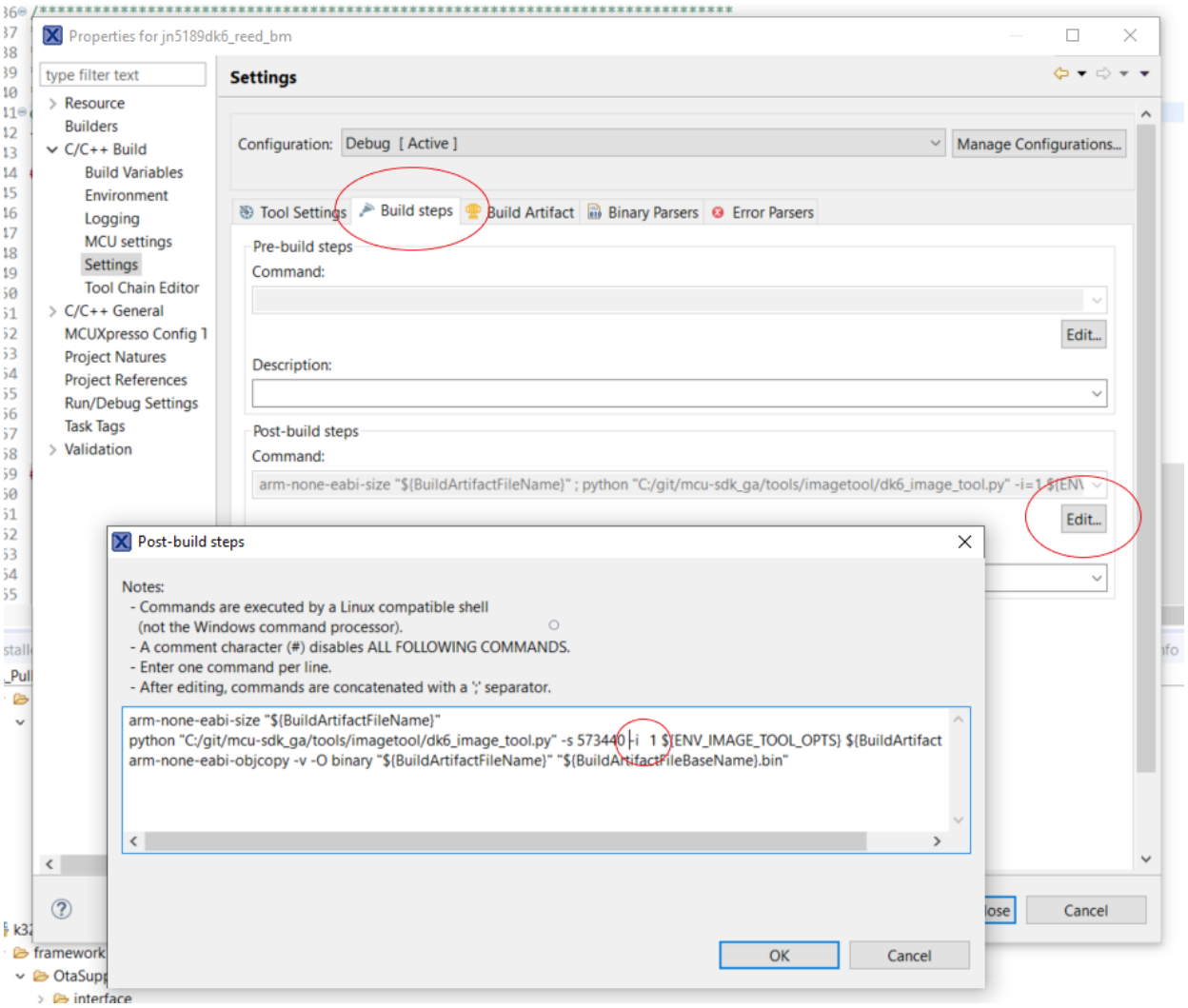
- The post build step of the project must be configured to include the application ID in the process.
  - From IAR's project options, go to Runtime Checking->Build Actions and add a "1" (encircled in red) at the end of the text in "Post-build command line", after the stated size (573440 in the example, as shown in the picture below) and click OK.

Figure 32. IAR Post-build Options



— From MCUXpresso's project Properties, go to C/C++ Build->Settings and click on the "Build Steps" tab. Click on the Edit button below the Command line in Post-Build steps and add "-i 1" after the stated size (-s 573440 in the example, as shown in the picture below) and click OK.

Figure 33. MCUXpresso Post-build Options



- Build the project

In order to flash the OTA client application to the board for the first deployment, the user requires the installation of DK6 Production Flash Programmer, found in tools/JN-SW-4407-DK6-Flash-Programmer. Instructions to install the program can be found in the folder.

From the installation folder of the DK6 Programmer, the following steps must be used to be able to flash and run the OTA client application (board must be programmed through FTDI port, replace <COMPORT> with corresponding COM value found in device manager):

1. Erase the device's flash memory

   ```
   DK6Programmer.exe -V 5 -P 1000000 -s <COMPORT> -e Flash
   ```

2. Provision the image directory. This step ensures that the SSBL knows the information about the application binary, such as the start address of the application, the size of the image (in pages), flags associated to the image and the type of image. This reflects the following structure found in rom_psector.h.

   ```
   typedef struct {
                   uint32_t img_base_addr; /* blob start address */
                   uint16_t img_nb_pages ; /* nb of pages used in flash by the blob */
                   uint8_t flags;          /* flags associated to the blob */
   ```

```
                uint8_t img_type;       /* blob position in flash */
} image_directory_entry_t;
```

In order to calculate the img_nb_pages, user must take the stated size value (in our example, 573440) and divide it by the number of bytes in one sector (for JN5189, this is 512). 573440/512=1120 (0x0460 in hexadecimal). The hexadecimal value must have its endianness changed (0x0460 becomes 0x6004).

The final command to the programmer is:

```
DK6Programmer.exe -V5 -s <COMPORT> -P 1000000 -w
PSECT:64@0x160=004000006004010100000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000
```

3. Flash the SSBL

```
DK6Programmer.exe -V2 -s <COMPORT> -P 1000000 -p FLASH@0x00=<sdk_package>\boards
\jn5189dk6\wireless_examples\framework\ssbl\binary\ssbl.bin
```

4. Flash the OTA client application binary

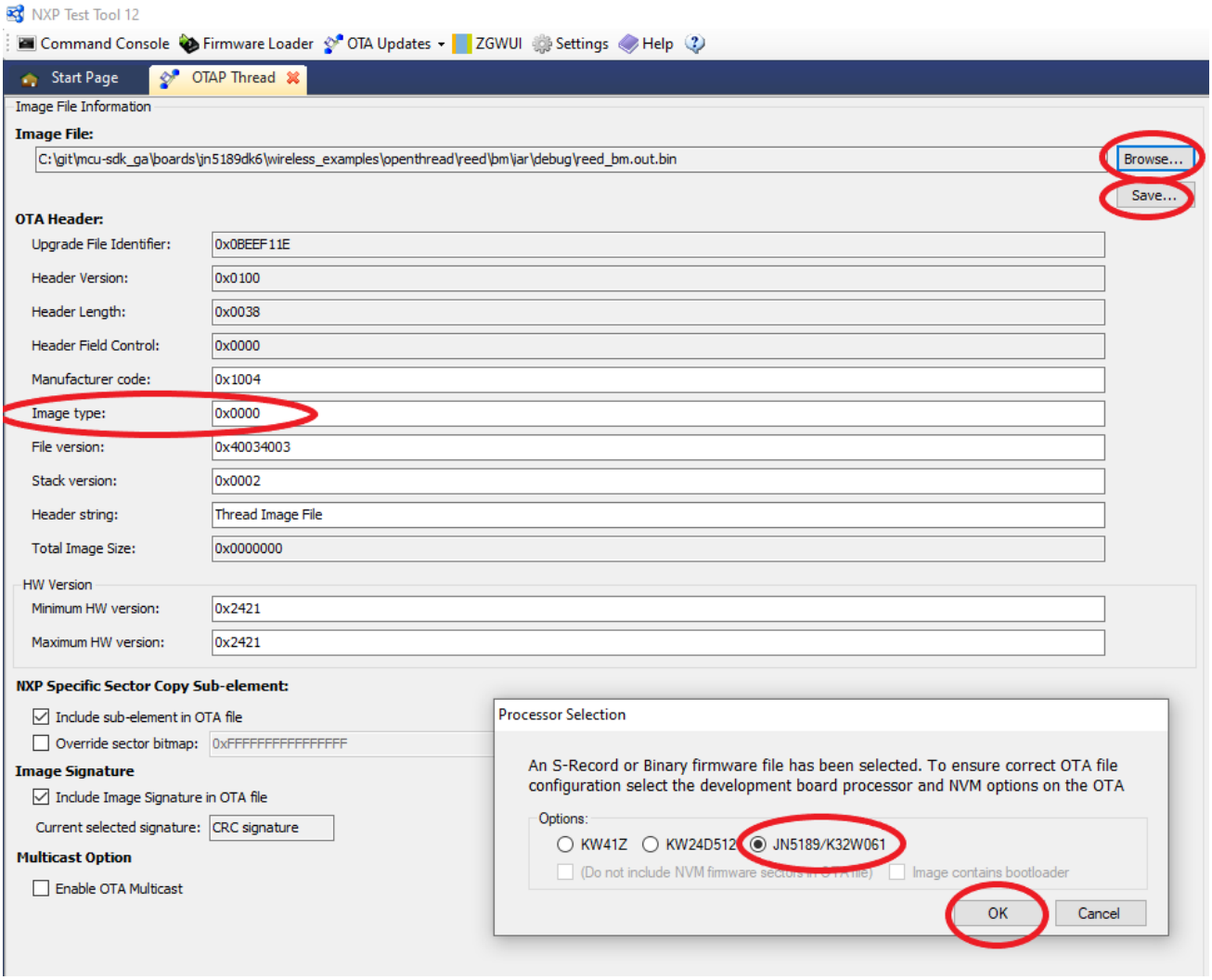
```
IAR binary: DK6Programmer.exe -V5 -s <COMPORT> -P 1000000 -p FLASH@0x4000=<sdk_package>\boards
\jn5189dk6\wireless_examples\openthread\reed\bm\iar\debug\reed_bm.out.bin
```

```
MCUXpresso binary: DK6Programmer.exe -V5 -s <COMPORT> -P 1000000 -p FLASH@0x4000=<workspace>\
jn5189dk6_reed_bm\Debug\jn5189dk6_reed_bm.bin
```

In order to obtain the binary for the OTA updates process, after the build step, user needs to use the associated NXP's Test Tool application and follow these steps:

- In Test Tool's OTA Updates->OTAP Thread, load the image by clicking Browse, select the appropriate board (JN5189/K32W061) and click OK.
- User can change the image type text box according to the project's *gOtaCurrentImageTypeNo_c* define (for REED is 0x0000 and for ED is 0x0001) in order to update a specific device type.
- Save the newly created OTA binary by clicking on Save button.
- Rename the saved file's extension to .bin.

Figure 34. NXP Test Tool OTA Image Creation



On the Border Router:

- Move the OTA image file using WinSCP or similar program to UDOO board. Then, move this file to the running container using the command:

```
sudo docker cp ota.bin  container_id:/home/ota.bin
```

  The id of the running container can be obtained using the command:

```
sudo docker container ls
```

- All the following commands will be executed inside the docker container, so the first step is to get a shell to the running container:

```
 sudo docker exec container_id /bin/bash
```

- In order to start the OTA updates process, the user can use the GUI or the wpanctl shell interface.
- The available wpanctl shell commands for OTA updates can be obtained using the following command:

```
sudo wpanctl -h
                wpanctl 0.08.00d (; Jan 25 2020 16:08:37)
                …
```

```
                start-ota          Start NXP OTA process
                stop-ota           Stop NXP OTA process
                get-ota-status     Get NXP OTA status
                …
```

- To start the OTA updates process, enter the following command:

```
sudo wpanctl start-ota -o T -p path
```

  where T needs to be replaced with 0 for OTA unicast process and 1 for OTA multicast process and path needs to be replaced with a valid path to the binary (for example /home/udooer/reed.out.bin)

- To stop the OTA updates process, enter the following command:

```
sudo wpanctl stop-ota
```

- To get the status of the OTA updates process, enter the following command:

```
sudo wpanctl get-ota-status
```

  — Examples of outputs:

    ◦ During multicast process:

```
OTA multicast process in progress, percentage done: 89 %
```

    ◦ During unicast process:

```
OTA unicast process
           Client ID: A88E, percentage: 33 %
```

  — Wait until the OTA process is finished. This can be indicated by the fact that the boards in the network are in Disconnected mode. Check the new image by reconnecting to the network and checking the changes.

Known limitations:

— Only OTA unicast is supported in this release.

— The server doesn't know how many clients are in the network and doesn't know when to reset the OTA server internal state. User must issue a *sudo wpanctl stop-ota* command after all boards finish the OTA update process in order to reset the server's state. This is needed before starting another OTA update process.

— Because of the usage of external flash memory through SPIFI, the D1 LED from the OM15082 board is unavailable for the application.

Chapter 12
Revision history

Table 15. Revision history

Revision number	Date	Substantive changes
0	11/2019	Updates for JN5189 Open Thread PRC1 milestone
1	03/2020	Updates for JN5189 RFP

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 10 March 2020

Document identifier: UM11324

arm