Application Note: JN-AN-1243

DK006: JN5189 / JN5188 / K32W061 / K32W041/K32W041A/K32W041AM

Zigbee 3.0 Base Device

This Application Note applies to the JN5189, JN5188, K32W061,K32W041,K32W041A and K32W041AM Zigbee 3.0 wireless microcontrollers used with the DK006 (Development Kit) platform. These microcontrollers will be referred to as the DK006 microcontrollers throughout this document.

The *Zigbee 3.0 Getting Started Application Note [JN-AN-1260]* contains instructions for installing MCUXpresso, DK006 microcontroller SDKs and other required tools to develop with this Application Note.

This Application Note provides example applications to demonstrate the features and operation of the Base Device in a Zigbee 3.0 network that employs the NXP DK006 Zigbee 3.0 microcontrollers. An example application can be employed as:

- **A demonstration using the supplied pre-built binaries that can be run on nodes of the DK006 Development Kits**
- **A starting point for custom application development using the supplied C source files and associated project files**

The devices described in this Application Note provide the standard mandatory features of the Zigbee Base Device Behaviour Specification for a Coordinator, Router and End Device. They do not implement a complete real device but provide the mandatory features of the Zigbee Base Device on which an application can be built and further developed.

The Zigbee 3.0 nodes of this Application Note can be used in conjunction with nodes of other Zigbee 3.0 Application Notes, available from the NXP web site.

1 Introduction

A Zigbee 3.0 wireless network comprises a number of Zigbee software devices that are implemented on hardware platforms to form nodes. This Application Note is concerned with implementing the Zigbee Base Device on the NXP DK006 platform.

This Application Note provides example implementations of the following Zigbee logical device types:

- Coordinator
- Router
- End Device

The examples of the above device types are not real-world devices, like those defined in the Zigbee Lighting & Occupancy Device Specification, but provide the basic behaviour required by the Zigbee Base Device Behaviour Specification. They are provided as templates on which to base further development into real physical devices. The Zigbee Base Device is introduced and detailed in the *Zigbee 3.0 Devices User Guide [JN-UG-3131]*.

The Zigbee Base Device Behaviour Specification provides definitions, procedures and methods for forming, joining and maintaining Zigbee 3.0 networks. It also defines the method for service discovery, in which a client and server of an operational cluster are bound together in order to achieve the functionality of the physical devices.

Note: If you are not familiar with Zigbee 3.0, you are advised to refer the *Zigbee 3.0 Stack User Guide [JN-UG-3130]* for a general introduction.

2 Development Environment

2.1 Software

In order to use this Application Note, you need to install the Eclipse-based Integrated Development Environment (IDE) and Software Developer's Kit (SDK) that are appropriate for your DK006 wireless microcontroller:

- MCUXpresso IDE
- JN518x Zigbee 3.0 SDK
- K32W061/K32W041A/K32W041AM Zigbee 3.0/Bluetooth SDK

The MCUXpresso software and installation instructions are described in the *Zigbee 3.0 Getting Started Application Note [JN-AN-1260]*. The *JN-AN-1243 Release Notes* (included in this folder) indicate the versions of MCUXpresso and SDK that you will need to work with this Application Note.

The DK006 wireless microcontroller specific resources and documentation are available via the MCUXpresso website to authorised users.

Note: Prebuilt application binaries are supplied in this Application Note package see the *Zigbee 3.0 Getting Started Application Note [JN-AN-1260]* for instructions on how to compile the application binaries on your own system.

2.2 Hardware

Hardware kits are available from NXP to support the development of Zigbee 3.0 applications. The following kits provide a platform for running these applications:

- JN518x-DK006 Development Kit, which features JN5189 devices
- IoT_ZTB-DK006 Development Kit, which features K32W061 devices

This kit supports the NFC commissioning of network nodes (see Router Functionality and End Device Functionality for details).

3 Application Note Overview

The example applications provided in this Application Note are listed in the following table.

Application	Device Type
Coordinator	Coordinator/Trust Centre
Router	Router
End Device	End Device (sleeping)

Table 1: Example Applications and Device Types

For each application, source files and pre-built binary files are provided in the Application Note ZIP package. The pre-built binaries can be run on components of the DK006 Development Kits.

- To load the pre-built binaries into the evaluation kit components and run the demonstration application, refer to Loading the Applications.
- To start developing you own applications based on the supplied source files, refer to Developing with the Application Note.

3.1 NFC Hardware Support

All the microcontrollers supplied with the DK006 Development Kits contain an internal NFC tag (NTAG) that connects to an NFC antenna on the OM15076-3 Carrier Boards. NTAG enabled applications use this internal NTAG by default.

The OM15076-3 Carrier Boards are also fitted with an external NTAG which can be used instead of the internal NTAG or with chip variants that do not have an internal NTAG (these are not supplied in the DK006 Development Kits). Minor hardware modifications are required to the OM15076-3 Carrier Boards to enable this external NTAG.

DK006 Development Kits for the development of Zigbee 3.0 applications provide the possibility of network commissioning through Near Field Communication (NFC). This Application Note also provides experimental functionality for updating device software through NFC.

4 Running the Demonstration Application

This section describes how to use the supplied pre-built binaries to run the example applications on components of a DK006 kit. All the applications run on a DK006 microcontroller module on a OM15076-3 Carrier Board, fitted with a specific expansion board.

4.1 Loading the Applications

The table below lists the application binary files supplied with this Application Note and indicates the DK006 kit components with which the binaries can be used. These files are located in the **Binaries** folder of the Application Note. Each device has its own folder (matching the name of the binary file), the file/folder names indicate the included functionality.

The binaries/folders with a **Ssbl** prefix are intended for use in image switching applications in conjunction with the Second-Stage Bootloader application. These binaries are included for use by other Application Notes that describe image switching and the use of these binaries in detail.

DK006 Hardware	Binary Files
OM15080 USB Dongle	**Coordinator_GpProxy_DONGLE**
OM15076-3 Carrier Board OM5578 NFC Controller Board	**Coordinator_NciIcode_NciOta_GpProxy_OM5578**
OM15076-3 Carrier Board OM15082-2 Generic Expansion Board	**Coordinator_GpProxy_OM15082** **EndDevice_OM15082** **EndDevice_NtagNwk_NtagOta_Ota_RamOpt_OM15082_V1** **Ssbl_Coordinator_GpProxy_OM15082** **Ssbl_EndDevice_NtagNwk_OM15082**
OM15076-3 Carrier Board OM15081-2 Lighting/Sensor Expansion Board	**Router_GpProxy_OM15081** **Router_NtagNwk_NtagOta_GpProxy_Ota_OM15081_V1** **Ssbl_Router_NtagNwk_GpProxy_OM15081**

Table 2: Application Binaries and Hardware Components

A binary file can be loaded into the Flash memory of a DK006 device using the *DK6 Production Flash Programmer [JN-SW-4407]*. This software tool is described in the *DK6 Production Flash Programmer User Guide [JN-UG-3127]*.

Note: You can alternatively load a binary file into a DK006 device using the Flash programmer built into the relevant IDE.

To load an application binary file into a DK006 module on a Carrier Board of a kit, follow the instructions below:

1. Connect a USB port of your PC to the USB Mini B port marked “FTDI USB” on the Carrier Board (next to the 34-pin header) using a ‘USB A to Mini B’ cable. At this point, you may be prompted to install the driver for the USB virtual COM port.
2. Determine which serial communications port on your PC has been allocated to the USB connection.
3. Put the DK006 device into programming mode by holding down the ISP button while pressing and releasing the RESET button.
4. Run the batch (**.bat**) file provided alongside the binary (**.bin**) file to erase the contents of the flash memory including the persistent data stored by the PDM and program the binary file into flash memory. The batch file will prompt for the COM port number to use.
5. Once the download has successfully completed, reset the board or module to run the application.

The batch files require the installation of the DK6 Production Flash Programmer to have been completed following the instructions in the *Zigbee 3.0 Getting Started Application Note [JN-AN-1260]*.

4.2 Coordinator Functionality

The functionality of the Coordinator application is described and illustrated below.

The Coordinator is responsible for initially forming the network and then, via the Trust Centre functionality, managing which other devices are allowed to join the network and distributing security materials to those that are allowed to join. The Coordinator supports the mandatory clusters and features of the Base Device as defined in the Zigbee Base Device Behaviour Specification.

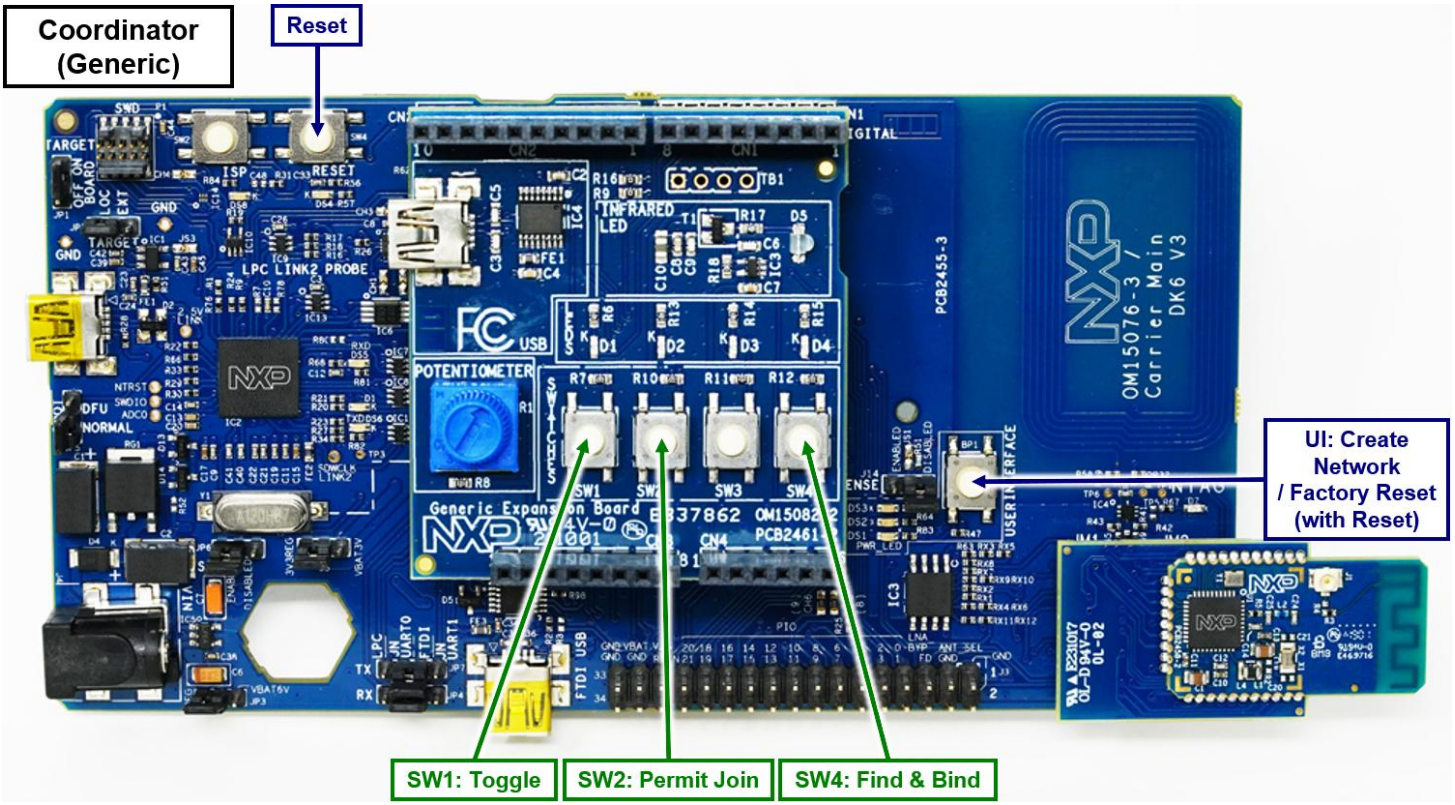
For the purpose of demonstrating the 'Finding and Binding' functionality, the Coordinator also supports the On/Off Cluster as a client.

The Coordinator provides two methods for triggering its functionality:

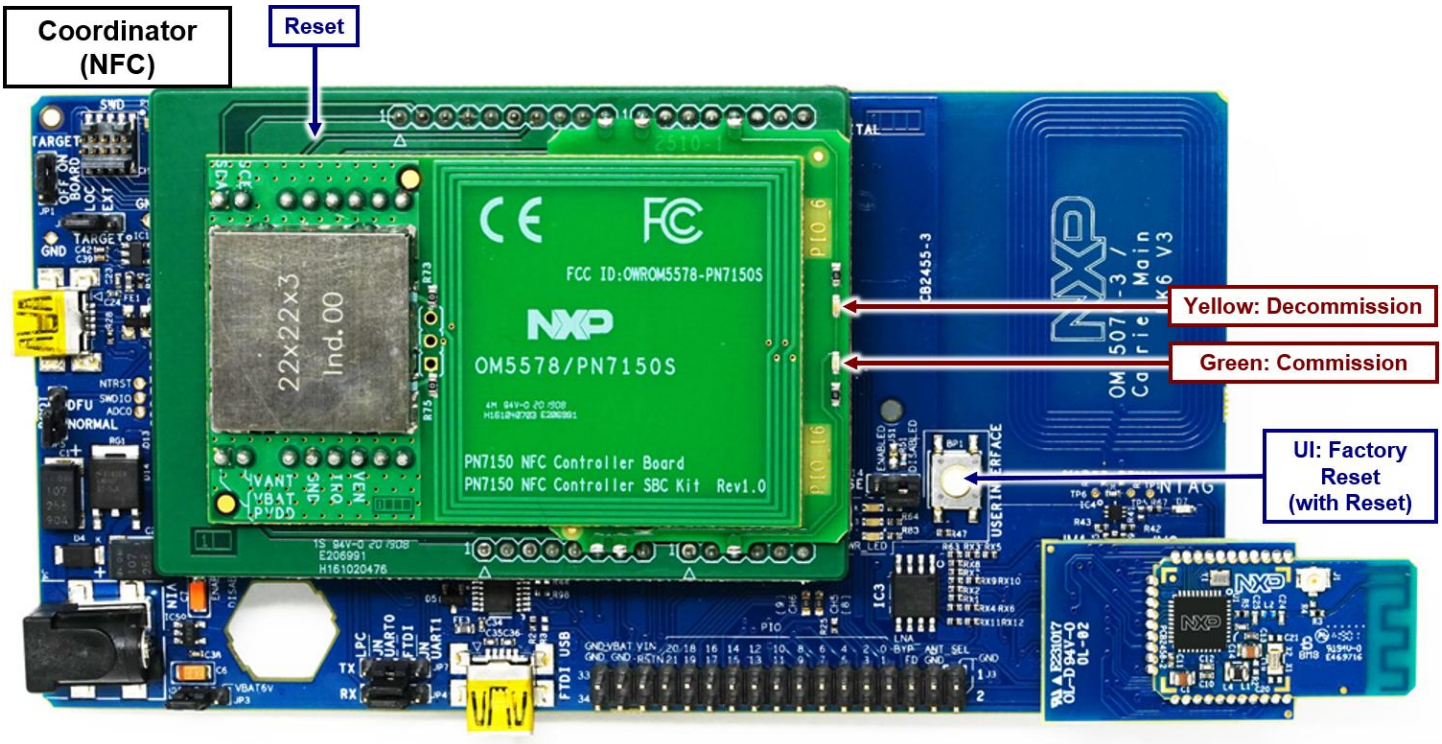
- Using commands from a serial interface on a host terminal connected to the Coordinator hardware – if the Coordinator application is programmed into the DK006 USB Dongle, this is the only interface available. The serial interface in not case-sensitive. For a summary of the serial interface, refer to Summary of Serial Interface Commands.
- Using the push-buttons on the OM15082 Generic Expansion Board – this is only available when a Coordinator application is programmed into a DK006 module on a Carrier Board fitted with the OM15082 Generic Expansion Board.

The Coordinator application can be run on three different hardware combinations, each offering slightly different interfaces:

- USB Dongle (PCB2459): Uses serial interface commands only, NFC joining is not supported.
- Carrier Board (OM15076) with Generic Expansion Board (OM15082): Uses serial interface commands and/or buttons, NFC joining is not supported:



- Carrier Board (OM15076) with NFC Controller Board (OM5578): Uses serial interface commands only, NFC joining is supported:



4.2.1 Forming a Network

A network can be formed from a factory-new Coordinator (Network Steering while not on a network) in either of the following ways, depending on the hardware being used:

- Press the button USER INTERFACE on the OM15076 Carrier Board.
- Enter “form” on the serial interface (Dongle or Carrier Board).

The Coordinator will then start a network. Using a packet sniffer (for example, on a USB Dongle), the periodic link status messages can then be observed on the operational channel.

4.2.2 Allowing Other Nodes to Join (Network Steering)

Once a network has been formed, the network must be opened to allow other devices to join (Network Steering while on a network) in either of the following ways, depending on the hardware being used:

- Press the button SW2 on the OM15082 Generic Expansion Board.
- Enter “steer” on the serial interface (Dongle or Carrier Board).

The Coordinator will then broadcast a Management Permit Join Request to the network to open the ‘permit join’ window for 180 seconds. The Network Steering process (for devices not on a network) can now be triggered on the devices that are to join the network.

4.2.3 Allowing Other Nodes to Join (NFC Joining)

Once a network is formed, other devices can join using NFC when using the NFC Controller Board (OM5578). The joining device must have appropriate NTAG hardware and supporting software. To join a device using NFC perform the following steps:

- Enter “nfc join” on the serial interface, the Coordinator is in this mode by default
- The green LED on NFC Controller Board will flash slowly when NFC joining is enabled
- Ensure the joining device has been powered on after programming to place the correct data into the NTAG
- Place the joining device in range of the NFC Controller Board
- The green LED will flash quickly while the data exchange takes place, then remain lit when complete (or be extinguished if the NTAG does not contain the correct data)
- When the joining device is removed from the field it will join the network and the green LED will return to flashing slowly

Devices can be removed from the network in a similar way using the “nfc leave” command on the serial interface. The yellow LED on the NFC Controller Board is used to provide similar feedback in this mode.

The Coordinator’s NFC functionality can be turned off using the “nfc disable” command on the serial interface. The LEDs will remain off when in this mode.

4.2.4 Binding Nodes

‘Finding and Binding’ is the process whereby controlling devices find controlled devices by matching operational clusters and create entries in the Binding table. The Coordinator supports Finding and Binding as an ‘Initiator’ that tries to find targets to bind to. For the purpose of the demonstration, the Coordinator supports the On/Off Cluster as a client, so the Finding and Binding process will look for devices that support the On/Off cluster as a server in order to create bindings.

To start Finding and Binding as an Initiator, first trigger Finding and Binding on any ‘Target’ device (such as the Router in this Application Note) and then do either of the following on the Coordinator (Initiator):

- Press the button SW4 on the OM15082 Generic Expansion Board.
- Enter “find” on the serial interface (Dongle or Carrier Board).

When Finding and Binding for a target has completed and a binding has been created, the Coordinator will send an Identify Off command to the target device, in order to signal completion of the process for the Target. Depending on the type of bindings being created (either unicast or groupcast), an Add Group command may be sent to the target device.

Reporting is a mandatory feature in Zigbee 3.0. A device wishing to receive periodic and on-change reports from an operational server should create a remote binding for itself on the target device. This Coordinator will send a Binding Request to a target with an On/Off cluster server. It will then receive periodic and on-change reports from that device, reporting the state of the OnOff attribute (0x0000) of the On/Off cluster. The frequency of the reports depends on the default report configuration of the individual target device. The device receiving the reports can request that this is changed by sending a Report Configuration command.

4.2.5 Operating the Device

The operational functionality of this device in this demonstration is provided by the On/Off cluster. You can now send an OnOff Toggle command to the bound devices (in the Binding table) in either of the following ways:

- Press the button SW1 on the OM15082 Generic Expansion Board.
- Enter “toggle” in the serial interface (Dongle or Carrier Board).

The effect that this command has on a bound device depends on the functionality related to the On/Off cluster on the device – for the Router in this demonstration, it will toggle a light (see Router Functionality).

4.2.6 Re-joining the Network

As a Coordinator, when this device is restarted in a state which is not factory-new, it will just resume operation in its previous state. All application, binding, group and network parameters are preserved in non-volatile memory.

4.2.7 Performing a Factory Reset

The Coordinator can be returned to its factory-new state (erasing all persistent data except the outgoing network frame counter) in either of the following ways, depending on the hardware being used:

- Hold down the USER INTERFACE button and press the RESET button on the Carrier Board
- Enter “factory reset” on the serial interface (Dongle or Carrier Board).

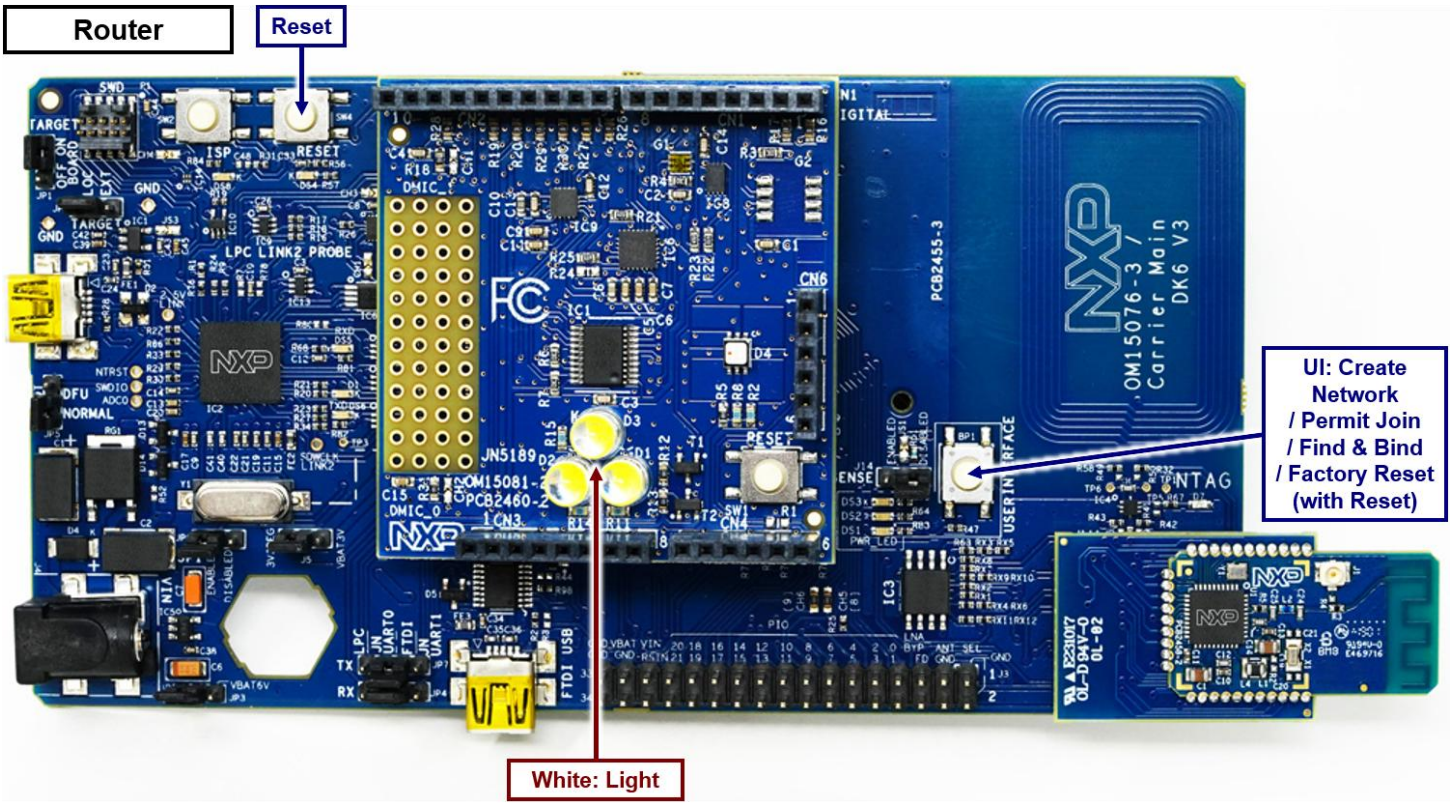
4.2.8 Summary of Serial Interface Commands

Button	Serial Command	Action
SW1	toggle	Sends an OnOff Toggle command to bound devices
SW2	steer	Triggers Network Steering for a device on the network
SW3	form	Triggers network formation for a device not on a network
SW4	find	Triggers Finding and Binding as an Initiator
Reset+USER INTERFACE	factory reset	Factory resets the device, erasing persistent data
Reset	soft reset	Triggers a software reset (no loss of data)
-	nfc join	Enable NFC joining (green LED flashes slowly)
-	nfc leave	Enables NFC leaving (yellow LED flashes slowly)
-	nfc ota	Enables experimental NFC OTA (green and yellow LEDs flash slowly.
-	nfc disable	Disables NFC functionality (LEDs off)
-	Print	Prints the Aps Key Table to the terminal
-	Code <MAC> <Install Code>	Provisions an install code into the Aps Key Table

The serial port is set up to use 115200 baud, 8 data bits, 1 stop bit, no parity. The serial commands are not case-sensitive. The install code may be entered as 16 hex bytes with either no separators or commas or colons.

4.3 Router Functionality

The functionality of the Router application is described and illustrated below.



The Router can either join an existing network or decide to form a distributed network itself for other nodes to join. For details of the differences between a Centralised Trust Centre Network and a Distributed Network, refer to the *Zigbee Devices User Guide [JN-UG-3114]*. The Router supports the mandatory clusters and features of the Base Device as defined in the Zigbee Base Device Behaviour Specification.

For the purpose of demonstrating the 'Finding and Binding' functionality, the Router also supports the On/Off Cluster as a server.

4.3.1 Forming or Joining a Network

The Router is capable of either joining an existing network or, in the absence of a network, to form a distributed network for other devices to join.

4.3.1.1 Joining an Existing Network using Network Steering

A factory-new Router can join an existing network once the network has been opened to accept new joiners (Network Steering for a device on a network). This is achieved as follows:

1. Trigger Network Steering on one of the devices already on the network.
2. Then reset (using the RESET button) or power-on the Router device.

This will cause the Router to start a network discovery and the associate process. Association is followed by an exchange of security materials and an update of the Trust Centre Link Key (if joining a Centralised Trust Centre Network).

If the join is unsuccessful, it can be retried by power-cycling again. Alternatively, the process for forming a distributed network can be followed, as described in Forming a Distributed Network.

4.3.1.2 Joining an Existing Network using NFC

A Router can join or move to an existing network by exchanging NFC data with either

- The *NciIcode* build of the Coordinator described above in Allowing Other Nodes to Join (NFC Joining).
- The *NciIcode* build of the Zigbee IoT Control Bridge, described in the Application Note *Zigbee 3.0 IoT Control Bridge [JN-AN-1247]*
- The Zigbee IoT Gateway Host, described in the Application Note *Zigbee IoT Gateway Host with NFC [JN-AN-1222]*

This provides a fast and convenient method to introduce new devices into such a network.

Instructions for this process are included in the above Application Notes (JN-AN-1243, JN-AN-1247, JN-AN-1222).

4.3.1.3 Forming a Distributed Network

The Router can form a distributed network in the absence of an open network to join. To achieve this on a factory-new device:

- Press the USER INTERFACE button on the Carrier Board (the same button is also used to start Network Steering as well as Finding and Binding, described below).

The Router will form a network with a random network key and begin operation. To allow other devices to join this network, follow the instructions in Allowing Other Devices to Join the Network.

4.3.2 Allowing Other Devices to Join the Network

Once the Router is part of a network, the network must be opened to allow other devices to join (Network Steering while on a network). To do this:

- Press the USER INTERFACE button on the Carrier Board (the same button is also used to start Finding and Binding, described in Binding Devices).

The Router will then broadcast a Management Permit Join Request to the network to open the ‘permit join’ window for 180 seconds. The Network Steering process (for devices not on a network) can now be triggered on the devices that are to join the network.

4.3.3 Binding Devices

The Router supports the On/Off cluster as a server and implements the Finding and Binding process as a Target. To trigger Finding and Binding as a target, do the following:

1. Press the USER INTERFACE button on the Carrier Boards of all the target devices (the same button also is also used to start Network Steering, described in Allowing Other Devices to Join the Network).
2. Start Finding and Binding on the Initiator device.

This will cause the Router to self-identify for 180 seconds, while the Initiator will try to find the identifying devices, query their capabilities and create bindings on those with matching operational clusters. As part of this process, the Router may receive an Add Group Command and/or a Binding Request Command.

Reporting is a mandatory feature in Zigbee 3.0. The Router supports the On/Off cluster as a server and the OnOff attribute of this cluster is a reportable attribute as defined in the Zigbee Base Device Behavior Specification. The Router holds a default configuration for reporting the state of the OnOff attribute. Once a device wishing to receive these periodic and on-change reports has created a remote binding, the Router will start to send reports to this bound device. The frequency of the reports depends on the default report configuration of the individual target device, 60 seconds in this case. The device receiving the reports can request that this is changed by sending a Report Configuration command.

4.3.4 Operating the Device

The operational functionality of this device in this demonstration is provided by the On/Off cluster. Since the device supports the On/Off cluster server, its operation is passive and it responds to commands sent by bound devices. It responds to an OnOff Toggle command from a bound controller device by toggling the white light on the OM15081 Lighting/Sensor Expansion Board.

4.3.5 Rejoining a Network

As a Router, when this device is restarted in a state which is not factory-new, it will just resume operation in its previous state. All application, binding, group and network parameters are preserved in non-volatile memory.

4.3.6 Performing a Factory Reset

The Router can be returned to its factory-new state (erasing all persistent data except the outgoing network frame counter) as follows:

- Hold down the USER INTERFACE button and press the RESET button on the Carrier Board.

The Router will then broadcast a Leave Indication on the old network, then delete all persistent data (except the outgoing network frame counter) and perform a software reset.

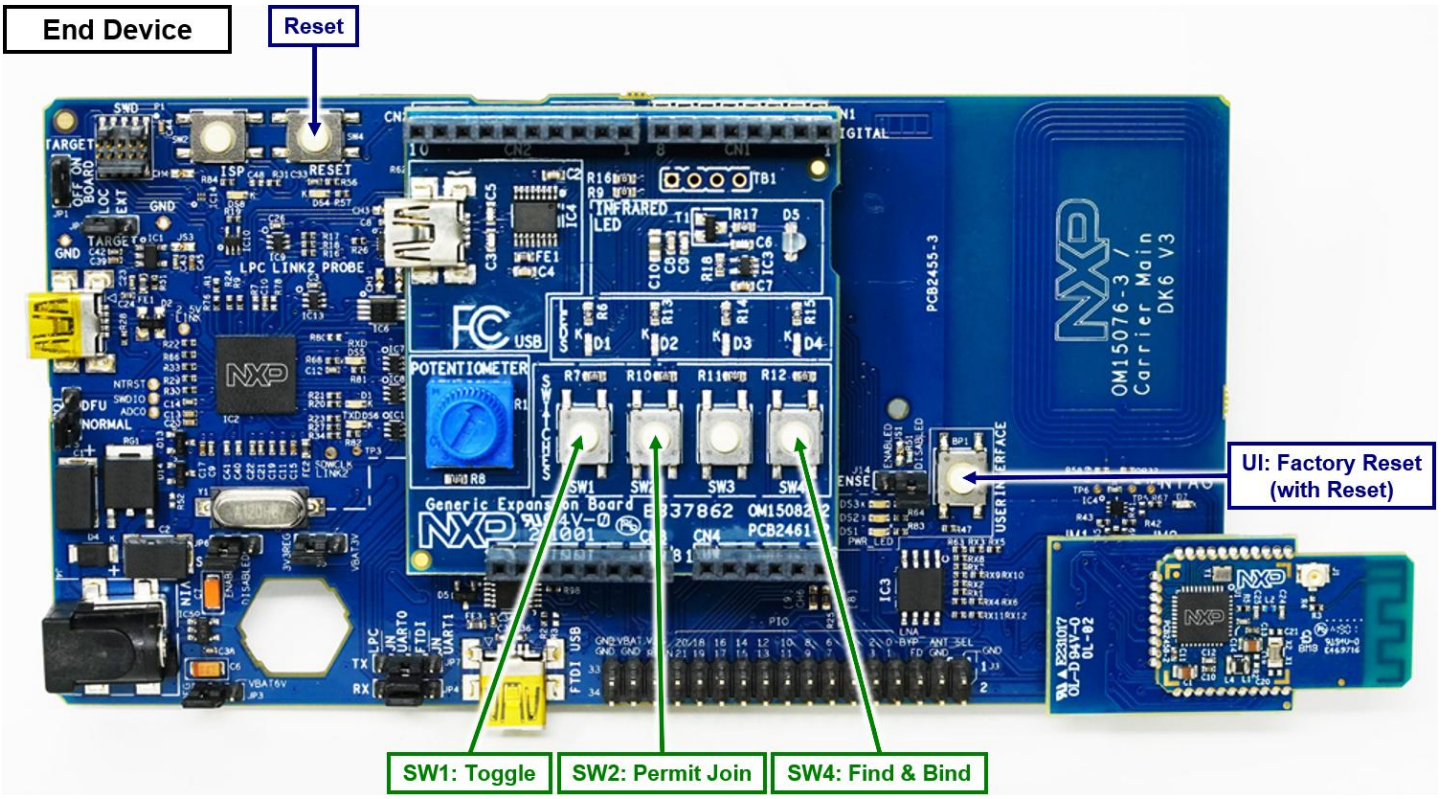
There are two supported over-the-air commands for removing a device from the network - these are:

- Network Leave Request without rejoin
- ZDO Management Network Leave Request without rejoin

The Reset command of the Basic cluster will cause the ZCL to be reset to its factory-new defaults, resetting all attributes and configured reports. This will not remove the device from the network - all network parameters, groups and bindings will remain in place.

4.4 End Device Functionality

The functionality of the End Device application is described and illustrated below.



The End Device is a sleepy, 'Rx Off when Idle' device. It is not capable of forming a network or being a parent to other devices joining a network. The End Device supports the mandatory clusters and features of the Base Device as defined in the Zigbee Base Device Behaviour Specification.

For purpose of demonstrating the 'Finding and Binding' functionality, the End Device also supports the On/Off cluster as a client.

All communications to/from the End Device are passed through its parent Coordinator or Router and the End Device must send periodic Poll Requests to the parent in order to receive any messages that may be waiting for it. The End Device implements a mixed sleep pattern - for a short period after initially waking, it will perform a series of warm sleep cycles from which it is woken on a timer or a DIO change, but after this it will enter deep sleep mode until woken by a DIO change.

4.4.1 Joining a Network

4.4.1.1 Joining an Existing Network using Network Steering

A factory-new End Device can join an existing network once the network has been opened to accept new joiners (Network Steering for a device on a network). This is achieved as follows:

1. Trigger Network Steering on one of the devices already on the network.
2. Press the button SW2 on the OM15082 Generic Expansion Board of the End Device.

4.4.1.2 Joining an Existing Network using NFC

An End Device can join or move to an existing network by exchanging NFC data with either

- The *NciIcode* build of the Coordinator described above in Allowing Other Nodes to Join (NFC Joining).
- The *NciIcode* build of the Zigbee IoT Control Bridge, described in the Application Note *Zigbee 3.0 IoT Control Bridge [JN-AN-1247]*
- The Zigbee IoT Gateway Host, described in the Application Note *Zigbee IoT Gateway Host with NFC [JN-AN-1222]*

This provides a fast and convenient method to introduce new devices into such a network.

Instructions for this process are included in the above Application Notes (JN-AN-1243, JN-AN-1247, JN-AN-1222).

4.4.2 Allowing Other Devices to Join the Network

Once the End Device is part of a network, the End Device can request that the network is opened to allow other devices to join (Network Steering while on a network). To do this:

- Press the button SW2 on the OM15082 Generic Expansion Board of the End Device.

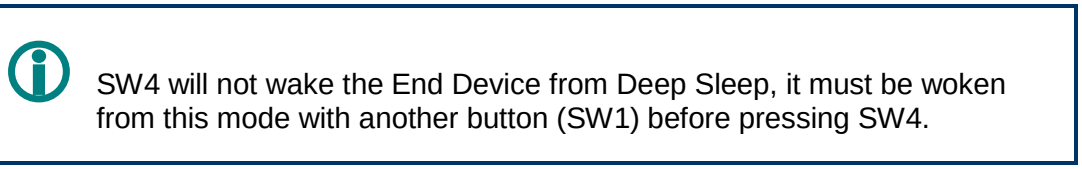
The End Device will then unicast to its parent a Management Permit Join Request. The parent will then re-broadcast this to the network and open the 'permit joining' window for 180 seconds. The Network Steering process (for devices not on a network) can now be triggered on the devices that are to join the network. The End Device is capable of opening the network to new joiners, but it is not capable of being a parent to these new joiners.

4.4.3 Binding Devices

The End Device supports 'Finding and Binding' as an Initiator that tries to find targets to bind to. For the purpose of the demonstration, the End Device supports the On/Off cluster as a client, so the Finding and Binding process will look for devices that support the On/Off cluster as a server in order to create bindings.

To trigger the Finding and Binding as an Initiator on the End Device, first trigger Finding and Binding on any target device and then do the following on the End Device:

- Press the button SW4 on the OM15082 Generic Expansion Board of the End Device



When Finding and Binding for a Target is completed and a binding is created, the End Device will send an Identify Off command to the target device to signal completion of the process for this Target. Depending on the type of bindings being created (either unicast or groupcast), an Add Group command may be sent to the Target.

Reporting is a mandatory feature in Zigbee 3.0, but it is not mandatory to request that reports are sent to a device. As a sleepy device, the End Device will most likely be asleep and not in a position to receive any reports, so this device does not create bindings on target devices for them to send reports.

4.4.4 Operating the Device

The operational functionality of this device in this demonstration is provided by the On/Off cluster. You can now send an OnOff Toggle command to the bound devices bound (in the Binding table) as follows:

- Press the button SW1 on the OM15082 Generic Expansion Board.

The effect that this command has on a bound device depends on the functionality related to the On/Off cluster on the device – for the Router in this demonstration, it will toggle a light (see Router Functionality).

4.4.5 Rejoining a Network

As an End Device, when this device is restarted in a state which is not factory-new, it will send a Network Rejoin Request to re-establish contact with its previous parent. If this fails, it will then try to join any Router on the network that will host it. The rejoin is attempted at power-on and when woken from deep sleep. All application, binding, group and network parameters are preserved in non-volatile memory.

4.4.6 Performing a Factory Reset

The End Device can be returned to its factory-new state (erasing all persistent data except the outgoing network frame counter) as follows:

- Hold down the USER INTERFACE button and press the RESET button on the Carrier Board.

The End Device will then unicast a Leave Indication to its parent, which will re-broadcast this message to the old network. The End Device will then delete all persistent data (other than the outgoing network frame counter) and perform a software reset.

There are two supported over-the-air commands for removing a device from the network - these are:

- Network Leave Request without rejoin
- ZDO Management Network Leave Request without rejoin

The Reset command of the Basic cluster will cause the ZCL to be reset to its factory-new defaults, resetting all attributes and configured reports. This will not remove the device from the network - all network parameters, groups and bindings will remain in place.

4.5 Installation Codes

The Zigbee Base Device allows for devices to join a network using unique install codes, rather than the well-known Default Link Key. This install code is only used for the initial join and is replaced by the Trust Centre immediately after the join with a new unique Link Key to secure future communication between the Trust Centre and the individual device. An installation code is 16 bytes in length.

To build the devices in this Application Note to use install codes for joining, edit the command line for each device to set the build option `ICODE=1`, then clean and rebuild each device.

Each joining device (Router or End Device) can have a random but not necessarily unique install code. How to program the install code into a device is described in more detail below. The install code should be provisioned to the Trust Centre via out-of-band means when the node is commissioned. It will be used to create a preconfigured Link Key.

To provision installation code in real devices, follow the instructions below:

1. Connect a USB port of your PC to the USB Mini B port marked “FTDI USB” on the Carrier Board (next to the 34-pin header) using a ‘USB A to Mini B’ cable. At this point, you may be prompted to install the driver for the USB virtual COM port.
2. Determine which serial communications port on your PC has been allocated to the USB connection.
3. On your PC, open a command window.
4. In the command window, navigate to the Flash Programmer directory:
   C:\NXP\JN518xProductionFlashProgrammer
5. Put the DK006 device into programming mode by holding down the ISP button while pressing and releasing the RESET button.
6. Write the installation code in the PSECT using a command with the following format at the command prompt:
   `JN518xProgrammer.exe –s <comport> -w install_code=<code>`
   where
   `<comport>` is the number of the serial communications port. E.g. COM3
   `<code>` is the 16 bytes installation code(alphabetic character are not case-sensitive). E.g. 01367ABD61045893AFF4A0E1C814B901
7. To commit the install code to psector memory the DK006 device should be reset either using the reset button or automatically by the flash programmer.

To read the installation code, enter a command with the following format at the command prompt, in Flash Programmer directory:
`JN518xProgrammer.exe –s <comport> -r install_code`
where
`<comport>` is the number of the serial communications port. E.g. COM3

Before a device can join to the network using the install code, the IEEE/MAC address and the install code need to be provided to the Trust Centre of the network.

To provide the install code and MAC address to the Coordinator, use the following command from a serial interface on a host terminal connected to the Coordinator hardware.

```
Code <MAC Address> <Install code>
```

where:

- `<MAC Address>` is the IEEE/MAC address of the device (MSB first, and alphabetic characters are not case-sensitive).
- `<Install code>` is the install code (MSB first, alphabetic characters are not case-sensitive, and the bytes may be separated by colons (':'), commas (',') or nothing).

For a device with the above IEEE/MAC address, the command would be:

```
code 00158D000035C9B8 01:36:7A:BD:61:04:58:93:AF:F4:A0:E1:C8:14:B9:01
```

To find the MAC address of the joining device, a command with the following format can be entered at the command prompt, in Flash Programmer directory:

```
JN518xProgrammer.exe –s <comport>
```

where
`<comport>` is the number of the serial communications port. E.g. COM3

After provisioning the install code and IEEE/MAC address in the Trust Centre, the normal procedure for joining a new device to the network can be followed. Details about forming a network and allowing other devices to the network can be found in Allowing Other Nodes to Join (Network Steering).

Once the install code has been used to join the new device, it is replaced with a new Trust Centre Link Key (the key derived from the install code is discarded and not stored for re-use). If a device is factory reset, it will not be able to re-associate with the network until the install code is re-provisioned in the Trust Centre.

5 Zigbee Over-The-Air (OTA) Upgrade

Over-The-Air (OTA) Upgrade is the method by which a new firmware image is transferred to a device that is already installed and running as part of a network. This functionality is provided by the OTA Upgrade cluster. In order to upgrade the devices in a network, two functional elements are required.

- **OTA Server:** First the network must host an OTA server, which will receive new OTA images from manufacturers, advertise the OTA image details to the network, and then deliver the new image to those devices that request it.
- **OTA Clients:** The second requirement is for OTA clients, which are located on the network devices that may need to be updated. These devices periodically interrogate the OTA server for details of the firmware images that it has available. If a client finds a suitable upgrade image on the server, it will start to request this image, storing each part as it is received. Once the full image has been received, it will be validated and the device will boot to run the new image.

New images are always pulled down by the clients, requesting each block in turn and filling in gaps. The server never pushes the images onto the network.

5.1 Overview

Support for the OTA Upgrade cluster as a client has been included for the Router and End Device devices. In order to build with these options, add `OTA=1` to the command line before building (this is disabled by default). This will add the relevant functionality to the device and invoke post-build processing to create a bootable image and two upgrade images. The produced binaries will be stored in the **OTABuild** directory. By default, unencrypted binaries will be produced. In order to build encrypted binaries, add the `OTA_ENCRYPTED=1` option to the command line before building (in addition to the `OTA=1` setting).

- The internal Flash memory is be used to store the upgrade image by default.

The initial client binaries to be programmed into the DK006 devices are V1 .bin files:

- **Router_NtagNwk_NtagOta_GpProxy_Ota_OM15081_V1.bin**
- **EndDevice_NtagNwk_NtagOta_Ota_RamOpt_OM15082_V1.bin**

The Application Note *Zigbee 3.0 IoT Control Bridge [JN-AN-1247]* has OTA server functionality built into it. The included ZGWUI tool provides an easy to use interface to serve OTA images to client devices. The OTA images are V2/V3 .ota files:

- **Router_NtagNwk_NtagOta_GpProxy_Ota_OM15081_V2.ota**
- **Router_NtagNwk_NtagOta_GpProxy_Ota_OM15081_V3.ota**
- **EndDevice_NtagNwk_NtagOta_Ota_RamOpt_OM15082_V2.ota**
- **EndDevice_NtagNwk_NtagOta_Ota_RamOpt_OM15082_V3.ota**

5.2 OTA Upgrade Operation

Follow the instructions in *Zigbee 3.0 IoT Control Bridge [JN-AN-1247]* to serve a .ota upgrade to a client device.

Any devices with OTA clients in the network will periodically send Match Descriptor Requests in order to find an OTA server. Once a server responds, it will then be sent an IEEE Address Request in order to confirm its address details. After this, the clients will periodically send OTA Image Requests to determine whether the server is hosting an image for that client device. In response to the Image Request, the server will return details of the image that it is currently hosting - Manufacturer Code, Image Tag and Version Number. The client will check these credentials and decide whether it requires this image. If it does not, it will query the server again at the next query interval. If the client does require the image, it will start to issue Block Requests to the server to get the new image. Once all blocks of the new image have been requested and received, the new image will be verified, the old one invalidated, and the device will reboot and run the new image. The client will resume periodically querying the server for new images.

The End Device is allowed to enter deep sleep mode when there has been no significant activity for a period of time, as this could result in long periods without OTA progressing if in deep sleep mode for extended periods of time. The application code therefore does not allow the End Device to enter deep sleep mode while OTA discovery or download is in progress. The End Device is allowed to enter sleep mode with RAM held and wake timers running and will do this between image block requests (except between the start of the download and receiving the OTA image string when it is kept awake). Following a reset or wake from deep sleep the End Device will work though the discovery process fairly quickly in order to allow deep sleep mode to be entered as soon as possible to extend battery life.

5.3 Image Credentials

There are four main elements of the OTA header that are used to identify the image, so that the OTA client is able to decide whether it should download the image. These are Manufacturer Code, Image Type, File Version and OTA Header String:

- **Manufacturer Code:** This is a 16-bit number that is a Zigbee-assigned identifier for each member company. In this application, this number has been set to 0x1037, which is the identifier for NXP. In the final product, this should be changed to the identifier of the manufacturer. The OTA client will compare the Manufacturer Code in the advertised image with its own and the image will be downloaded only if they match.
- **Image Type:** This is a manufacturer-specific 16-bit number in the range 0x0000 to 0xFFBF. Its use is for the manufacturer to distinguish between devices. In this application, the Image Type is normally set to the Zigbee Device Type however this application uses 0x0001 for the Router device and 0x0002 for the End Device (0x8001 and 0x8002 are used for encrypted images). The OTA client will compare the advertised Image Type with its own and only download the image if they match. The product designer is entirely free to implement an identification scheme of their own.
- **File Version:** This is a 32-bit number representing the version of the image. The OTA client will compare the advertised version with its current version before deciding whether to download the image.
- **OTA Header String:** This is a 32-byte character string and its use is manufacturer-specific. In this application, the OTA client will compare the string in the advertised image with its own string before accepting an image for download. If the strings match then the image will be accepted. In this way, the string can be used to provide extra detail for identifying images, such as hardware sub-types.

5.4 Encrypted and Unencrypted Images

OTA images can be provided to the OTA server in either encrypted or unencrypted form. Encrypting the image will protect sensitive data in the image while it is being transferred from the manufacturer to the OTA server. Regardless of whether the image itself is encrypted, the actual transfer over-air will always be encrypted in the same way as any other on-air message.

5.5 Upgrade and Downgrade

The decision to accept an image following a query response is under the control of the application. The code, as supplied, will accept an upgrade or a downgrade. As long as the notified image has the right credentials and a version number which is different from the current version number, the image will be downloaded. For example, if a client is running a v3 image and a server is loaded with a v2 image then the v2 image will be downloaded. If it is required that the client should only accept upgrade images (v2 -> v3 -> v5), or only accept sequential upgrade images (v2 -> v3 -> v4 -> v5) then the application callback function that handles the Image

6 NFC Over-The-Air (OTA) Upgrade

When running the **Coordinator_NciIcode_NciOta_GpProxy_OM5578.bin** binary on a Carrier Board fitted with the NFC Controller board the firmware in the Router and End Devices can be upgraded via NFC.

> This is an experimental feature that is currently being evaluated and there are limitations on its use:
>
> Encrypted OTA files are not supported.
>
> The OTA image to be transferred must be stored in the Coordinator's flash memory.
>
> The device being updated remains fully awake during the OTA process.

This method allows new firmware images to be transferred from the Coordinator to a Router or End Device using NFC. The same files used for Zigbee OTA are also used for NFC OTA.

The initial client binaries to be programmed into the DK006 devices are V1 .bin files:

- **Router_NtagNwk_NtagOta_GpProxy_Ota_OM15081_V1.bin**
- **EndDevice_NtagNwk_NtagOta_Ota_RamOpt_OM15082_V1.bin**

The OTA images to be stored in the Coordinator's flash are V2/V3 .ota files:

- **Router_NtagNwk_NtagOta_GpProxy_Ota_OM15081_V2.ota**
- **Router_NtagNwk_NtagOta_GpProxy_Ota_OM15081_V3.ota**
- **EndDevice_NtagNwk_NtagOta_Ota_RamOpt_OM15082_V2.ota**
- **EndDevice_NtagNwk_NtagOta_Ota_RamOpt_OM15082_V3.ota**

The OTA (**.ota**) files to be served to the Router or End Device should be programmed into the Coordinator's flash memory using the batch file (**.bat**) provided alongside the OTA (**.ota**) file. (This is programmed to a different location to the firmware the Coordinator executes and also different to the location of the PDM non-volatile memory.)

Take care to program the .ota file into the Coordinator device and not the device to be updated.

With the Coordinator running in a network it can be placed into NFC OTA mode by entering "nfc ota" on the serial interface. When in this mode the green and yellow LEDs on the OM5578 NFC Controller board will flash together slowly.

To perform OTA the Router or End Device should be placed into the Coordinator's NFC field. During OTA the green and yellow LEDs will flash together quickly. After a successful OTA the green and yellow LEDs will remain lit until the Router or End Device is removed from the field when they return to flashing slowly. A failure to update is indicated by the LEDs being extinguished until the Router or End Device is removed from the field.

When the Router or End Device is removed from the field the new firmware is validated and activated with the device running the new firmware.

It takes approximately 3 minutes to update the Router firmware using this method. It may be useful to have a support to rest the Router or End Device upon next to the Coordinator during this process (ensuring no physical contact between the board to avoid shorting). If the process is interrupted due to early removal it will resume when the devices are placed close together again.

7 Zigbee Green Power (GP) Support

This section describes the provision for the addition of Zigbee Green Power (GP) devices to the network. To support GP devices, the Zigbee devices in the network must also act as GP 'infrastructure devices' to facilitate the reception, routing and handling of GP frames from GP devices. The following GP infrastructure devices are available in this Application Note:

- **GP Proxy Basic** support is provided for the Coordinator and Router device types. Devices with GP Proxy Basic support act as GP proxy nodes that can forward GP frames to Combo Basic devices and help to extend the range of the network.

Zigbee Green Power is described in the *Zigbee Green Power User Guide [JN-UG-3134]*.

The applications in this Application Note are supplied pre-built with GP support as follows:

- Coordinator with GP Proxy Basic functionality
- Router with GP Proxy Basic functionality

The GP binary files are indicated by **GpProxy** in the folder/file names.

7.1 Green Power Configuration Settings

The security keys to be used by the Proxy Basic devices must be configured in the **zcl_options.h** file on the devices, as described below.

To set the security key type to be used, add the following line to the above file:

```
#define GP_KEYTPE <key_type>
```

where `<key_type>` is any one of the following enumerated values:

```
typedef enum
{
  E_GP_NO_KEY = 0x00,
  E_GP_ZIGBEE_NWK_KEY,
  E_GP_ZGPD_GROUP_KEY,
  E_GP_NWK_KEY_DERIVED_ZGPD_GROUP_KEY,
  E_GP_OUT_OF_THE_BOX_ZGPD_KEY,
  E_GP_DERIVED_INDIVIDUAL_ZGPD_KEY = 0x07
}teGP_GreenPowerSecKeyType;
```

The key will be derived from the Zigbee network key or from a key sent by the GP device, except for a group key. When the key type is set to E_GP_ZGPD_GROUP_KEY, a shared group key must be specified using the GP_SHARED_KEY macro, as follows:

```
#define GP_SHARED_KEY  <key>
```

where `<key>` is the value of the key.

8 Developing with the Application Note

The example applications provided in this Application Note were developed using the:

- MCUXpresso IDE
- JN518x Zigbee 3.0 SDK
- K32W061/K32W041 Zigbee 3.0/Bluetooth SDK

These are the resources that you should use to develop Zigbee 3.0 applications for DK006 microcontrollers. They are available free-of-charge to authorised users via the MCUXpresso web site.

For instructions on how to install the MCUXpresso IDE, DK006 SDK and Application Notes including how to rebuild the applications see the *Zigbee 3.0 Getting Started Application Note [JN-AN-1260]*.

Throughout your Zigbee 3.0 application development, you should refer to the documentation listed in Related Documents.

8.1 Compilation for Specific Chips/SDKs

The Application Notes are provided ready for compilation for a single chip on a single SDK, the default configuration is specified in the Release Notes for each Application Note. To alter the compilation for a different chip/SDK use comments near the top of the makefile to select the appropriate chip using the JENNIC_CHIP variable which will also select the appropriate SDK. This assumes that MCUXpresso and the SDK has been installed following the instructions in the *Zigbee 3.0 Getting Started Application Note [JN-AN-1260]*. The example below selects the K32W061 chip and the appropriate SDK:

```
# Set specific chip   (choose one)
JENNIC_CHIP            ?= K32W061
#JENNIC_CHIP           ?= K32W041
#JENNIC_CHIP           ?= JN5189
#JENNIC_CHIP           ?= JN5188
```

8.2 Common Code

This section lists and describes the source files that provide functionality common to all the devices in this Application Note and are held in the **Common/source** directory.

app_zpscfg is a configuration file for the Zigbee stack. For each of the devices in the application, it defines all the required stack parameters, table sizes, servers etc. This file is processed as part of the build process and creates device-specific source files to be built into each device.

app_buttons.c provides an interface to read the switches/buttons on the expansion boards and to post button-press events to the application event queue.

app_events.h contains type definitions of the application events.

app_ntag.c contains the code that drives the NFC commissioning data exchange and initiates the joining process when valid data is read from the NTAG. This code uses the newer NTAG data format that employs Zigbee Installation Code encryption. This is the code used by devices fitted with an NTAG that is used to join a network (Routers and End Devices).

app_nci_icode.c contains the code that drives the NFC commissioning data exchange using the NFC Controller board to read and write data when NTAGs are presented. This is the code used by devices fitted with an NCI that is used to allow other devices to join a network (Coordinators).

app_ota_client.c contains the application code that drives the OTA process.

app_pdm.c provides error event callbacks for the Persistent Data Manager (PDM), in order to notify the application of the state of the PDM.

PDM_IDs.h provides unique identifiers for all the data records in the PDM.

8.3 NFC Folder

The NFC libraries and header files containing the public APIs for NFC are held in the **NFC** directory. This code uses the newer NTAG data format that employs Zigbee Installation Code encryption and is used in the default builds.

Documentation for these APIs and the **app_ntag.c/h** and **app_nci_icode.c/h** application layer APIs can be found in the **NFC.chm** help file in the **Doc** directory of this Application Note.

8.4 Coordinator Application Code

This section lists and describes the source files for the Coordinator application code, which are provided in the **Source** directory for the application. You may wish to use this code as a basis for your own application development. You can rebuild your customised application as described in Rebuilding the Applications.

app_main.c hosts the main program loop, and defines and initialises system resources, queues, timers etc.

app_start.c manages the chip start-up, calls the initialisation functions and launches the main program loop.

app_coordinator.c hosts the event handlers for the application and the Base Device callback. The Base Device callback receives Base Device Events and AF Stack events after the Base Device has completed any processing that is required. These events can then be further processed by the application. The events include data indications that are passed to the ZCL for processing and network management events (such a 'network formed' and 'new node has joined') in order to keep the application informed of the network state. The application event queue is processed to receive button-press events.

app_zcl_task.c hosts the device-specific ZCL initialisation and callback functions. The ZCL callbacks notify the application of the results of received ZCL commands and responses, so that the application can take the appropriate action. The ZCL tick timer is used to provide ticks for the ZCL to manage timer-dependent events or state transitions.

app_serial_commands.c provides the command interpreter of the serial interface - where appropriate, the serial commands create application button-press events to trigger the required actions.

uart.c receives and handles characters transmitted on the serial interface.

bdb_options.h defines parameters used by the Base Device, such as primary and secondary channel masks.

zcl_options.h defines the features of the ZCL, such as which clusters are supported, whether a client and or a server, and which optional commands and attributes are supported. The mandatory commands and attributes of the selected cluster will be automatically included.

8.4.1 Command Line Build Options

The following command line options can be used to configure the built devices:

- `GROUPS=0` to build so that bound commands use unicast transmission
- `GROUPS=1` to build so that bound commands use groupcast transmission
- `APP_NCI_ICODE=1` to build with NCI/NFC joining functionality. **NciIcode** is included in output filenames when using this setting.
- `APP_NCI_OTA=1` to build with NCI/NFC OTA experimental functionality. **NciOta** is included in output filenames when using this setting.
- `GP_SUPPORT=1, GP_DEVICE=PROXY_BASIC` to build with Green Power Proxy Basic functionality. **GpProxy** is included in output filenames when using this setting.

8.5 Router Application Code

This section lists and describes the source files for the Router application code, which are provided in the **Source** directory for the application. You may wish to use this code as a basis for your own application development. You can rebuild your customised application as described in Rebuilding the Applications.

app_main.c hosts the main program loop, and defines and initialises system resources, queues, timers etc.

app_start.c manages the chip start-up, calls the initialisation functions and launches the main program loop.

app_router_node.c hosts the event handlers for the application and the Base Device callback. The Base Device callback receives Base Device Events and AF Stack events after the Base Device has completed any processing that is required. These events can then be further processed by the application. The events include data indications that are passed to the ZCL for processing and network management events (such a 'network formed' and 'new node has joined') in order to keep the application informed of the network state. The application event queue is processed to receive button-press events.

app_zcl_task.c hosts the device-specific ZCL initialisation and callback functions. The ZCL callbacks notify the application of the results of received ZCL commands and responses, so that the application can take the appropriate action. The ZCL tick timer is used to provide ticks for the ZCL to manage timer-dependent events or state transitions.

app_reporting.c provides the support for the reporting functionality of the device. It manages the restoring of the reporting configuration and the saving of any changes, when a Configure Reports command is received.

bdb_options.h defines parameters used by the Base Device, such as primary and secondary channel masks.

zcl_options.h defines the features of the ZCL, such as which clusters are supported, whether a client and or a server, and which optional commands and attributes are supported. The mandatory commands and attributes of the selected cluster will be automatically included.

8.5.1 Command Line Build Options

The following command line options can be used to configure the built devices:

- `ICODES=0` to build so that install codes are not used
- `ICODES=1` to build so that install codes are used
- `APP_NTAG_NWK=1` to build with NTAG/NFC joining support. **NtagNwk** is included in output filenames when using this setting.
- `APP_NTAG_OTA=1` to build with NTAG/NFC OTA experimental functionality. **NtagOta** is included in output filenames when using this setting.
- `GP_SUPPORT=1, GP_DEVICE=PROXY_BASIC` to build with Green Power Proxy Basic functionality. **GpProxy** is included in output filenames when using this setting.
- `OTA=1` to build with OTA client. **Ota** is included in output filenames when using this setting.
- `OTA_ENCRYPTED=1` to build OTA images with encryption. **OtaEnc** is included in output filenames when using this setting.

8.6 End Device Application Code

This section lists and describes the source files for the End Device application code, which are provided in the **Source** directory for the application. You may wish to use this code as a basis for your own application development. You can rebuild your customised application as described in Rebuilding the Applications.

app_main.c hosts the main program loop, and defines and initialises system resources, queues, timers etc.

app_start.c manages the chip start-up, calls the initialisation functions and launches the main program loop.

app_end_device_node.c hosts the event handlers for the application and the Base Device callback. The Base Device callback receives Base Device Events and AF Stack events after the Base Device has completed any processing that is required. These events can then be further processed by the application. The events include data indications that are passed to the ZCL for processing and network management events (such a 'network formed' and 'new node has joined') in order to keep the application informed of the network state. The application event queue is processed to receive button-press events.

app_zcl_task.c hosts the device-specific ZCL initialisation and callback functions. The ZCL callbacks notify the application of the results of received ZCL commands and responses, so the application can take the appropriate action. The ZCL tick timer is used to provide ticks for the ZCL to manage timer-dependent events or state transitions.

bdb_options.h defines parameters used by the Base Device, such as primary and secondary channel masks.

zcl_options.h defines the features of the ZCL, such as which clusters are supported, whether a client and or a server, and which optional commands and attributes are supported. The mandatory commands and attributes of the selected cluster will be automatically included.

8.6.1 RAM Retention Options During Sleep

The JN5189/K32W061 RAM is stored in 8 banks: 4 of 16kb, 2 of 8kb, 2 of 4kb. During sleep each bank can be powered (retaining its data) or powered down (discarding its data). The amount of energy consumed while sleeping is directly affected by the amount of data being retained. The Power Manager (PWRM) is responsible for controlling the RAM banks that are powered, any bank with no data being used is always powered down during sleep. End Device applications can be built in two different ways that affect how the RAM containing data is powered during sleep.

Output in the MCUXpresso console panel highlights the amount of RAM held during sleep and the banks in which it resides.

8.6.1.1 Full RAM Retention

This mode keeps all the RAM banks that contain data powered during sleep. This is the easiest mode to work with as all data is retained during sleep at the expense of retaining data that might not be necessary.

The following example output shows 16kb of RAM being retained in banks 5 to 7. 13992 bytes of the 16kb are used by the firmware, leaving 2392 bytes spare until another RAM bank is required:

```
retain spare kb    banks filename
 13992  2392 16 -----567 EndDevice_OM15082.map
```

This example is taken from the End Device build with minimal features.

8.6.1.2 Partial RAM Retention

This mode allows data variables that can be discarded during sleep to be placed into separate “discard” linker sections. RAM data not placed into a discard linker section is retained as above. The Zigbee stack already has data that can be discarded in such sections and some application data in the End Device template is also allowed to be discarded. Whilst this mode reduces the energy used to retain RAM during sleep the wake-up process will be longer as the discarded data is re-initialised to default values and some stack data is transferred from PDM.

The following example output shows 4kb of RAM being retained in bank 7 only. 2969 bytes of the 4kb are used by the firmware, leaving 1400 bytes spare until another RAM bank is required:

```
retain spare kb    banks filename
  2696  1400  4 -------7 EndDevice_NtagNwk_NtagOta_Ota_RamOpt_OM15082.map
```

This example is taken from the End Device build with full features.

To place an application variable into the discard linker section it should be declared with a section attribute as follows:

```
PRIVATE uint8 au8NtagData[NTAG_DATA_SIZE]
                                    __attribute__((section(".bss.discard.app")));
```

Setting RAMOPT=1 on the make command-line (or in the makefile) uses the appropriate linker file for partial RAM retention (the discard section attributes will have no effect which building with the full RAM retention setting RAMOPT=0).

8.6.2 Command Line Build Options

The following command line options can be used to configure the built devices:

- `ICODES=0` to build so that install codes are not used
- `ICODES=1` to build so that install codes are used
- `GROUPS=0` to build so that bound commands use unicast transmission
- `GROUPS=1` to build so that bound commands use groupcast transmission
- `APP_NTAG_NWK=1` to build with NTAG/NFC joining support. **NtagNwk** is included in output filenames when using this setting.
- `APP_NTAG_OTA=1` to build with NTAG/NFC OTA experimental functionality. **NtagOta** is included in output filenames when using this setting.
- `OTA=1` to build with OTA client. **Ota** is included in output filenames when using this setting.
- `OTA_ENCRYPTED=1` to build OTA images with encryption. **OtaEnc** is included in output filenames when using this setting.
- `RAMOPT=1` to build with reduced RAM held through sleep with RAM held. This reduces the energy used whilst sleeping by retaining less RAM, however wake will take longer as data will need to be re-initialised. **RamOpt** is included in output filenames with this setting.

9 Related Documents

The following manuals will be useful in developing custom applications based on this Application Note:

- Zigbee 3.0 Getting Started Application Note [JN-AN-1260]
- DK6 Production Flash Programmer User Guide [JN-UG-3127]
- Zigbee 3.0 Stack User Guide [JN-UG-3130]
- Zigbee 3.0 Devices User Guide [JN-UG-3131]
- Zigbee 3.0 Cluster Library User Guide [JN-UG-3132]
- Zigbee 3.0 Green Power User Guide [JN-UG-3134]
- Encryption Tool User Guide [JN-UG-3135]

Important Notice

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.

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer’s applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EMBRACE, GREENCHIP, HITAG, ICODE, JCOP, LIFE VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, AltiVec, CodeWarrior, ColdFire, ColdFire+, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, Tower, TurboLink, EdgeScale, EdgeLock, eIQ, and Immersive3D are trademarks of NXP B.V.

All other product or service names are the property of their respective owners. AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, µVision, Versatile are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

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

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

Date of release: 06-Sep-2021

Document identifier: JN-AN-1243