Application Note: JN-AN-1218

ZigBee 3.0 Light Bulbs

This Application Note provides example applications for light bulbs in a ZigBee 3.0 network that employs the NXP JN516x and/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 hardware kits**
- **A starting point for custom application development using the supplied C source files and associated project files**

The light bulbs described in this Application Note are based on ZigBee device types from the ZigBee Lighting & Occupancy (ZLO) Device Specification.

The Application Note also includes a demonstration of a typical ZigBee 3.0 network with Green Power (GP) support. This solution employs GP devices from the ZigBee PRO Green Power Specification version 1.0, along with ZLO devices.

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 device types for light bulbs on the NXP JN516x and JN517x platforms.

This Application Note provides example implementations of light bulbs that use one of the following device types from the ZigBee Lighting & Occupancy (ZLO) Device Specification:

- Dimmable Light
- Extended Colour Light
- Colour Temperature Light

The above device types are detailed in the *ZigBee 3.0 Devices User Guide [JN-UG-3114]* and the clusters used by the devices are detailed in the *ZigBee Cluster Library (for ZigBee 3.0) User Guide [JN-UG-3115]*.

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.

For information on the ZigBee Green Power (GP) support provided in this Application Note, refer to Section 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 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 8.8).

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 table below with the lighting device types that they support (for device type descriptions, refer to Section 4).

Application Name	Device Type
App_DimmableLight	Dimmable Light
App_ExtendedColorLight	Extended Colour Light
App_ColorTemperatureLight	Colour Temperature Light

Table 1: Example Applications

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 hardware 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 7.

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 *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.*	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 Supported Device Types

As indicated in Section 3, the supported ZLO device types in this Application Note are:

- Dimmable Light
- Extended Colour Light
- Colour Temperature Light

The above devices types are introduced in Sections 4.1, 4.2 and 4.3 respectively.

These lighting devices types must be paired for operation with switch/controller device types. Example applications for the paired device types are provided in the Application Note *ZigBee 3.0 Controller and Switch [JN-AN-1219]*. Two device types can be paired if they support the same cluster (Colour Control, Level Control or On/Off cluster) such that the cluster client on the switch/controller device type can access/control attributes of the cluster server on the lighting device type.

The table below lists the lighting device types (as well as the ZigBee Base Device) and, for each device type, indicates which types of cluster attributes can potentially be written/read.

Device Type	**Attribute Types**					
	OnOff	**Level**	**X & Y Colour**	**Hue & Saturation**	**Colour Temperature**	**Colour Loop**
Dimmable Light	Yes	Yes	No	No	No	No
Extended Colour Light	Yes	Yes	Yes	Yes	Yes	Yes
Colour Temperature Light	Yes	Yes	No	No	Yes	No
Base Device Router	Yes	No	No	No	No	No

Table 3: Lighting Device Types and Accessible Attributes

The table below lists the switch/controller device types (as well as the ZigBee Base Device) and, for each device type, indicates which types of cluster attributes can potentially be controlled (on the lighting device).

Device Type	**Attribute Types**					
	OnOff	**Level**	**X & Y Colour**	**Hue & Saturation**	**Colour Temperature**	**Colour Loop**
Dimmer Switch	Yes	Yes	No	No	No	No
Colour Scene Controller	Yes	Yes	Yes	Yes	Yes	Yes
Control Bridge	Yes	Yes	Yes	Yes	Yes	Yes
Base Device Coordinator	Yes	No	No	No	No	No
Base Device End Device	Yes	No	No	No	No	No

Table 4: Switch/Controller Device Types and Controllable Attributes

The ZLO device types used in this Application Note are outlined below.

4.1 ZLO Dimmable Light

The Dimmable Light is a lighting device that can be switched on or off and the brightness of the light output varied. The device can be controlled by a bound controller device such as a dimmer switch. The complete lists of supported clusters, attributes and commands can be found in the ZigBee Lighting & Occupancy Devices Specification.

4.2 ZLO Extended Colour Light

The Extended Colour Light is a lighting device that can be switched on or off and the brightness of the light output varied. The colour of the light can also be adjusted via the colour commands. The full range of colour control is supported by the Extended Colour Light, XY, Hue and Saturation, Extended Hue and Saturation, Colour Temperature and Colour Loop commands. The device can be controlled by a bound controller device such as a colour controller. The complete lists of supported clusters, attributes and commands can be found in the ZigBee Lighting & Occupancy Devices Specification.

4.3 ZLO Colour Temperature Light

The Colour Temperature Light is a lighting device that can be switched on or off and the brightness of the light output varied. The colour of the light can also be adjusted via the colour temperature commands. The device can be controlled by a bound controller device such as a Colour Controller. The complete lists of supported clusters, attributes and commands can be found in the ZigBee Lighting & Occupancy Devices Specification.

5 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. All the applications run on a JN5169 module on a DR1174 Carrier Board or a JN5179 module on an OM15028 Carrier Board, fitted with a specific expansion board.

5.1 Loading the Applications

The table below lists the application binary files supplied with this Application Note and indicates the JN516x/7x hardware 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**
DimmableLight	**DimmableLight_NtagIcode_JN5169_DR1175.bin**	DR1174 Carrier Board with JN5169 module DR1175 Lighting/Sensor Expansion Board OM15044 NTAG Adaptor Board OM5569/NT322E NTAG Board
ExtendedColorLight	**ExtendedColorLight_NtagIcode_JN5169_DR1175.bin**	
ColorTemperatureLight	**ColorTemperatureLight_NtagIcode_JN5169_DR1175.bin**	
Application	**JN5179 Binary File**	**JN517x-DK005**
DimmableLight	**DimmableLight_NtagIcode_JN5179_DR1175.bin**	OM15028 Carrier Board with JN5179 module DR1175 Lighting/Sensor Expansion Board
ExtendedColorLight	**ExtendedColorLight_NtagIcode_JN5179_DR1175.bin**	
ColorTemperatureLight	**ColorTemperatureLight_NtagIcode_JN5179_DR1175.bin**	

Table 5: 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.

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.

5.2 Commissioning the Network

Before one of the application devices can be used, it must first be commissioned into a network. This is a two-stage process:

1. First the device forms or joins a network, exchanging network parameters and security keys etc. This is called 'Network Steering for a device not on a network'.
2. Then the process of service discovery is performed, in which controller type devices are bound to the light type devices they are to control. This is called 'Finding and Binding'.

There is a further part to Network Steering for a device that is part of the network. This process 'opens' the network for new joiners for 180 seconds. This is initiated by one of the nodes in the network prior to the new device attempting to join the network.

The devices in this Application Note can be commissioned into either a Centralised Trust Centre network or a Distributed network (a network without a Trust Centre).

- In a Centralised network, the Trust Centre is responsible for deciding which devices are allowed onto the network and for provisioning the permitted devices with the appropriate network keys and application-level link keys. This decision is made by the Trust Centre according to its local security policies.
- In a Distributed network, there is no Trust Centre to manage which devices are allowed onto the network. The network key is passed from the parent node to the child node encrypted with the distribute link key.

Commissioning into a network is identical for all device types in this Application Note and is described in the following sections.

5.2.1 Joining a Centralised Network

In order for the devices in this Application Note to join a Centralised network, there must first be an existing network with a Trust Centre in place, and this network must be open for new devices to join. A suitable Coordinator/Trust Centre can be found in either the Application Note *ZigBee 3.0 IoT Control Bridge (JN-AN-1216)* or the Application Note *ZigBee 3.0 Base Device Template (JN-AN-1217)*.

1. If there is no existing network then start a network using a Coordinator device from one of the above two Application Notes.

2. Once there is a Coordinator running, trigger 'Network Steering for a device on a network' in order to open the network for joiners. Depending on the Trust Centre device being used, this is achieved as follows:

- On a Control Bridge, configure the device as a Router, set up the channel mask and run the E_FL_MSG_START_SCAN (0x25) command.
- On a Coordinator board from the *ZigBee 3.0 Base Device Template* Application Note, press the **SW2** button on the DR1199 Generic Expansion Board.

- For a Coordinator dongle from the *ZigBee 3.0 Base Device Template* Application Note, enter 'steer' into the serial interface to send a command from the PC to the dongle.
- On a light device (from this Application Note) that is already part of this network, press the **DIO8/GPIO4** button once or press the **RST/RESET** button 3 times at one-second intervals (both buttons are on the DR1174/OM15028 Carrier Board).
- On a Colour Scene Controller (from the *ZigBee 3.0 Controller and Switch* Application Note), press *** * C**.
- On a Dimmer Switch (from the *ZigBee 3.0 Controller and Switch* Application Note), hold down the **DIO8/GPIO4** button on the DR1174/OM15028 Carrier Board and press the **SW1** button on the DR1199 Generic Expansion Board.

This will cause a Management Permit Join Request to be broadcast to the network to open the Permit Join status for a period of 180 seconds.

3. Now start the device that is to be joined to the network. The device will perform network discovery across the primary and secondary channels, and attempt to associate with any open networks that are discovered. Once associated, the joining device will receive a network key from the Trust Centre via the parent device. After this, the joining device will attempt to update its link key – the new key will be issued by the Trust Centre and sent to the new device via the parent device. If the association or exchange of security keys fail then the new device will resume network discovery on any un-scanned channels to look for other networks to join.

Once this process has successfully completed, the light will be part of the network but not fully commissioned, as it has not been bound to any controlling devices. This is described in Section 5.2.3.

5.2.2 Forming or Joining a Distributed Network

The devices in this Application Note are capable of joining a Distributed network. This can be done either by classical 'discovery and association' or by Touchlink. Light devices in this Application Note support Touchlink as a Target only - this means that they are not capable of initiating Touchlink and bringing other devices into the network.

5.2.2.1 Discovering and Joining an Existing Distributed Network

A device may join an existing network in the same way in which it would join a Centralised network, the only difference being how the security key is exchanged after the association.

1. On one of the devices already on the network, trigger Network Steering as follows:

- On a Router board from the *ZigBee 3.0 Base Device Template* Application Note, press the **DIO8** button on the DR1174 Carrier Board or GPIO4 on the OM15028 Carrier Board.
- On an End Device board from the *ZigBee 3.0 Base Device Template* Application Note, press the **SW2** button on the DR1199 Generic Expansion Board.
- On a light device (from this Application Note) that is already part of this network, press the **RST/RESET** button 3 times at one-second intervals, or press **DIO8/GPIO4** once.
- On a Colour Scene Controller (from the *ZigBee 3.0 Controller and Switch* Application Note), press *** * C**.
- On a Dimmer Switch (from the *ZigBee 3.0 Controller and Switch* Application Note), hold down the **DIO8/GPIO4** button on the Carrier Board and press the **SW1** button on the DR1199 Generic Expansion Board.

This will cause a Management Permit Join Request to be broadcast to the network to open the Permit Join status for a period of 180 seconds.

2. Now start the device that is to be joined to the network. The device will perform network discovery across the primary and secondary channels, and attempt to associate with any open networks that are discovered. Once associated, the joining device will receive a network key from the parent device encrypted with the distributed link key. If the association or exchange of security keys fail then the new device will resume network discovery on any un-scanned channels to look for other networks to join.

Once this process has successfully completed, the light will be part of the network but not fully commissioned, as it has not been bound to any controlling devices. This is described in Section 5.2.3.

5.2.2.2 Forming or Joining a Distributed Network using Touchlink

Touchlink is a procedure for forming, adding to and maintaining a Distributed network. It is based on the proximity of the devices involved and is performed at low power to limit range. There are two type of Touchlink device: Initiator and Target. The devices in this Application Note are all Targets and need to be interrogated by Initiator devices in order to complete a Touchlink exchange. For an example of a Touchlink Initiator, refer to the Colour Scene Controller in the Application Note *ZigBee 3.0 Controller and Switch (JN-AN-1219)*.

Touchlink can be used to form a new Distributed network, if the Initiator is factory-new and at least one of the Initiator and Target is a ZigBee Router device. If the Initiator is not factory-new then the Initiator will use a Join command to add the Target to the network, passing its network parameters and network key to the new device.

The Touchlink process is as follows:

1. To add one of the light devices into a network or form a new network, bring the chosen Initiator device into close proximity (approximately 10cm) of the Target device, then trigger the Touchlink process on the Initiator. For the Target devices in this application, there is no need to take any action to be a Touchlink Target other than being in close proximity of the Initiator.
2. The Initiator will then send a series of Touchlink Scan requests across the primary channels. If no Target is found on the primaries, it will then send the requests on the secondary channels. After receiving a Scan request, the light device will reply with a Touchlink Scan response. Once the Initiator has completed scanning on the primary and secondary channels, for further processing it will select the responding Target device that it considers to be the closest.
3. The Initiator may then send a Touchlink Identify command to the Target, in order to visually confirm which device was selected as the Target, as well as device information requests to further interrogate the Target regarding its supported endpoints.
4. The Initiator will then send either a Touchlink Network Start command or a Touchlink Router Join Request to the Target. These commands will contain the network parameters and network key of the network to be joined or formed. The light device will then start as a Router on the network.

Following the successful completion of Touchlink, unlike the ‘discovery and associate’ joining process previously described, the target light will be bound to any matching operational clusters on the initiating device. The Initiator will have created entries in its Binding table for the matching clusters and will be able to control the light through these bindings. However, the light will only be bound to the one Initiator that performed the Touchlink process. If the light is also to be controlled by another controller type device then either a further round of Touchlinking with that controller will be necessary, or ‘Finding and Binding’ will be required (as described in Section 5.2.3).

5.2.2.3 Joining an Existing Network using NFC

A light node 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).

5.2.3 Finding and Binding

‘Finding and Binding’ is the process in which ‘controller’ type devices (e.g. switches) are bound to ‘controlled’ type devices (e.g. lights). Bindings are created in the Binding table of the controller device for any matching operational clusters of the controlled device. An operational cluster is a cluster that directly controls the output of a device – for example, On/Off, Level Control and Colour Control are operational clusters. Support clusters, such as Groups, Identify and Commissioning, are not operational clusters.

As controlled devices, those in this Application Note implement the Finding and Binding process as targets.

The Finding and Binding process is as follows:

1. Trigger Finding and Binding as a target on a light of this Application Note by doing either one of the following on the DR1174/OM15028 Carrier Board:
   - Press the **DIO8/GPIO4** button
   - Press the **RST/RESET** button 3 times at one-second intervals

   This will cause the device to identify itself for 180 seconds.
2. Once all the target devices are identifying, trigger Finding and Binding as an initiator on the controller device that you wish to bind to the target controlled devices.
3. The initiator will then send out an Identify Query Request in order to use the responses to create a list of devices that are currently identifying. Each device in this list will be sent a Simple Descriptor Request to determine the supported clusters on the target device.
4. For any matching operational clusters, bindings will be created in the initiator's Binding table. Depending on the type of binding being created, an Add Group Command may be sent to the target.
5. Once a binding has been created, the initiator may send an Identify command with time of zero to the target to take it out of identify mode and stop it responding to any further Identify Query Requests.

5.2.4 Network Steering for a Device on a Network

Network Steering for a device already on a network is the process whereby a device opens up the network for other devices to join the network through discovery and association. It does not affect the ability of devices to join through Touchlink.

For the devices in this Application Note, Network Steering for a device on the network can be triggered in either one of the following ways on the DR1174/OM15028 Carrier Board:

- Pressing the **DIO8/GPIO4** button
- Pressing the **RST/RESET** button 3 times at one-second intervals

This will cause the device to broadcast to the network a Management Permit Joining Request with the joining time set to 180 seconds. This will cause the Routers and Coordinator in the network to set their Permit Join bit in their beacons, advertising the network as open to joiners. After the expiry of the 180-second time window, the Permit Join bit will be cleared and the network will close to joiners.

5.2.5 Touchlink Stealing a Device from a Network

The Touchlink process allows the possibility of a Target device on one network being asked to leave that network and to either join or form a network with the new Touchlink Initiator. The application implemented on the Target device determines whether it will comply with such requests. The light devices in this Application Note have been implemented such that once part of a network, they will refuse such 'network stealing' attempts unless the device has first been put into Finding and Binding mode as a target.

Therefore, to allow these Touchlink stealing attempts, put the light device into Finding and Binding mode as described in Step 1 of Section 5.2.3. The device will then self-identity for 180 seconds and will accept Touchlink stealing until this period expires, after which stealing will not be allowed again.

5.2.6 Start-up Behaviour

All light devices in this Application Note are ZigBee Router devices. On start-up, if a device is factory-new, it will attempt to discover and join any open networks, either centralised or distributed. If this fails, the device will remain with its radio receiver ready to become a Touchlink Target. If a further attempt at discovery is required then this can be done by pressing the **RST/RESET** button on the Carrier Board.

If the light device is already part of a network, it just restarts using the stored network parameters and continues on the operational channel.

5.2.7 Performing a Factory Reset

The ZigBee 3.0 specification provides a variety of methods for resetting a device with impacts ranging from resetting application clusters but leaving network parameters intact to a full factory reset causing the device to leave the network and removing all persistent data, network parameters, keys, bindings etc.

All reset methods maintain the value of the outgoing network frame counter across the reset. This parameter is never cleared out by a reset.

5.2.7.1 Factory Reset by Local Means

Either one of the following methods can be used to factory reset a device in this Application Note using buttons on the Carrier Board:

- Hold down the **DIO8/GPIO4** button and press the **RST/RESET** button
- Press the **RST/RESET** button 7 times at one-second intervals

Following either of these stimuli, the device will send a self-leave indication to the network and perform the 'leave without rejoin' procedure. This will clear all network parameters and keys, and reset the ZigBee stack to the factory-new state. In addition, the application will clear out all persisted application data and states, returning them to factory-new values, removing any stored scenes, reporting configurations and light start-up parameters.

There will then be a software reset, causing the device to re-boot. This will trigger an attempt at discovery and association, in line with the factory-new start-up behaviour.

5.2.7.2 Factory Reset by Touchlink

A device that is part of a Distributed network (that is, not a Centralised Trust Centre network) can be requested to perform a factory reset via the Touchlink mechanism. It is the application's responsibility to determine whether or not to comply with this factory reset request. The devices in this Application Note will comply with such requests.

To perform a Touchlink factory reset of a device in this Application Note, bring the Target device into close proximity of a Touchlink Initiator and trigger a 'Touchlink with factory reset' in line with the application methods of the Initiator.

Following the exchange of Touchlink Scan requests and responses, and the receipt of a valid Touchlink factory reset command, the Target device will send a self-leave indication to the network and perform the 'leave without rejoin' procedure. This will clear all network parameters and keys, and reset the ZigBee stack to the factory-new state. In addition, the application will clear out all persisted application data and states, returning them to factory-new values, removing any stored scenes, reporting configurations and light start-up parameters.

There will then be a software reset, causing the device to re-boot. This will trigger an attempt at discovery and association, in line with the factory-new start-up behaviour.

5.2.7.3 Factory Reset by Network Leave Command

Following the receipt of a valid 'network leave without rejoin' request, a device will remove all ZigBee persistent data and keys, then the application data will be returned to a factory-new state, and a software reset will trigger the device to re-boot.

5.2.7.4 Factory Reset by Mgmt Leave Request ZDO Command

Following receipt of a valid 'management leave request ZDO' command, a device will remove all ZigBee persistent data and keys, then the application data will be returned to a factory-new state, and a software reset will trigger the device to re-boot.

5.2.7.5 Reset by Basic Cluster

The 'Reset to Factory Defaults' command of the Basic cluster provides a method of resetting the attributes of all ZCL clusters to their default values. All ZigBee network parameters, bindings and groups will remain intact, and the device will continue to operate on the network.

The Scenes table will be cleared out, any configured reports will be replaced by the default reporting configuration, and the start-up OnOff Level and Colour Temperature values will be returned to their default values.

5.3 Configurations Parameters

There are several attributes that control the start-up state of a light and whether or not commands that alter the operational attributes of the light are obeyed if the light is in the Off state (these attributes are new for Zigbee 3.0 and the ZLO Devices specification).

Cluster	Attribute	Function	Defined Behaviour
OnOff	0x4003	Determines the OnOff attribute at power-up.	Restore to the status from before power was removed.
Level Control	0x000F	Determines whether level can be changed if light is Off. Determines whether level is coupled to colour temperature.	Level will not change when Off. Colour temperature and level are coupled.
Level Control	0x4000	Determines the light brightness at power-up.	Restore the previous brightness.
Colour Control	0x000F	Determines whether the colour can be changed if the light is Off.	Colour cannot be changed when Off.
Colour Control	0x4010	Determines the colour temperature at start-up.	Restore the previous colour temperature.

Note: These default behaviours are defined in the **zcl options.h** file and can be changed at compile-time, if required. For full details of the available options, refer to the ZigBee Lighting & Occupancy Devices Specification.

The attributes are writeable and persisted in non-volatile memory, so they can be changed at run-time to suit the installed application.

The options that control whether the level or colour can be changed when the light is Off allow the lights to be configured to match the behaviour of either the previous ZigBee Light Link Specification (no change if Off) or ZigBee Home Automation Specification (change if Off). In addition, all cluster commands in the Level Control and Colour Control clusters that change the level or colour now carry optional parameters that define whether the local operational mode should be over-ridden by this command.

5.4 Attribute Reporting Configurations

The ZigBee Lighting & Occupancy Devices Specification mandates that some attributes are reportable - that is, once bound to a device to receive these reports, the attributes will periodically report their status and also report changes to their status. To this end, each reportable attribute has a default configuration to determine this schedule. This default configuration can be altered by the report receiving device at run-time, as required. The report configuration is stored in persistent memory.

Cluster	Attribute	Attribute ID
OnOff	OnOff	0x0000
Level Control	Current Level	0x0000
Colour Control	Current Hue	0x0000
Colour Control	Current Saturation	0x0001
Colour Control	Current X	0x0003
Colour Control	Current Y	0x0004
Colour Control	Colour Temperature	0x0007

6 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.

6.1 Overview

Support for the OTA Upgrade cluster as a client has been included for the Extender Colour Light, Colour Temperature Light and Dimmable Light devices. In order to build with these options, add `OTA=1` to the command line before building. This will add the relevant functionality to the lights and invoke post-build processing to create a bootable image and two upgrade images. The produced binaries will be stored in the **OTA_build** 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.

- If built for the JN5168 device then external Flash memory will be used to store the upgrade image before replacing the old one.
- If built for the JN5169 or JN5179 device then the internal Flash memory will be used to store the upgrade image by default. External Flash memory could be used if desired.

The Application Note *ZigBee 3.0 IoT Control Bridge (JN-AN-1216)* has OTA server functionality built into it. A device called OTA_server is provided to host the upgrade images that the clients will request.

6.2 OTA Upgrade Operation

To implement an OTA upgrade:

1. Build the light application with `OTA=1` in the makefile to enable OTA upgrade (this option is not enabled by default). There is an OTA debug flag defined in the makefile: `CFLAGS += -DDEBUG_APP_OTA`. Uncomment this line if the OTA debug is required.

   The binary files for the light are created in the **OTABuild** folder – bootable binaries have the extension **.bin** and no version suffix, and upgrade binaries have the extension **.ota** or **.bin** and a version suffix. The upgrade image is intended to be loaded into external Flash memory of the OTA server using the JN51xx Production Flash Programmer (JN-SW-4107), as described in Step 4 below. Encrypted upgrade binary images will have a **_ENC** suffix. There are three binaries in a set, with the files having different versions with different headers so that the upgrading of the light can be tested - a bootable image, version 1 (v1), and two upgrade images, versions 2 and 3 (v1 and v2).

2. Program one of the bootable binary files from the **OTABuild** folder into the internal Flash memory of the JN516x/7x device on the Carrier Board of the light node - for example, **ExtendedColorLight_NtagIcode_JN5169_DR1175.bin**. You can do this using the JN51xx Flash Programmer within the relevant IDE. Alternatively, you can use the JN51xx Production Flash Programmer (JN-SW-4107) described in the *JN51xx Production Flash Programmer User Guide (JN-UG-3099).*
3. Form a network with a light node and the Control Bridge in the normal way using Touchlink.
4. Load a **.ota** upgrade image (v2 or v3) into the <u>external</u> Flash memory of the Control Bridge using the JN51xx Production Flash Programmer (JN-SW-4107) - the required command line will be similar to the following:

   ```
   Jn51xxProgrammer –S external –s COM<port> -f <filename>
   ```
5. When viewing the UART output from the light node, the upgraded image should be found and the light node upgraded.

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

6.3 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 0x000 to 0xFFBF. Its use is for the manufacturer to distinguish between devices. In this application, the Image Type is set to the ZigBee Device Type of the bulb - for example, 0x010D for an Extended Colour Light or 0x110D if the image is transferred in an encrypted format. 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.

6.4 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. The encryption key is stored in protected e-fuse and is set by the manufacturer.

For JN5169 and JN517x builds, to use encrypted images the following define must be included as a build option in the **zcl_options.h** file:

```
#define INTERNAL_ENCRYPTED
```

6.5 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 Notifications in the OTA client will need to be modified.

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 Combo Basic** support is provided for the Extender Colour Light, Colour Temperature Light, and Dimmable Light device types. Lights with GP Combo Basic support act as GP sink nodes that can be controlled by both GP switches and ZLO switches, after successful commissioning. These lights support the following GP commands: On, Off, Identify, Move Up with On/Off, Move Down with On/Off, Level Control/Stop. The commands are detailed the in ZigBee Green Power Specification. A Combo Basic device also provides the proxy functionality described below.
- **GP Proxy Basic** support is provided for the Dimmable Light device type. Lights 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-3119)*.

The applications in this Application Note are supplied pre-built with GP support as follows (and can be re-built as described in Section 8.8):

- Dimmable light with GP Combo Basic functionality
- Extended Colour Light with Combo Basic functionality
- Colour Temperature Light Combo Basic functionality
- Dimmable light with GP Proxy Basic functionality

Each GP binary file is located in the **Build** directory of relevant application and indicated by **GpCombo** or **GpProxy** in the filename, as appropriate. The table below lists the GP binary files, where *xx* indicates the chip number (69 or 79).

Application	JN5169 Binary File	JN516x-EK004
DimmableLight	**DimmableLight_NtagIcode_GpCombo_ JN5169_DR1175.bin**	DR1174 Carrier Board with JN5169 module DR1175 Lighting/Sensor Expansion Board OM15044 NTAG Adaptor Board OM5569/NT322E NTAG Board
ExtendedColorLight	**ExtendedColorLight_NtagIcode_GpCombo_ JN5169_DR1175.bin**	
ColorTemperatureLight	**ColorTemperatureLight_NtagIcode_GpCombo_ JN5169_DR1175.bin**	
DimmableLight	**DimmableLight_NtagIcode_GpProxy_ JN5169_DR1175.bin**	
Application	**JN5179 Binary File**	**JN517x-DK005**
DimmableLight	**DimmableLight_NtagIcode_GpCombo_ JN5179_DR1175.bin**	OM15028 Carrier Board with JN5179 module DR1175 Lighting/Sensor Expansion Board
ExtendedColorLight	**ExtendedColorLight_NtagIcode_GpCombo_ JN5179_DR1175.bin**	
ColorTemperatureLight	**ColorTemperatureLight_NtagIcode_GpCombo_ JN5179_DR1175.bin**	
DimmableLight	**DimmableLight_NtagIcode_GpProxy_ JN5179_DR1175.bin**	

7.1 Commissioning Combo Basic Lights and GP Switches

The lights with GP Combo Basic support, which act as GP sink nodes, must be commissioned to operate with the GP source devices (switches). This commissioning can be performed as follows:

1. Put the light into commissioning mode by pressing the **RST/RESET** button on the DR1174/OM15028 Carrier Board 3 times at one-second intervals. The light will then enter commissioning mode and indicate this by flashing.
2. While the light is in commissioning mode, send a Commissioning command from the GP switch that is to control the light (this step is switch-specific).

   The light comes out of commissioning mode after successful commissioning or after a timeout period of 3 minutes. The light stops flashing once it has exited commissioning mode.

Once the light is successfully commissioned (paired), the light can be controlled by the GP switch.

Note: During commissioning, no user action is required on the lights with Proxy Basic support. These devices create tables internally when a pairing is created between Combo Basic device and GP switch, to allow the forwarding of GP messages.

Note: Commissioning mode can be toggled on a light using the **DIO8** button on the DR1174 Carrier Board or the **GPIO4** button on the OM15028 Carrier Board. If the light is in operating mode, pressing this button causes the light to enter commissioning mode. If the light is in commissioning mode, pressing of this button causes the light to exit commissioning mode.

Once a GP switch is commissioned, the switch can be de-commissioned by sending out an encrypted Decommissioning command. The light accepts a Decommissioning command in both commissioning mode and in decommissioning mode.

7.2 Removing Sink /Proxy Table Entries

On a Combo Basic or Proxy Basic device, the Sink table/Proxy table can be removed by pressing the **RST/RESET** button on the Carrier Board twice at one-second intervals. This will erase the Sink table/Proxy table and remove all commissioned GP devices.

7.3 Green Power Configuration Settings

The security keys to be used by the Combo Basic and 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:

- 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 9.

8.1 Touchlink Preconfigured Link Key

The Touchlink Preconfigured Link Key is used to encrypt the network key when passed to joining devices as part of Touchlink operations. The key that should be used in real world applications is a secret key and is supplied by the ZigBee organisation upon application after successfully completing the ZLO Certification process. This key is not supplied in this Application Note.

The supplied applications have been set up to use the Certification Key as defined in the Base Device Behavior (BDB) Specification. Once in possession of the real Touchlink Preconfigured Link Key, the following changes need to be made in order to use this key rather than the Certification Key:

1. In the **zcl_options.h** file, add the following definition:

```
#define TL_SUPPORTED_KEYS ( TL_MASTER_KEY_MASK )
```

2. Copy the ZigBee-supplied key into the definition of *sTLMasterKey* in the file **bdb_link_keys.c** (located in **BDB\Source\Common**). This will make the key available to all applications built using the BDB component.

   Alternatively, you can over-ride the definition in the BDB component by including a definition in the individual device applications (e.g. in **App_DimmableLight.c**) and then including the following definition in **bdb_options.h**:

```
#define BDB_APPLICATION_DEFINED_TL_MASTER_KEY
```

8.2 App_DimmableLight Application Code

This section describes the application code for App_DimmableLight, which is 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 8.8.

App_DimmableLight.c is specific to the Dimmable Light. It includes endpoint registration and constructor, reporting configuration, Basic cluster attribute initialisation, and Identify handler.

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

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

8.3 App_ExtendedColorLight Application Code

This section describes the application code for App_ExtendedColorLight, which is 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 8.8.

App_ExtendedColorLight.c is specific to the Extended Colour Light. It includes endpoint registration and constructor, reporting configuration, Basic cluster attribute initialisation, and Identify handler.

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

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

8.4 App_ColorTemperatureLight Application Code

This section describes the application code for App_ColorTemperatureLight, which is 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 8.8.

App_ColorTemperatureLight.c is specific to the Colour Temperature Light. It includes endpoint registration and constructor, reporting configuration, Basic cluster attribute initialisation, and Identify handler.

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

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

8.5 Common Code

Code common to all light device types is held in the **Common_Light\Source** directory. This code contains the main application event handlers, as well as chip and stack initialisation.

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

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

zlo_light_node.c hosts the event handlers for the application and the ZigBee Base Device callback. This callback receives ZigBee Base Device events and AF Stack events after the Base Device has completed any processing that it requires. These events can then be further processed by the application. These events include data indications that are passed to the ZCL for processing, and network management events, such as Joined or Failed to Join events, in order to keep the application informed of the network state. The application event queue is processed to receive button-press events. Sleep scheduling and polling for data are also handled here.

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

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.zpscfg provides the configuration data to initialise the ZigBee stack. It sets the sizes of the various stack tables, such Neighbour, Routing and Key tables.

DriverBulb is the source for various light-bulb hardware interfaces.

app_buttons.c is the driver software to read the switches on the hardware kit boards and present events to the application.

irq.s defines which of the hardware interrupts are supported, serviced and at which priority. This is defined by two tables - an interrupt priority table and a table of handler functions.

app_light_interpolation.c smooths out the updates to the LED outputs to remove flicker. The ZCL updates the light at 10Hz. This code will further divide the update into 10 steps, updating the bulb at 100Hz.

app_manage_temperature.c manages the chip for changes in temperature to maintain accuracy.

app_ota_client.c provides the application part of the OTA client. This manages finding the OTA server, polling it for new images, downloading the image and invalidating the old image and booting on the new image.

app_power_on_counter.c counts how many times the device is switched on within a short period, generates application events to trigger Network Steering or a factory-reset depending on how many.

app_reporting.c manages the setting up of the default reporting configuration, updates and saves the configuration if it is changed remotely. The actual sending of reports is managed by the ZCL itself.

app_scenes.c provides the application interface between the ZCL and Persistent Data Manager (PDM). It contains a routine to save and restore scene cluster data to the PDM. The actual handling of scenes and sending scene data to the light-bulb hardware is managed by the ZCL.

PDM_IDs.h provides unique identifiers for all persistent data records used by the PDM

app_green_power.c manages the ZigBee Green Power features of the light-bulbs.

8.6 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.

8.7 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.

8.8 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.

8.8.1 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-1218] 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-1218-Zigbee-3-0-Light-Bulb

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.

8.8.2 Build Instructions

The applications provided in this Application Note can be built for both JN516x and JN517x.

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

- To build using makefiles, refer to Section 8.8.2.1.
- To build using the IDE, refer to Section 8.8.2.2.

8.8.2.1 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.

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

- `JENNIC_CHIP_FAMILY=JN516x` to build for a JN516x microcontrollers
- `JENNIC_CHIP_FAMILY=JN517x` to build for a 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
- `OTA=0` to build without OTA client
- `OTA=1` to build with OTA client

- `OTA_ENCRYPTED=0` to build OTA images without encryption
- `OTA_ENCRYPTED=1` to build OTA images with encryption
- `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
- `GP_SUPPORT=1` to build with Green Power Combo
- `GP_SUPPORT=1 GP_DEVICE=PROXY_BASIC` to build with Green Power Proxy Basic

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 5.1.

8.8.2.2 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.

9 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 Device User Guide [JN-UG-3114]
- ZigBee Cluster Library (for ZigBee 3.0) User Guide [JN-UG-3115]
- ZigBee Green Power User Guide [JN-UG-3119]
- 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

NXP Semiconductors

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

www.nxp.com