



Application Note: JN-AN-1217

ZigBee 3.0 Base Device

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 JN516x or JN517x wireless microcontrollers. An example application can be employed as:

- A demonstration using the supplied pre-built binaries that can be run on nodes of the JN516x/7x Evaluation 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 JN516x and JN517x platforms.

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-3114]*.

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-3113]* 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 the chip family which you are using - either JN516x or JN517x:

- **JN516x:** If developing for the JN516x microprocessors, you will need:
 - 'BeyondStudio for NXP' IDE [JN-SW-4141]
 - JN516x ZigBee 3.0 SDK [JN-SW-4170]

For installation instructions, refer to the *BeyondStudio for NXP Installation and User Guide (JN-UG-3098)*.

- **JN517x:** If developing for the JN517x microprocessors, you will need:
 - LPCXpresso IDE
 - JN517x ZigBee 3.0 SDK [JN-SW-4270]

For installation instructions, refer to the *JN517x LPCXpresso Installation and User Guide (JN-UG-3109)*.

The LPCXpresso software can be obtained as described in the *JN517x ZigBee 3.0 SDK Release Notes*, which indicate the version that you will need.

All other resources are available via the [ZigBee 3.0](#) page of the NXP web site.



Note: The code in this Application Note can be used in either BeyondStudio or LPCXpresso and the process for importing the application into the development workspace is the same for both.



Note: Prebuilt JN5169 and JN5179 application binaries are supplied in this Application Note package, but the applications can be rebuilt for other devices in the JN516x and JN517x families (see Section 5.7).

The software and documentation resources referenced in this Application Note are available free-of-charge via the [ZigBee 3.0](#) page of the NXP web site.

2.2 Hardware

Hardware kits are available from NXP to support the development of ZigBee 3.0 applications. The following kits respectively provide JN516x-based and JN517x-based platforms for running these applications:

- JN516x-EK004 Evaluation Kit, which features JN5169 devices
- JN517x-DK005 Development Kit, which features JN5179 devices

Both of these kits support the NFC commissioning of network nodes (see Section 3.1).

It is also possible to develop ZigBee 3.0 applications to run on the components of the earlier JN516x-EK001 Evaluation Kit, which features JN5168 devices, but this kit does not support NFC commissioning.

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 (sleepy)

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 JN516x/7x Evaluation Kits.

- To load the pre-built binaries into the evaluation kit components and run the demonstration application, refer to Section 4.
- To start developing you own applications based on the supplied source files, refer to Section 5.

3.1 NFC Hardware Support

Some NXP hardware kits for the development of ZigBee 3.0 applications provide the possibility of network commissioning through Near Field Communication (NFC). The kits and components that provide NFC support are indicated in the table below.

Hardware Kit	Hardware Components for NFC	Field Detect Connection
JN517x-DK005	NFC is built into the OM15028 Carrier Board	GPIO 17
JN516x-EK004	DR1174 Carrier Board plus OM15044 and either OM55679/NT3120 or OM5569/NT322E <i>Note: A 4K7 resistor should be fitted to the R1 pads on the OM15044 board to avoid unnecessary reads of the NTAG due to the FD line floating.</i>	DIO 0

Table 2: NFC Support in JN516x/7x Hardware Kits

The Field Detect of the NFC chip needs to be connected to an IO line of the JN516x/7x module so that an interrupt can be generated as the device is moved in or out of the field. This is achieved by fitting a jumper to the pin specified in the above table.



Note: Early samples of the JN516x-EK004 kit used a yellow wire rather than a jumper for the Field Detect connection, but the pin is the same.

3.2 NFC Data Formats

Two different NFC data formats are supported for commissioning. The Router and End Device applications can be built to support only one (or none) of these:

- **ZigBee Installation Code Format:** This is a newer format introduced with v1003 of this Application Note. The applications are built to use this format by default. This format uses a key derived from the device's ZigBee Installation Code to encrypt data in the NTAG.
- **AES Encryption Format:** This older format uses an AES key to encrypt data in the NTAG.

The selection of the data format can be made at compile-time by using makefile variables described in the [Router Command Line Build Options](#) or [End Device Command Line Build Options](#).



Note: The Application Note JN-AN-1222, IoT Gateway Host With NFC, versions v2007 and later is able to commission either of these formats depending upon the data in the presented NTAG. Earlier versions support only AES Encryption Format.

4 Running the Demonstration Application

This section describes how to use the supplied pre-built binaries to run the example applications on components of the JN516x-EK004 or JN517x-DK005 kit.

4.1 Loading the Applications

The table below lists the application binary files supplied with this Application Note and indicates the JN516x-EK004 or JN517x-DK005 kit components with which the binaries can be used. These files are located in the **Build** directories for the relevant applications.

Application	JN5169 Binary File	JN516x-EK004 Hardware
Coordinator	Coordinator_JN5169_DR1199.bin	DR1174 Carrier Board with JN5169 module DR1199 Generic Expansion Board
Coordinator	Coordinator_JN5169_DONGLE.bin	OM15020 JN5169 USB Dongle
Router	Router_Ntaglcode_JN5169_DR1175.bin	DR1174 Carrier Board with JN5169 module DR1175 Lighting/Sensor Expansion Board OM15044 NTAG Adaptor Board OM5569/NT322E NTAG Board
End Device	Enddevice_Ntaglcode_JN5169_DR1199.bin	DR1174 Carrier Board with JN5169 module DR1199 Generic Expansion Board OM15044 NTAG Adaptor Board OM5569/NT322E NTAG Board
Application	JN5179 Binary File	JN517x-DK005 Hardware
Coordinator	Coordinator_JN5179_DR1199.bin	OM15028 Carrier Board with JN5179 module DR1199 Generic Expansion Board
Coordinator	Coordinator_JN5179_DONGLE.bin	OM15020 JN5179 USB Dongle
Router	Router_Ntaglcode_JN5179_DR1175.bin	OM15028 Carrier Board with JN5179 module DR1175 Lighting/Sensor Expansion Board
End Device	Enddevice_Ntaglcode_JN5179_DR1199.bin	OM15028 Carrier Board with JN5179 module DR1199 Generic Expansion Board

Table 3: Application Binaries and Hardware Components

A binary file can be loaded into the Flash memory of a JN516x/7x device using the JN51xx Flash Programmer [JN-SW-4107], available via the NXP web site. This software tool is described in the *JN51xx Production Flash Programmer User Guide* [JN-UG-3099].



Note: You can alternatively load a binary file into a JN516x/7x device using the Flash programmer built into the relevant IDE (see Section 5).

To load an application binary file into a JN516x/7x 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 on the Carrier Board using a 'USB A to Mini B' cable. At this point, you may be prompted to install the driver for the cable.
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\ProductionFlashProgrammer

5. Run the Flash programmer to download your binary file to JN516x/7x Flash memory by entering a command with the following format at the command prompt:

```
JN51xxProgrammer.exe -s <comport> -f <path to .bin file>
```

where <comport> is the number of the serial communications port.

6. Once the download has successfully completed, disconnect the USB cable and, if required, reset the board or module to run the application.

Operating instructions for the different applications are provided in the sections below.

4.2 Using the Coordinator

This section describes how to commission and operate the Coordinator application in a ZigBee 3.0 network. To use this application, you must have programmed the application binary into the relevant physical device – either of the following:

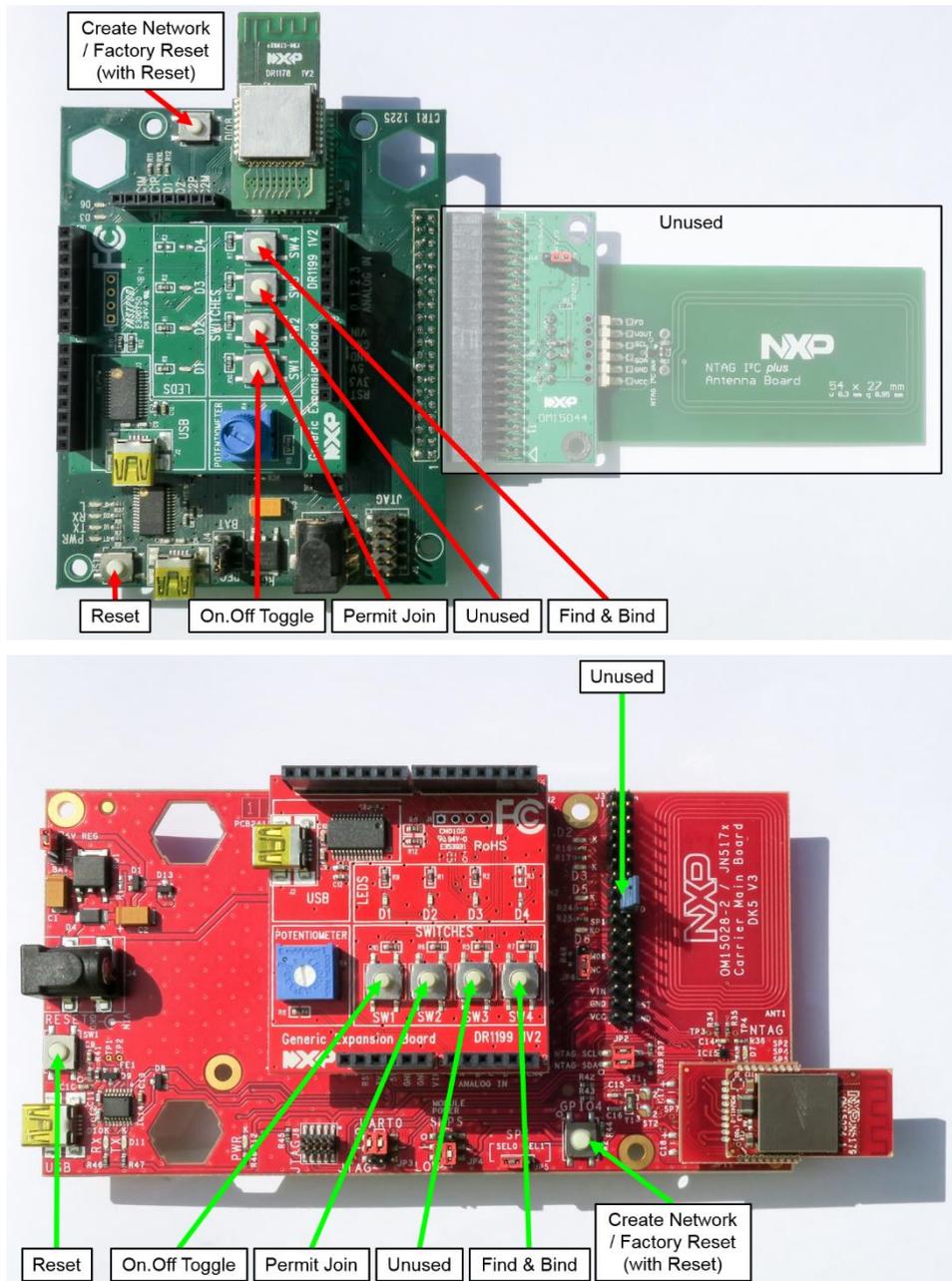
- **Coordinator_JN51xx_DR1199.bin** into the JN516x/7x module on a Carrier Board fitted with the DR1199 Generic Expansion Board
- **Coordinator_JN51xx_DONGLE.bin** into a JN5169/79 USB Dongle

Programming instructions are provided in Section 4.1.

To incorporate the device into a ZigBee 3.0 network and then operate the device, work through Sections 4.2.2 to 4.2.5.

4.2.1 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 Behavior 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 JN5169/JN5179 USB Dongle, this is the only interface available. The serial interface is not case-sensitive. For a summary of the serial interface, refer to Section 4.2.8.
- Using the push-buttons on the DR1199 Generic Expansion Board – this is obviously only available when a Coordinator application is programmed into a JN516x/7x module on a Carrier Board.

4.2.2 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 DIO8/GPIO4 on the DR1174/OM15028 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 JN5169 USB Dongle), the periodic link status messages can then be observed on the operational channel.

4.2.3 Allowing Other Nodes to Join

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 DR1199 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.

4.2.4 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 and then do either of the following on the Coordinator (Initiator):

- Press the button SW4 on the DR1199 Generic Expansion Board.
- Enter “find” on the serial interface (Dongle or DR1199).

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 DR1199 Generic Expansion Board.
- Enter “toggle” in the serial interface (Dongle or DR1199).

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 Section 4.3).

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 DIO8/GPIO4 button and press the RST button on the Carrier Board
- Enter “factory reset” on the serial interface (Dongle or DR1199).

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+DIO8	Factory Reset	Factory resets the device, erasing persistent data
Reset	Soft Reset	Triggers a software reset (no loss of data)
-	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.

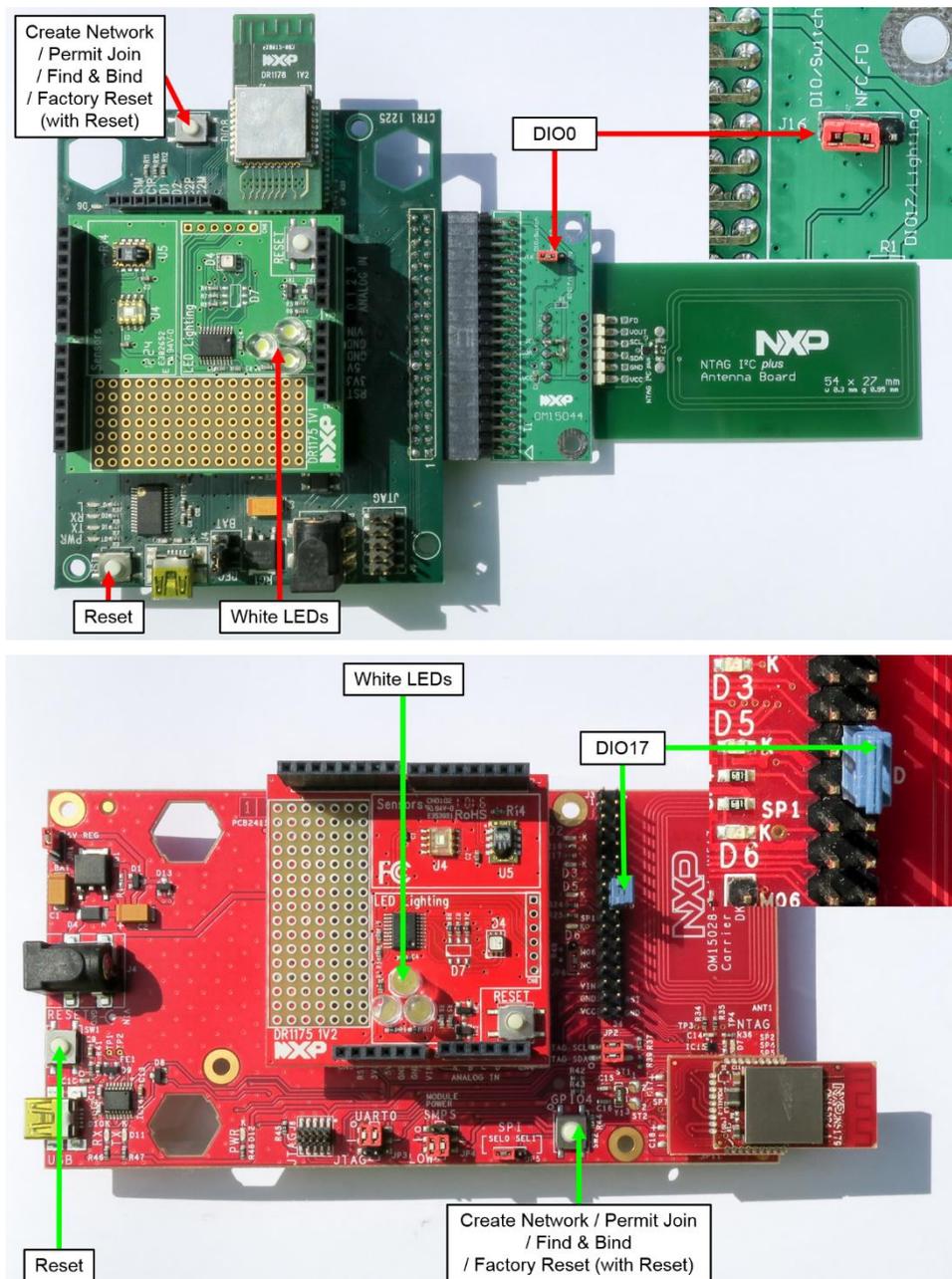
4.3 Using the Router

This section describes how to commission and operate the Router application in a ZigBee 3.0 network. To use this application, you must have programmed the application **Router_Ntagcode_JN51xx_DR1175.bin** into the JN516x/7x module on a Carrier Board fitted with the DR1175 Lighting/Sensor Expansion Board, as described in Section 4.1

To incorporate the device into a ZigBee 3.0 network and then operate the device, work through Sections 4.3.2 to 4.3.5.

4.3.1 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 Behavior Specification.

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

4.3.2 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.

4.3.2.1 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 RST or 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 follow, as described in Section 4.3.2.3.

4.3.2.2 Joining an Existing Network using NFC

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

Ensure the hardware is set up for NFC as described in Section 3.1.

Instructions for this process are included in the above Application Note (JN-AN-1222).

4.3.2.3 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 DIO8/GPIO4 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 Section 4.3.3.

4.3.3 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 DIO8/GPIO4 button on the Carrier Board (the same button is also used to start Finding and Binding, described in Section 4.3.4).

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.

4.3.4 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 DIO8/GPIO4 button on the Carrier Boards of all the target devices (the same button also is also used to start Network Steering, described in Section 4.3.3).
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.

4.3.5 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 DR1175 Lighting/Sensor Expansion Board.

4.3.6 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.

4.3.7 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 DIO8/GPIO4 button and press the RST 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.

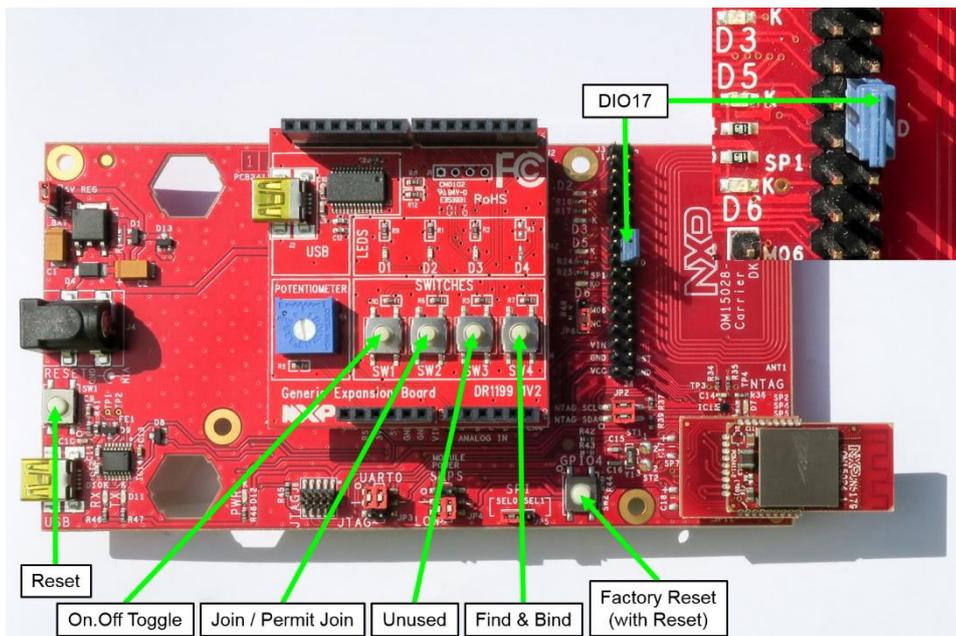
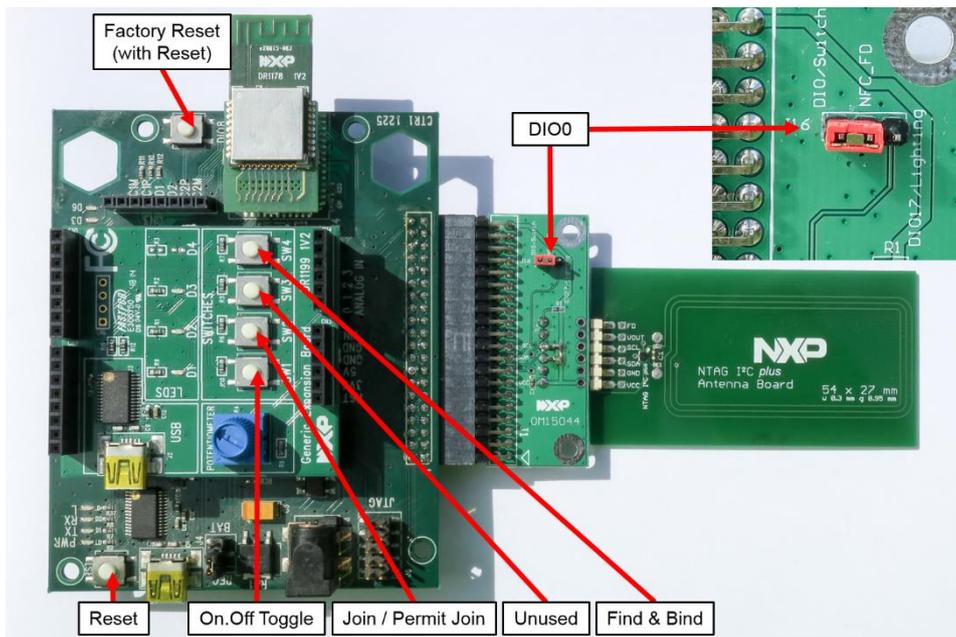
4.4 Using the End Device

This section describes how to commission and operate the End Device application in a ZigBee 3.0 network. To use this application, you must have programmed the application **EndDevice_Ntagcode_JN51xx_DR1199.bin** into the JN516x/7x module on a Carrier Board fitted with the DR1199 Generic Expansion Board, as described in Section 4.1.

To incorporate the device into a ZigBee 3.0 network and then operate the device, work through Sections 4.4.2 to 4.4.5.

4.4.1 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 Behavior 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.

4.4.2 Joining a Network

4.4.2.1 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 DR1199 Generic Expansion Board of the End Device.

4.4.2.2 Joining an Existing Network using NFC

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

Ensure the hardware is set up for NFC as described in Section 3.1.

Instructions for this process are included in the above Application Note (JN-AN-1222).

4.4.3 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 DR1199 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.

4.4.4 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 DR1199 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.

4.4.5 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 bound (in the Binding table) as follows:

- Press the button SW1 on the DR1199 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 Section 4.3).

4.4.6 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.

4.4.7 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 DIO8/GPIO4 button and press the RST 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.

Each joining device (Router or End Device) must have a unique install code. How this code is generated and provisioned in real devices is beyond the scope of this Application Note. For the purpose of the demonstration, each joining device will create an install code that is its 8-byte IEEE/MAC address repeated once. For example, a device with an IEEE/MAC address of 00158D000035C9B8 will generate an install code of:

```
00:15:8D:00:00:35:C9:B8: 00:15:8D:00:00:35:C9:B8
```

Before a device using install codes can be joined to the network, the IEEE/MAC address and install code of this device need to be added to the Trust Centre of the network. The serial interface provides a command to do this. This command has the format:

```
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 00,15,8D,00,00,35,C9,B8, 00,15,8D,00,00,35,C9,B8
```

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.

Once the install code has been used to join the new device, it is replaced with a new Trust Centre Link Key (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.

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.

5 Developing with the Application Note

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

- JN516x ZigBee 3.0 SDK [JN-SW-4170] and the 'BeyondStudio for NXP' IDE [JN-SW-4141]
- JN517x ZigBee 3.0 SDK [JN-SW-4270] and the LPCXpresso IDE

These are the resources that you should use to develop JN516x and JN517x ZigBee 3.0 applications, respectively. They are available free-of-charge via the [ZigBee 3.0](#) page of the NXP web site.

Throughout your ZigBee 3.0 application development, you should refer to the documentation listed in Section 6.

5.1 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.zpscfcg 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_aes.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 older NTAG data format that employs AES encryption and is not used in the default builds.

app_ntag_icode.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 and is used in the default builds.

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.

5.2 NTAG Folder (AES Format)

The NTAG library and header files containing the public APIs for NFC are held in the **NTAG** directory. This code uses the older NTAG data format that employs AES encryption and is not used in the default builds.

5.3 NFC Folder (ZigBee Installation Code Format)

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_icode.c/h** APIs can be found in the **NFC.chm** help file in the **Doc** directory of this Application Note.

5.4 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 Section 5.7.

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

app_start.c manages the JN516x/7x 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.

irq_JN516x.s defines which of the JN516x hardware interrupts are supported and serviced, and at which priority. This is defined by two tables - an interrupt priority table and a table of handler functions. This file is not used for JN517x.

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.

5.4.1 Command Line Build Options

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

- `JENNIC_CHIP_FAMILY=JN516x` to build for a JN516x microcontroller
- `JENNIC_CHIP_FAMILY=JN517x` to build for a JN517x microcontroller
- `JENNIC_CHIP=JN5169` to build for a JN5169 microcontroller
- `JENNIC_CHIP=JN5168` to build for a JN5168 microcontroller
- `JENNIC_CHIP=JN5164` to build for a JN5164 microcontroller
- `JENNIC_CHIP=JN5179` to build for a JN5179 microcontroller
- `JENNIC_CHIP=JN5178` to build for a JN5178 microcontroller
- `JENNIC_CHIP=JN5174` to build for a JN5174 microcontroller

- `DR=DR1199` to build for hardware based around DR1199 expansion board (default)
- `DR=DONGLE` to build for hardware based on USB dongle
- `GROUPS=0` to build so that bound commands use unicast transmission
- `GROUPS=1` to build so that bound commands use groupcast transmission
- `ICODES=0` to build so that install codes are not used
- `ICODES=1` to build so that install codes are used

5.5 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 Section 5.7.

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

app_start.c manages the JN516x/7x 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 as '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.

irq_JN516x.s defines which of the JN516x hardware interrupts are supported and serviced, and at which priority. This is defined by two tables - an interrupt priority table and a table of handler functions. This file is not used for JN517x.

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.

5.5.1 Command Line Build Options

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

- `JENNIC_CHIP_FAMILY=JN516x` to build for a JN516x microcontroller
- `JENNIC_CHIP_FAMILY=JN517x` to build for a JN517x microcontroller
- `JENNIC_CHIP=JN5169` to build for a JN5169 microcontroller
- `JENNIC_CHIP=JN5168` to build for a JN5168 microcontroller
- `JENNIC_CHIP=JN5164` to build for a JN5164 microcontroller
- `JENNIC_CHIP=JN5179` to build for a JN5179 microcontroller
- `JENNIC_CHIP=JN5178` to build for a JN5178 microcontroller
- `JENNIC_CHIP=JN5174` to build for a JN5174 microcontroller
- `ICODES=0` to build so that install codes are not used
- `ICODES=1` to build so that install codes are used
- `APP_NTAG_ICODE=0` to build without NTAG/NFC (ZigBee Installation Code format) support
- `APP_NTAG_ICODE=1` to build with NTAG/NFC (ZigBee Installation Code format) support (this is the default option)
- `APP_NTAG_AES=0` to build without NTAG/NFC (AES Encryption format) support (this is the default option)
- `APP_NTAG_AES=1` to build with NTAG/NFC (AES Encryption format) support

5.6 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 Section 5.7.

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

app_start.c manages the JN516x/7x 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.

irq_JN516x.s defines which of the JN516x hardware interrupts are supported and serviced, and at which priority. This is defined by two tables - an interrupt priority table and a table of handler functions. This file is not used for JN517x.

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.

5.6.1 Command Line Build Options

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

- `JENNIC_CHIP_FAMILY=JN516x` to build for JN516x microcontrollers
- `JENNIC_CHIP_FAMILY=JN517x` to build for JN517x microcontrollers
- `JENNIC_CHIP=JN5169` to build for a JN5169 microcontroller
- `JENNIC_CHIP=JN5168` to build for a JN5168 microcontroller
- `JENNIC_CHIP=JN5164` to build for a JN5164 microcontroller
- `JENNIC_CHIP=JN5179` to build for a JN5179 microcontroller
- `JENNIC_CHIP=JN5178` to build for a JN5178 microcontroller
- `JENNIC_CHIP=JN5174` to build for a JN5174 microcontroller
- `GROUPS=0` to build so that bound commands use unicast transmission
- `GROUPS=1` to build so that bound commands use groupcast transmission
- `ICODES=0` to build so that install codes are not used

- `ICODES=1` to build so that install codes are used
- `APP_NTAG_ICODE=0` to build without NTAG/NFC (ZigBee Installation Code format) support
- `APP_NTAG_ICODE=1` to build with NTAG/NFC (ZigBee Installation Code format) support (this is the default option)
- `APP_NTAG_AES=0` to build without NTAG/NFC (AES Encryption format) support (this is the default option)
- `APP_NTAG_AES=1` to build with NTAG/NFC (AES Encryption format) support

5.7 Rebuilding the Applications

This section describes how to rebuild the supplied applications, which you will need to do if you customise the applications for your own use.

5.7.1 Pre-requisites

It is assumed that you have installed the relevant NXP development software on your PC, as detailed in Section 2.

In order to build the application, this Application Note [JN-AN-1217] must be unzipped into the directory:

<IDE installation root>\workspace

where **<IDE Installation root>** is the path in which the IDE was installed. By default, this is:

- **C:\NXP\bstudio_nxp** for BeyondStudio
- **C:\NXP\LPCXpresso_<version>_<build>\lpcxpresso** for LPCXpresso

The **workspace** directory is automatically created when you start the IDE.

All files should then be located in the directory:

...\workspace\JN-AN-1217-Zigbee-3-0-Base-Device

There is a sub-directory for each application, each having **Source** and **Build** sub-directories. There will also be sub-directories **JN516x** and **JN517x** containing the project definition files.

5.7.2 Build Instructions

The software provided with this Application Note can be built for the JN516x and JN517x devices.

The applications can be built from the command line using makefiles or from the IDE (BeyondStudio or LPCXpresso) – makefiles and Eclipse-based project files are supplied.

- To build using makefiles, refer to Section 5.7.2.1.
- To build using the IDE, refer to Section 5.7.2.2.

5.7.2.1 Using Makefiles

This section describes how to use the supplied makefiles to build the applications. Each application has its own **Build** directory, which contains the makefiles for the application.

To build an application and load it into a JN516x/7x board, follow the instructions below:

1. Ensure that the project directory is located in

<IDE installation root>\workspace

2. Start an MSYS shell by following the Windows Start menu path:
All Programs > NXP > MSYS Shell
3. Navigate to the **Build** directory for the application to be built and at the command prompt enter an appropriate `make` command for your chip type, as illustrated below.

For example, for JN5169:

```
make JENNIC_CHIP_FAMILY=JN516x JENNIC_CHIP=JN5169 clean all
```

For example, for JN5179:

```
make JENNIC_CHIP_FAMILY=JN517x JENNIC_CHIP=JN5179 clean all
```

The binary file will be created in the **Build** directory, the resulting filename indicating the chip type (e.g. **5169**) for which the application was built.

4. Load the resulting binary file into the board. You can do this from the command line using the JN51xx Production Flash Programmer, as described in Section 4.1.

5.7.2.2 Using the IDE (BeyondStudio for NXP or LPCXpresso)

This section describes how to use the IDE to build the demonstration application.

To build the application and load it into JN516x/7x boards, follow the instructions below:

1. Ensure that the project directory is located in
<IDE installation root>\workspace
2. Start the IDE and import the relevant project as follows:
 - a) In the IDE, follow the menu path **File>Import** to display the **Import** dialogue box.
 - b) In the dialogue box, expand **General**, select **Existing Projects into Workspace** and click **Next**.
 - c) Enable **Select root directory** and browse to the **workspace** directory.
 - d) In the **Projects** box, select the project to be imported, only select the project file appropriate for the chip family and IDE that you are using, and click **Finish**.
3. Build an application. To do this, ensure that the project is highlighted in the left panel of the IDE and use the drop-down list associated with the hammer icon  in the toolbar to select the relevant build configuration – once selected, the application will automatically build. Repeat this to build the other applications.

The binary files will be created in the relevant **Build** directories for the applications.
4. Load the resulting binary files into the board. You can do this using the integrated Flash programmer, as described in the User Guide for the IDE that you are using.

6 Related Documents

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

- ZigBee 3.0 Stack User Guide [JN-UG-3113]
- ZigBee 3.0 Devices User Guide [JN-UG-3114]
- ZigBee Cluster Library (for ZigBee 3.0) User Guide [JN-UG-3115]
- JN51xx Core Utilities User Guide [JN-UG-3116]
- BeyondStudio for NXP Installation and User Guide [JN-UG-3098]
- JN517x LPCXpresso User Guide [JN-UG-3109]
- JN51xx Production Flash Programmer User Guide [JN-UG-3099]

All the above manuals are available as PDF documents from the [ZigBee 3.0](#) page of the NXP web site.

Important Notice

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the *Terms and conditions of commercial sale* of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

All trademarks are the property of their respective owners.

NXP Semiconductors

For the contact details of your local NXP office or distributor, refer to:

www.nxp.com