Application Note: JN-AN-1220

ZigBee 3.0 Sensors

This Application Note provides example applications for sensors 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 sensors described in this Application Note are based on ZigBee device types from the ZigBee Lighting & Occupancy (ZLO) Device Specification.

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 sensors on the NXP JN516x and JN517x platforms.

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

- Light Sensor
- Occupancy Sensor
- Light, Temperature & Occupancy Sensor (combination device type)

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]*. The Light, Temperature & Occupancy Sensor is a combination device type based on the Light Sensor device type with the addition of the Temperature Measurement cluster and Occupancy Sensing cluster.

> ⓘ **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.

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 5.8).

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 following table. For each application, the table indicates the required device type as well as the device type (on another node) with which it can be paired for operation.

Application	Device Type	Paired Device Type
App_LightSensor	Light Sensor	Control Bridge
App_OccupancySensor	Occupancy Sensor	Control Bridge
App_LightTemperatureOccupancySensor	Light, Temperature & Occupancy Sensor *	Control Bridge

Table 1: Example Applications and Device Types

* Light Sensor device type with addition of Temperature Measurement and Occupancy Sensing clusters

The Control Bridge is described in in the Application Note *ZigBee Control Bridge [JN-AN-1216]*.

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-EK004 Evaluation Kit.

- 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 *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 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 on a JN5179 module on an OM15028 Carrier Board, fitted with a specific expansion board.

4.1 Loading the Applications

The table below lists the application binary files supplied with this Application Note and indicates the 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 Hardware
LightSensor	**LightSensor_ NtagIcode_JN5169_DR1175.bin**	DR1174 Carrier Board with JN5169 module DR1175 Lighting/Sensor Expansion Board OM15044 NTAG Adaptor Board OM5569/NT322E NTAG Board
LightTemperature OccupancySensor	**LightTemperatureOccupancySensor_ NtagIcode_JN5169_DR1175.bin**	
OccupancySensor	**OccupancySensor_ NtagIcode_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
LightSensor	**LightSensor_ NtagIcode_JN5179_DR1175.bin**	OM15028 Carrier Board with JN5179 module DR1175 Lighting/Sensor Expansion Board
LightTemperature OccupancySensor	**LightTemperatureOccupancySensor_ NtagIcode_JN5169_DR1175.bin**	
OccupancySensor	**OccupancySensor_ NtagIcode_JN5169_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.

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.

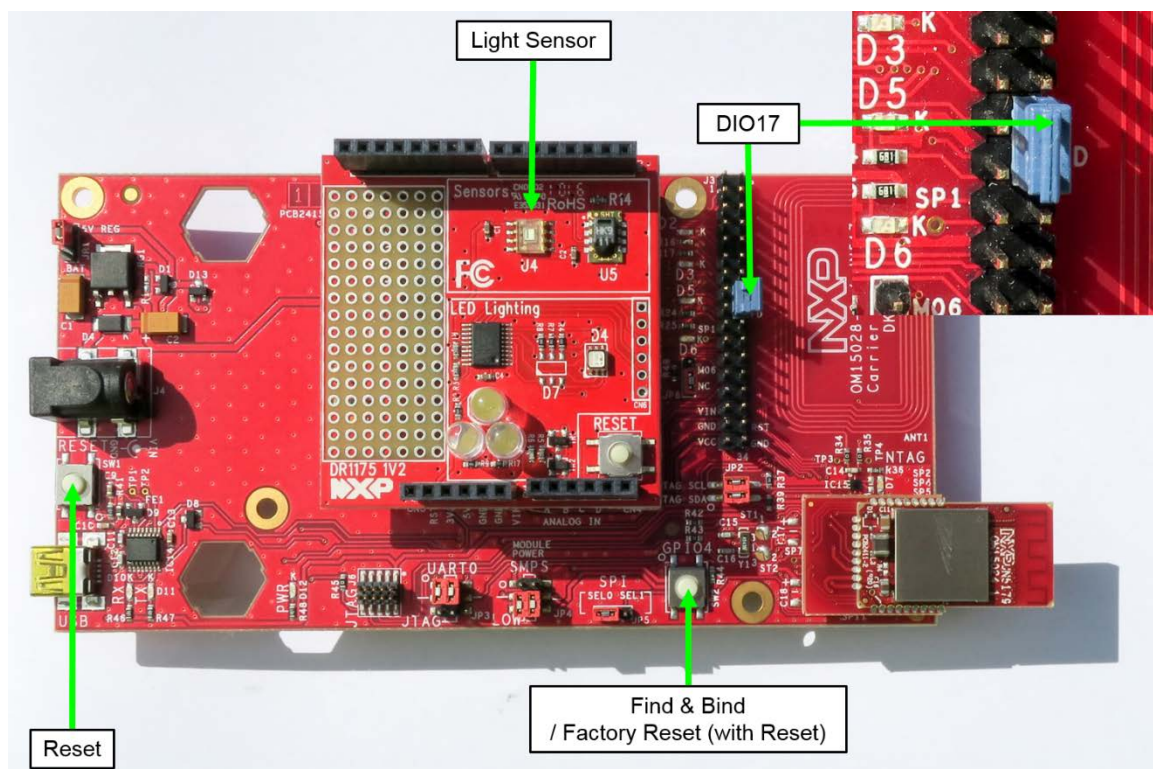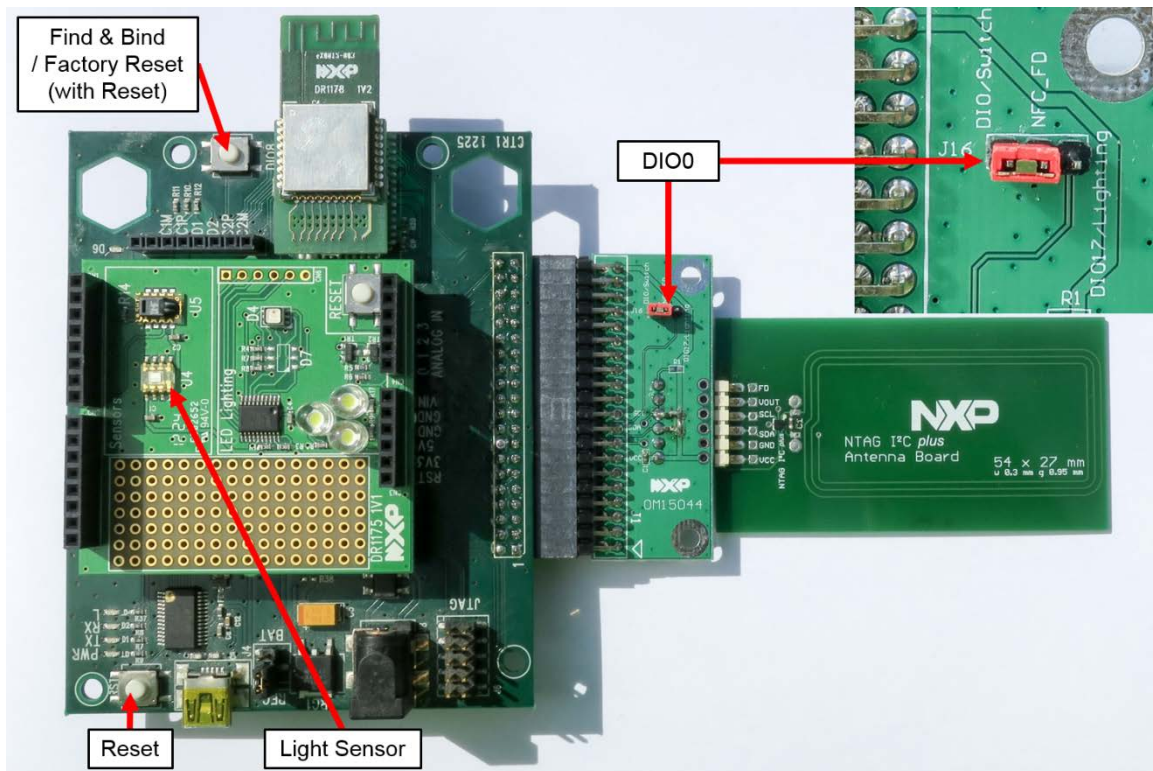## 4.2 Using the LightSensor Application

This section describes how to commission and operate the LightSensor application in a ZigBee 3.0 network. To use this application, you must have programmed the relevant application binary into the JN5169/79 module on a Carrier Board fitted with the DR1175 Lighting/Sensor Expansion Board, as described in Section 4.1:

- **LightSensor_NtagIcode_JN5169_DR1175.bin** for a JN5169 module
- **LightSensor_NtagIcode_JN5179_DR1175.bin** for a JN5179 module

4.2.1 Light Sensor Device Functionality

The Light Sensor can be used to provide regular illuminance measurements (from the Illuminance Measurement cluster) to a control application that adjusts the level of light emitted by light nodes – these lights can be any ZLO Lighting devices that support the Level Control cluster. The control application resides on the device to which the Light Sensor is bound, and this device forms the lights into a group for synchronous control.

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

Find & Bind / Factory Reset (with Reset)

DIO0

Reset

Light Sensor

Light Sensor

DIO17

Reset

Find & Bind / Factory Reset (with Reset)

4.2.2 Clearing Context Data on the Device

When loading the application for the first time, any persistent context data must be cleared in the device.

The context data can be cleared by pressing and releasing the **RST/RESET** button while holding down the **DIO8/GPIO4** button (both buttons are on the Carrier Board).

A reset is indicated by the white LED cluster on the Lighting/Sensor Expansion Board illuminating for a second and switching off.

4.2.3 Commissioning

The commissioning operations for this device, such as forming/joining a network and binding, are the same as for the other device types described in this Application Note and are detailed in Section 4.5. To incorporate the device into a ZigBee 3.0 network, work through Section 4.5 and then return to this point for device operation.

4.2.4 Operation

There are two modes of operation of the Light Sensor, as follows:

- **Normal Mode:** The Light Sensor sleeps and wakes up every 61 seconds (`ZLO_MAX_REPORT_INTERVAL`), when it obtains a new light reading, updates the `u16MeasureValue` attribute with the new reading and sends a periodic report. This is done to reduce the current consumption, which increases the battery life.

- **Keep-alive Mode:** The Light Sensor is permanently active and polling its parent every second. If the light level changes since the last reading by at least `LIGHT_SENSOR_MINIMUM_REPORTABLE_CHANGE` then the `u16MeasuredValue` attribute is updated and the Light Sensor sends out an attribute report after 1 second. If there is no change, it will send a report every 60 seconds.

   The light sensor will be reading the value of the driver at the rate of once every `LIGHT_SENSOR_SAMPLING_TIME_IN_SECONDS` (5 seconds) and updating its `u16MeasuredValue` attribute accordingly. Thus, the changes in light sensor reading should be done only at intervals of 5 seconds.

   If there is any change in the `u16MeasuredValue` attribute, the sensor will send out a report after 1 second (`ZLO_MIN_REPORT_INTERVAL`), otherwise every 61 seconds (`ZLO_MAX_REPORT_INTERVAL`).

The Light Sensor can but put into 'keep-alive' mode by power-cycling 3 times at one-second intervals (otherwise you will not allow the right task to be triggered).

ⓘ **Note 1:** As a sleepy End Device, the Light Sensor goes through a sleep/wake cycle. If an attribute change occurs just before the device enters sleep mode, the attribute report will be sent out on the next wake-up and there will therefore be a delay.

ⓘ **Note 2:** If the Light Sensor is to report frequently, it is recommended that the bound transmission management feature is enabled in the ZCL by defining CLD_BIND_SERVER in the **zcl_options.h** file. This feature and the required resources are described in the *ZigBee Cluster Library User Guide (JN-UG-3115)*.

4.2.5 Light Level Measurement

The calculation detailed below determines the light level that should be produced by the light for a given illuminance measured at the Light Sensor.

At the Light Sensor, the maximum lux level (as measured by the ALS driver on the DR1175 board) is 4015 and the minimum lux level is 1.

At the light, the light level to be produced is represented as a value in the range 0 to 255 (this is the `u8CurrentLevel` attribute of the Level Control cluster).

A divisor is defined which is used (later) in calculating the produced light level from the measured Lux value:

ILLUMINANCE_LUX_LEVEL_DIVISOR = 4015/254 ~ 16

(i.e. ILLUMINANCE_MAXIMUM_LUX_LEVEL/CLD_LEVELCONTROL_MAX_LEVEL)

The value of the `u8CurrentLevel` attribute (of the Level Control cluster) for the light is then calculated as follows:

CLD_LEVELCONTROL_MAX_LEVEL - (u16MeasuredLux/ILLUMINANCE_LUX_LEVEL_DIVISOR)

where `u16MeasuredLux` is the attribute of the Illuminance Measurement cluster reported to the light.

Therefore:

- When the measured illuminance is at its maximum value of 4015 lux, the light level to be produced by the light is 4

- When the measured illuminance is at its minimum value of 1 lux, the light level to be produced by the light is 254

> **(i)** **Note:** The Light Sensor needs a light source to measure the correct thresholds for proper operation (sensitivity increases based on the amount of light falling on sensor).

4.2.6 LED Indication of Different States

The following table lists the different behaviours of LED D4 on the Lighting/Sensor Expansion Board and the corresponding states of the node.

Light/LED Indication
LED D4 flashes 1 second ON, 1 second OFF: Joining or re-joining the network
LED D4 flashes 250ms ON, 250ms OFF: Keep-alive mode
LED D4 OFF: Node is in the network or sleeping
LED D4 flashes 500ms ON, 500ms OFF: Initiator mode active

4.2.7 Sensing Clusters

The Light Sensor uses the Illuminance Measurement cluster to hold its results. The Illuminance Measurement cluster contains the "Measured Value" attribute which is used to store the light level measured by the sensor. Binding to this cluster (Id **0x0400**) will enable a remote device to receive reports when the value of this attribute on the sensor changes.

4.3 Using the OccupancySensor Application

This section describes how to commission and operate the OccupancySensor application in a ZigBee 3.0 network. To use this application, you must have programmed the relevant application binary into the JN5169/79 module on a Carrier Board fitted with the DR1199 Generic Expansion Board, as described in Section 4.1:

- **OccupancySensor_NtagIcode_JN5169_DR1199.bin** for a JN5169 module
- **OccupancySensor_NtagIcode_JN5179_DR1199.bin** for a JN5179 module

4.3.1 Occupancy Sensor Device Functionality

The functionality of the OccupancySensor application is described and illustrated below.
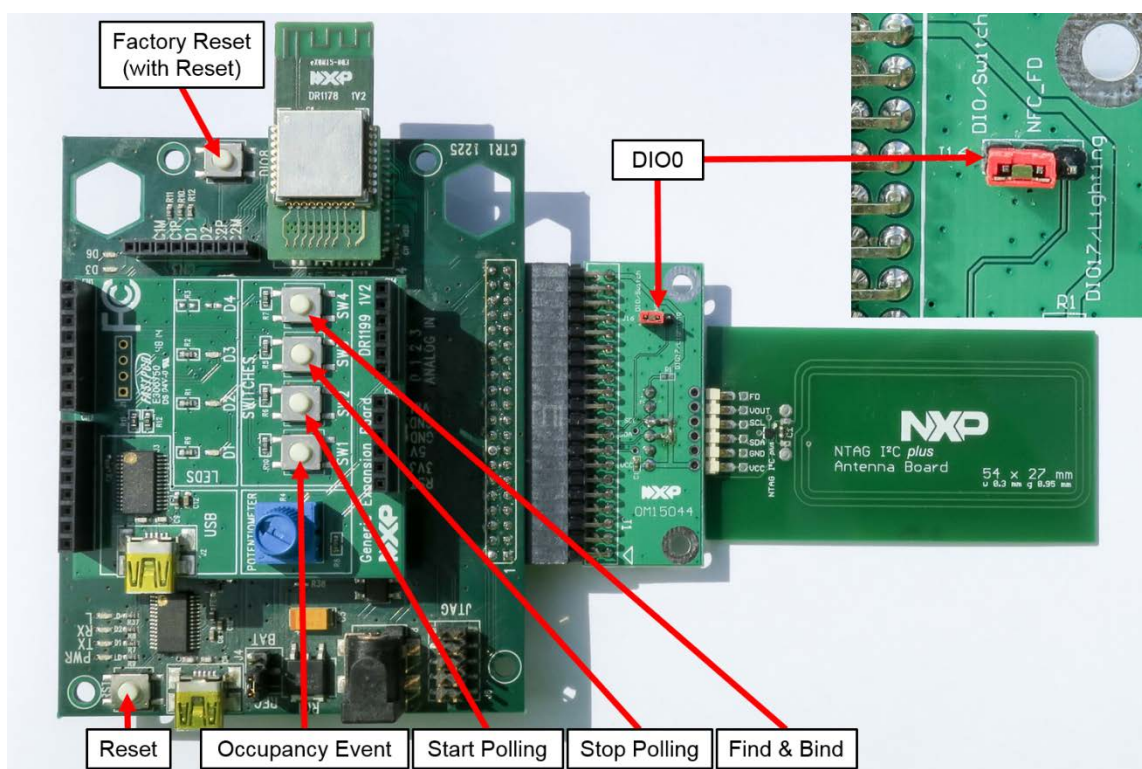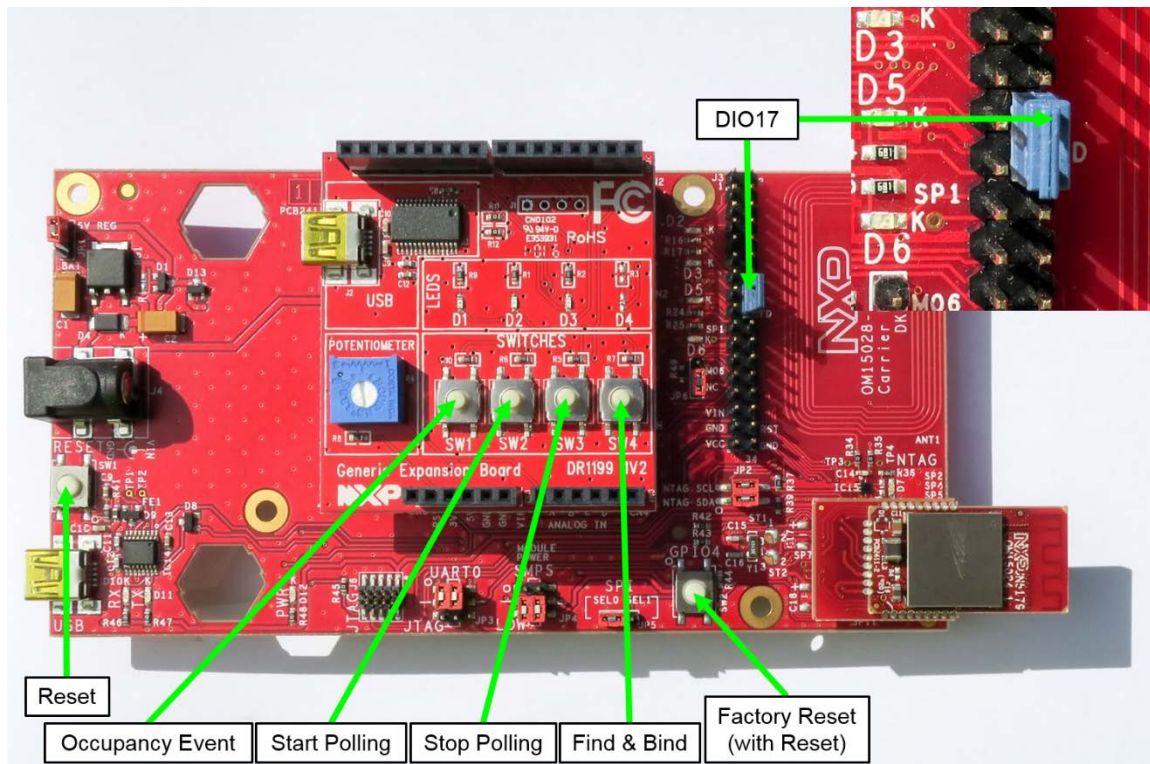
LED Name	Physical LED on Board
LED1	D1 on DR1199 expansion board
LED2	D2 on DR1199 expansion board
LED3	D2 on OM15028 carrier board
	D6 on DR1174 carrier board

Table 4: LED Hardware Mapping

4.3.2 Clearing Context Data on the Device

When loading the application for the first time, any persistent context data must be cleared in the device.

The context data can be cleared by pressing and releasing the **RST/RESET** button while holding down the **DIO8/GPIO4** button (both buttons are on the Carrier Board).

4.3.3 Commissioning

The commissioning operations for this device, such as forming/joining a network and binding, are the same as for the other device types described in this Application Note and are detailed in Section 4.5. To incorporate the device into a ZigBee 3.0 network, work through Section 4.5 and then return to this point for device operation.

4.3.4 Operation

Occupancy events are simulated by pressing a button on the board - button **SW1** is assigned as the output of a virtual PIR detector. There are two types of virtual occupancy sensor: Open Collector and PWM. They are defined in the makefile under "PIR Sensor Type". The different sensor types and how to use them are detailed below.

Note that for both types of sensor, LED1 on the Generic Expansion Board is used as an indicator of the state of the `u8Occupancy` attribute of the Occupancy Sensing cluster:

- If the `u8Occupancy` attribute is 0 (i.e. Unoccupied), the LED1 is OFF

- If the `u8Occupancy` attribute is 1 (i.e. Occupied), the LED1 is ON

Also note the following uses of LED2 and LED3 on the Generic Expansion Board:

- LED3 will flash during a 'Finding and Binding' operation started using the button **SW4** - if it continues to flash after completing the 'Finding and Binding' then press the button again.

- LED2 and LED3 may flash intermittently to indicate the operational state when the JN516x/7x module is awake for sampling or re-joining.

Open Collector Sensor

An Open Collector sensor outputs a constant digital high/low signal for occupancy. In this demonstration, 'occupied' is represented by digital low and 'unoccupied' is represented by digital high.

To define a sensor as an Open Collector, uncomment the compile flag `PIR_TYPE_OPEN_COLLECTOR` in the makefile.

Simulating Unoccupied to Occupied Event:

To move the sensor from the unoccupied to occupied state, press and hold down the button **SW1** on the Generic Expansion Board.

Simulating Occupied to Unoccupied Event:

Once in the occupied state, to simulate an 'unoccupied' event, release the button **SW1**. If no further occupancy event is simulated by pressing the **SW1** button within a certain timeout period (180 seconds, by default), the sensor will automatically move to the unoccupied state.

> (i) **Note:** The timeout period can be customised using the macro `APP_OCCUPANCY_SENSOR_OCCUPIED_TO_UNOCCUPIED_DELAY`.

Once in the occupied state, the sensor can be kept in the occupied state with a single button-press - a single transition of **SW1** will reset the timer that keeps track of the timeout defined by `APP_OCCUPANCY_SENSOR_OCCUPIED_TO_UNOCCUPIED_DELAY`. This feature is provided to simulate maintaining the occupied state.

PWM Sensor

A PWM sensor toggles its digital output between high and low while occupied.

Simulating Unoccupied to Occupied Event:

To move the sensor from the unoccupied to occupied state, press the button **SW1** on the Generic Expansion Board a certain number of times (5, by default) within a certain timeout period (10 seconds, by default).

> **(i)** **Note 1:** The number of button-presses required can be customised using the `APP_OCCUPANCY_SENSOR_UNOCCUPIED_TO_OCCUPIED_THRESHOLD` macro.
>
> **(i)** **Note 2:** The timeout period can be customised using the `APP_OCCUPANCY_SENSOR_UNOCCUPIED_TO_OCCUPIED_DELAY` macro.

Simulating Occupied to Unoccupied Event:

Once in the occupied state, if no further occupancy event is simulated by pressing the **SW1** button within a certain timeout period (180 seconds, by default), the sensor will automatically move to the unoccupied state.

> **(i)** **Note:** The timeout period can be customised using the macro `APP_OCCUPANCY_SENSOR_OCCUPIED_TO_UNOCCUPIED_DELAY`

Once in the occupied state, the sensor can be kept in the occupied state by repeating the simulated occupancy event. This will reset the timer that keeps track of the timeout defined by `APP_OCCUPANCY_SENSOR_OCCUPIED_TO_UNOCCUPIED_DELAY`.

Additional Sensor Functionality

The Occupancy Sensor will report its attributes if it is bound to at least one device, the occupancy state is 'occupied' and one of the following actions occurs:

- It has joined a new network
- It has rejoined the network
- It has exited 'Finding and Binding'

The Occupancy Sensor will start a timer for the number of seconds defined by the macro `APP_OCCUPANCY_SENSOR_OCCUPIED_TO_UNOCCUPIED_DELAY` if the occupancy state is 'unoccupied' and one of the following actions occurs:

- It has joined a new network
- It has rejoined the network
- It has exited 'Finding and Binding'

Attribute Reporting

Attribute Reporting can be configured optionally by defining the macro `ZLO_SYSTEM_MIN_REPORT_INTERVAL` to be greater than zero. If defined to be greater than zero, the Occupancy Sensor reports the Occupancy attribute value to the light (to which it is bound) after 1 second (or `ZLO_SYSTEM_MIN_REPORT_INTERVAL`) if there is any change in the attribute or, otherwise, every 60 seconds (or `ZLO_SYSTEM_MAX_REPORT_INTERVAL`). On receiving the attribute report:

- If the `u8Occupancy` attribute is 0 (i.e. Unoccupied), the light will switch OFF

- If the `u8Occupancy` attribute is 1 (i.e. Occupied), the light will switch ON

4.3.5 Sleep Options

The Occupancy Sensor is a sleeping End Device and will, by default, attempt to sleep with RAM held and the 32kHz oscillator on, whenever possible. It will wake up every `ZLO_SYSTEM_MAX_REPORT_INTERVAL`, which by default is 60 seconds.

Enabling Deep Sleep

The Occupancy Sensor can be configured to enter deep sleep by setting the `ZLO_SYSTEM_MAX_REPORT_INTERVAL` to zero, which will disable periodic reporting. Since periodic reporting is disabled, when in the unoccupied state the device does not need to keep track of time, meaning it can go into deep sleep and wait for the virtual sensor to trigger an occupancy event.

> (i) **Note:** When the Occupancy Sensor is in the occupied state, the device needs to start the 'occupied to unoccupied' timer, which means the device will sleep with RAM held and the 32kHz oscillator on.
>
> (i) **Note:** Since the Occupancy Sensor is a sleepy End Device, there may be a delay in sending out the attribute reports.

Enabling Sleep Prevention

The Occupancy Sensor can be kept awake in 'keep-alive' mode by pressing the **SW2** button on the Generic Expansion Board, which causes the LED3 on the board to start flashing 250ms on, 250ms off. The device is put back into normal operational mode by pressing the **SW3** button, which stops LED3 from flashing.

4.3.6 LED Indication of Different States

The following table lists the different behaviours of the LEDs on the Generic Expansion Board and the corresponding states of the node.

Light/LED Indication
LED1 (Sensor State): ON – Occupied, OFF - Unoccupied
LED1 Sensor Identifying ON, Sensor not Identifying OFF
LED3 flashes 1 second ON, 1 second OFF: Initiator mode active
LED3 flashes 1 second ON, 1 second OFF: Joining or re-joining the network
LED3 flashes 250ms ON, 250ms OFF: Keep-alive mode
During sleep, LED1 indicates the sensor state but LED2 and LED3 are OFF

4.3.7 Sensing Clusters

The occupancy sensing cluster contains the `u8Occupancy` attribute which is used to indicate the occupied/unoccupied status of the sensor. Binding to this cluster (Id **0x0406**) will enable a remote device to receive reports when the value of the `u8Occupancy` attribute on the sensor changes.

## 4.4 Using the LightTemperatureOccupancySensor Application

This section describes how to commission and operate the LightTemperatureOccupancySensor application in a ZigBee 3.0 network. To use this application, you must have programmed the relevant application binary into the JN5169/79 module on a Carrier Board fitted with the DR1175 Lighting/Sensor Expansion Board, as described in Section 4.1:

- **LightTemperatureOccupancySensor_NtagIcode_JN5169_DR1175.bin** for a JN5169 module

- **LightTemperatureOccupancySensor_NtagIcode_JN5179_DR1175.bin** for a JN5179 module

4.4.1 Light, Temperature & Occupancy Sensor Device Functionality

The functionality of the LightTemperatureOccupancySensor application is described and illustrated below.
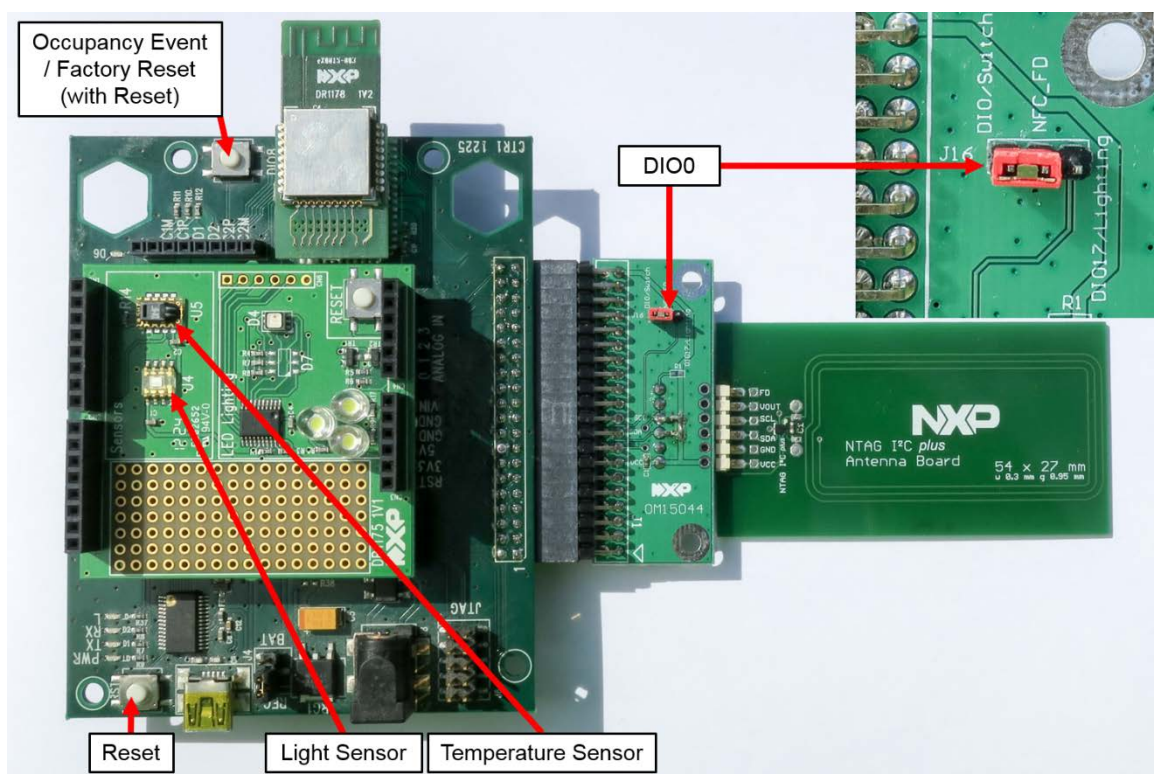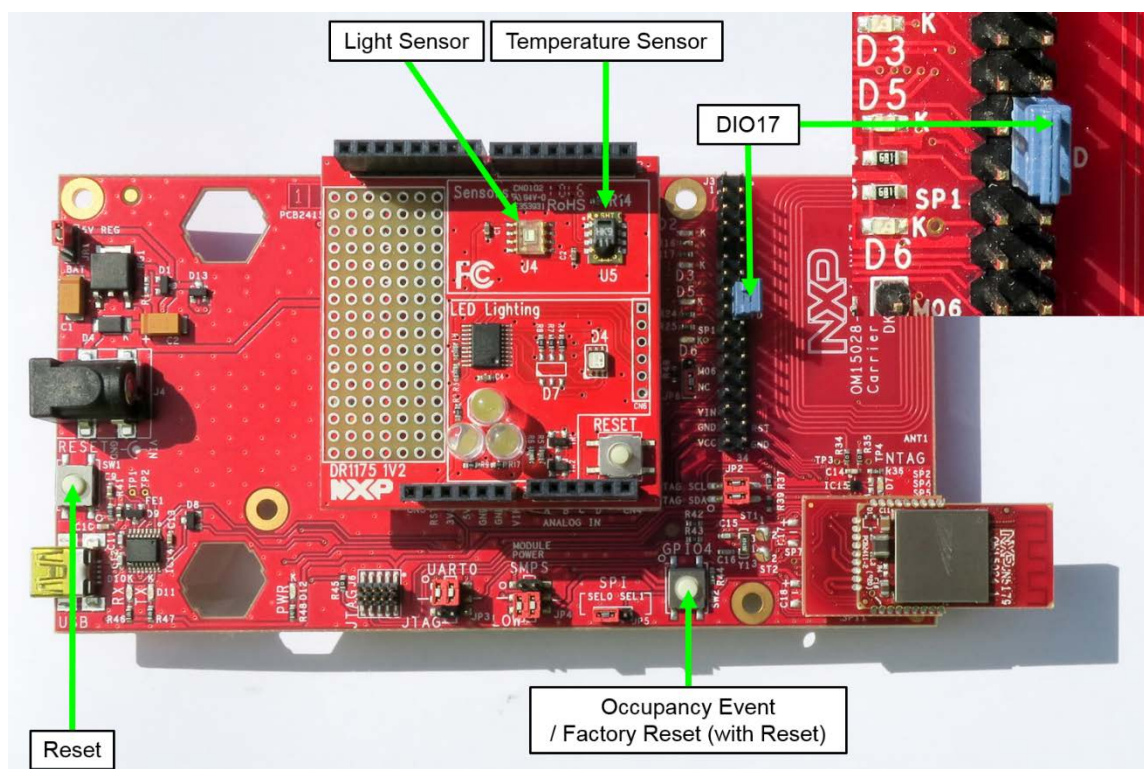
4.4.2 Clearing Context Data on the Device

When loading the application for the first time, any persistent context data must be cleared in the device.

The context data can be cleared by pressing and releasing the **RST/RESET** button while holding down the **DIO8/GPIO4** button (both buttons are on the Carrier Board).

A reset is indicated by the white LED cluster on the Lighting/Sensor Expansion Board illuminating for a second and switching off.

4.4.3 Commissioning

The commissioning operations for this device, such as forming/joining a network and binding, are the same as for the other device types described in this Application Note and are detailed in Section 4.5. To incorporate the device into a ZigBee 3.0 network, work through Section 4.5 and then return to this point for device operation.

4.4.4 Operation

There are two modes of operation of the Light, Temperature & Occupancy (LTO) Sensor, as follows:

- **Normal Mode:** The Light, Temperature & Occupancy Sensor sleeps and wakes up every second. If the light level / temperature level changes since the last reading by at least `LIGHT_SENSOR_MINIMUM_REPORTABLE_CHANGE` / `TEMPERATURE_SENSOR_MINIMUM_REPORTABLE_CHANGE` then the `u16MeasuredValue` attribute is updated and the Light, Temperature & Occupancy Sensor sends out an attribute report after 1 second. If there is no change, it will send a report every 61 seconds (`ZLO_MAX_REPORT_INTERVAL`).

- **Keep-alive Mode:** The Light, Temperature & Occupancy Sensor is permanently active and polls its parent every second. If the light level / temperature level changes since the last reading by at least `LIGHT_SENSOR_MINIMUM_REPORTABLE_CHANGE` / `TEMPERATURE_SENSOR_MINIMUM_REPORTABLE_CHANGE` then the `u16MeasuredValue` attribute is updated and the Light, Temperature & Occupancy Sensor sends out an attribute report after 1 second. If there is no change, it will send a report every 61 seconds (`ZLO_MAX_REPORT_INTERVAL`).

  The Light, Temperature & Occupancy Sensor will be reading the value of driver at the rate of `LIGHT_SENSOR_SAMPLING_TIME_IN_SECONDS` / `TEMPERATURE_SENSOR_SAMPLING_TIME_IN_SECONDS` (5 seconds) and updating its `u16MeasuredValue` attribute accordingly. Thus the changes in light sensor reading should be done only at intervals of 5 second.

  If there is any change in the `u16MeasuredValue` attribute, sensor will send out report after 1 second (`ZLO_MIN_REPORT_INTERVAL`), otherwise every 61 seconds (`ZLO_MAX_REPORT_INTERVAL`).

The LTO Sensor device can be put into 'keep-alive' mode by pressing the **RST/RESET** button on the Carrier Board 3 times. The button-presses must be done at one-second intervals, as failure to do this will not allow the right task to be triggered at the LTO Sensor.

To return to normal mode from 'keep-alive' mode or 'Finding and Binding' mode, simply reset the LTO Sensor.

4.4.4.1 Light Sensor

The maximum lux level as measured by ALS driver on the Lighting/Sensor Expansion Board is 4015 and the minimum lux level is 1.

> ⓘ **Note:** The light sensor needs a light source to measure the correct thresholds for proper operation (sensitivity increases based on the amount of light falling on sensor).

In order to address the above issue, you can simulate this light source by shining a torch over the light sensor and then placing your hand over the light sensor.

4.4.4.2 Occupancy Sensor

Occupancy events are simulated by pressing a button on the Carrier Board - button **DIO8/GPIO4** is assigned as the output of a virtual PIR detector. There are two types of virtual occupancy sensor: Open Collector and PWM. By default, the Light, Temperature & Occupancy Sensor operates as an Open Collector sensor.

Open Collector Sensor

An Open Collector sensor outputs a constant digital high/low signal for occupancy. In this demonstration, 'occupied' is represented by digital low and 'unoccupied' is represented by digital high.

To define a sensor as an Open Collector, uncomment the compile flag `PIR_TYPE_OPEN_COLLECTOR` in the makefile.

Simulating Unoccupied to Occupied Event:

To move the sensor from the unoccupied to occupied state, press and hold down the button **DIO8/GPIO4** on the Carrier Board.

Simulating Occupied to Unoccupied Event:

Once in the occupied state, to simulate an 'unoccupied' event, release the button **DIO8/GPIO4**. If no further occupancy event is simulated by pressing the **DIO8/GPIO4** button within a certain timeout period (180 seconds, by default), the sensor will automatically move to the unoccupied state.

PWM Sensor

A PWM sensor toggles its digital output between high and low while occupied.

Simulating Unoccupied to Occupied Event:

To move the sensor from the unoccupied to occupied state, press **DIO8/GPIO4** on the Carrier Board a certain number of times (5, by default) within a certain timeout period (10 seconds, by default).

Simulating Occupied to Unoccupied Event:

Once in the occupied state, if no further occupancy event is simulated by pressing the **DIO8/GPIO4** button within a certain timeout period (180 seconds, by default), the sensor will automatically move to the unoccupied state.

4.4.4.3 Temperature Sensor

The maximum temperature level as measured by HTS driver on the Lighting/Sensor Expansion Board is 125 and the minimum temperature level is 1.

4.4.5 LED Indication of Different States

The following table lists the different behaviours of LED D4 on the Lighting/Sensor Expansion Board and the corresponding states of the node.

Light/LED Indication
LED D4 flashes 1 second ON, 1 second OFF: Joining or re-joining the network
LED D4 **flashes 250ms ON, 250ms OFF:** Keep-alive mode
LED D4 OFF: Node is in the network or sleeping
LED D4 flashes 500ms ON, 500ms OFF: Initiator mode active

4.4.6 Sensing Clusters

The Light, Temperature & Occupancy Sensor uses the Illuminance Measurement, Temperature Measurement and Occupancy Sensing clusters to hold its results.

- The Illuminance Measurement cluster contains the `u16MeasuredValue` attribute which is used to indicate the light level measured by the sensor. Binding to this cluster (Id **0x0400**) will enable a remote device to receive reports when the value of this attribute on the sensor changes.

- The Temperature Measurement cluster contains the `u16MeasuredValue` attribute which is used to indicate the temperature measured by the sensor. Binding to this cluster (Id **0x0402**) will enable a remote device to receive reports when the value of this attribute on the sensor changes.

- The Occupancy Sensing cluster contains the `u8Occupancy` attribute which is used to indicate the occupied/unoccupied status of the sensor. Binding to this cluster (Id **0x0406**) will enable a remote device to receive reports when the value of the occupancy attribute on the sensor changes.
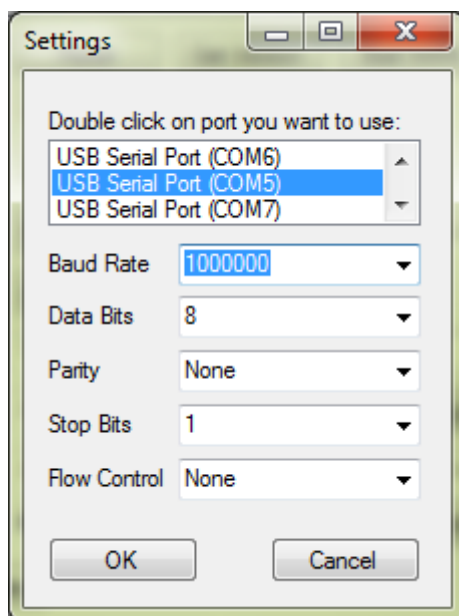
4.5 Network Commissioning Operations

This section describes the network commissioning operations for all the device types described in this Application Note. You should work through this section to incorporate a device in a network.

4.5.1 Forming and Joining a Network

To create a network with the Control Bridge (JN5169/79 USB Dongle) as the Coordinator and add one sensor node to it, follow the procedure below.

1. Plug a JN5169/79 USB Dongle into your PC and program the dongle with the Control Bridge binary **ZigBeeNodeControlBridge_JN51xx_FULL_FUNC_DEVICE.bin**, supplied in the Application Note *ZigBee Control Bridge [JN-AN-1216]*.

2. On the PC, start the ZigBee Gateway User Interface application, **ZGUI.exe** (supplied with the above Application Note).

3. In the interface, click **Settings** (top-left) and then, in the resulting dialogue box, select the serial port to which the Control Bridge device is connected and set the **Baud Rate** field to 1000000.

4. Click **OK** to apply the serial port configuration.

5.   In the interface, fill in the **Set CMSK** (channel mask) and **Permit Join** fields, and ensure the other fields are as shown below.

6.   Connect to the Control Bridge device by clicking **Open Port** (top-left).

7.   Reset the Control Bridge device by clicking the **Reset** button on the **Management** tab.

8.   Set the Control Bridge device type to COORDINATOR in the **Set Type** field on the **Management** tab.

9. Start the network by clicking the **Start NWK** button on the **Management** tab. This should start the network and information should be output via the **Raw Data** and **Received Message View** panes, as shown below.
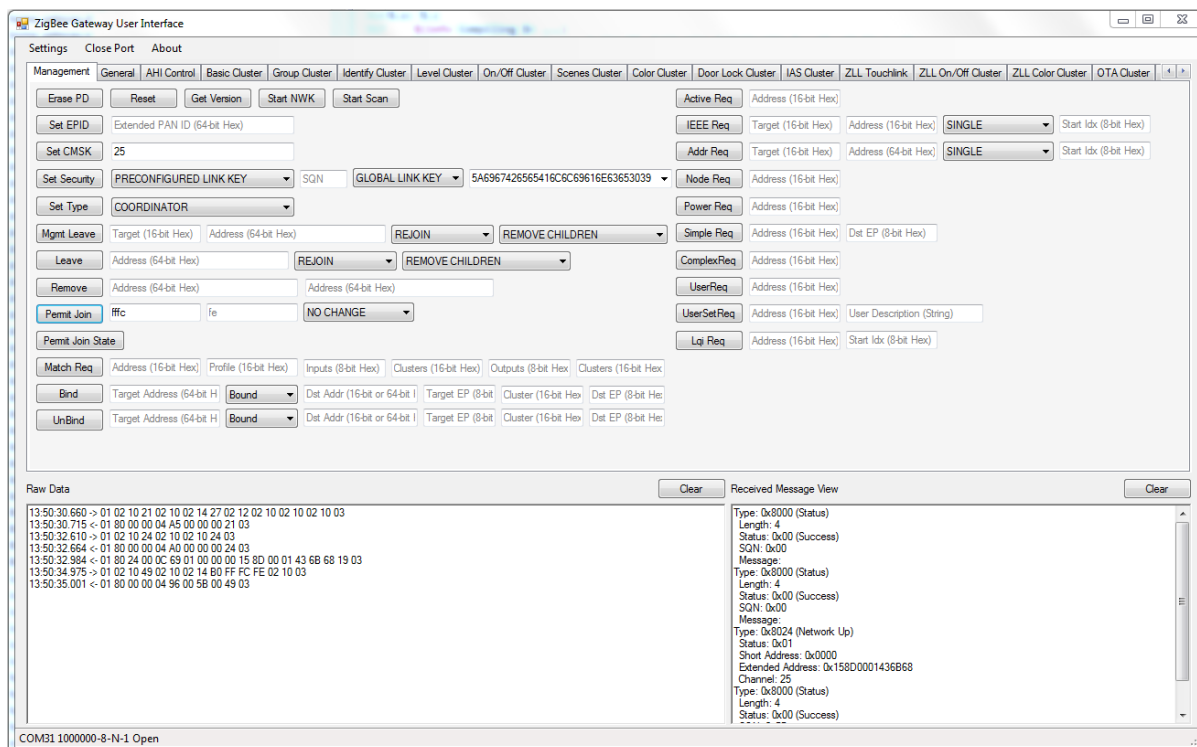
10. Open the network for devices to join by clicking the **Permit Join** button and start a sensor device. When the device has joined the network, an End Device Announce message will be displayed in the **Received Message View** pane, as shown below.
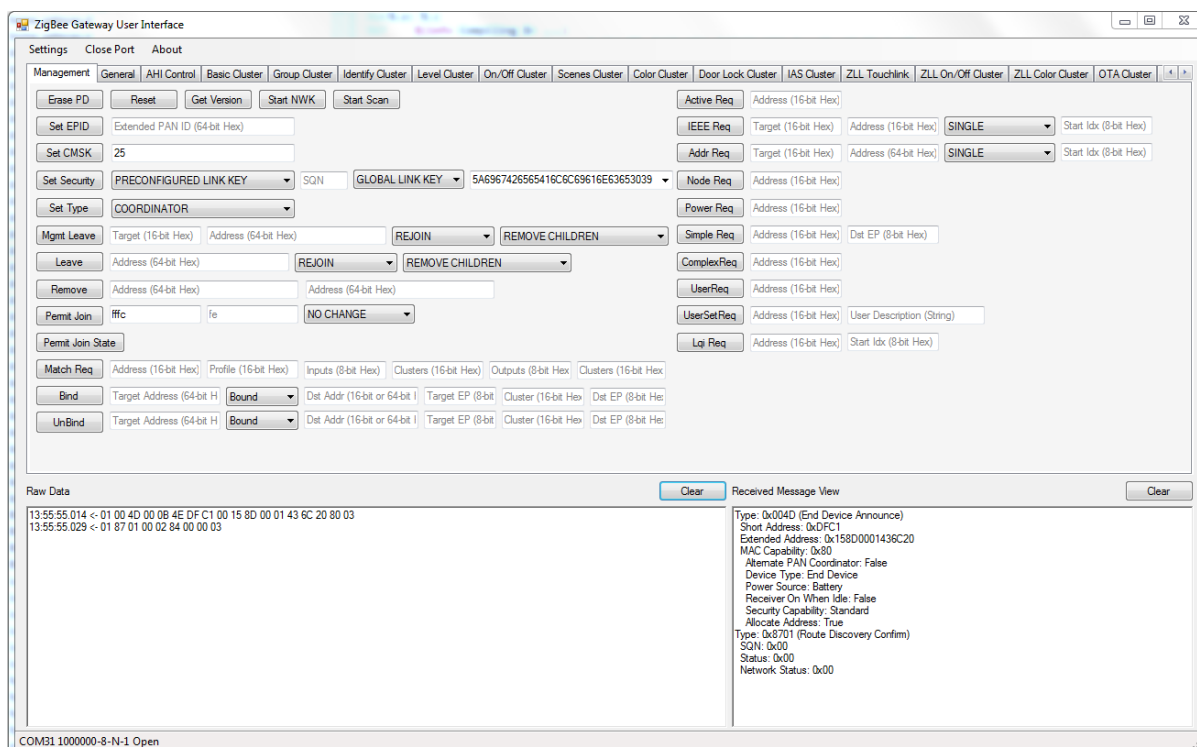
> **Note 1:** If a sensor node does not find a suitable network to join, it goes into a soft sleep after scanning all the primary and secondary channels.

> **Note 2:** If a sensor node loses its parents, it goes into a soft sleep after trying to rejoin on all the channels.

4.5.2 Joining an Existing Network using NFC

A sensor 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).

4.5.3 Allowing Other Nodes to Join

If you wish to add other nodes to the network, open the network for devices to join by clicking the **Permit Join** button in the interface again (see Step 10 above).

4.5.4 Binding Nodes (Control Bridge)

To create a binding between a cluster on a sensor node and the Control Bridge, follow the procedure below.

1.   In the interface, fill in the **Bind** fields on the **Management** tab as shown below.

2.   Place the sensor node in the persistent poll/keep-alive mode to ensure that it is able to receive a Bind Request from the Control Bridge.

3. Initiate binding by clicking the **Bind** button. If successful, the result will be shown in the **Received Message View** pane, as shown below.

4. Once binding to the desired cluster is complete, the sensor node can be taken out of the persistent poll/keep-alive mode. At this point, any reports generated by a sensor node for bound clusters should be displayed in the **Received Message View** pane. An example of this is shown below for an Occupancy Sensor reporting a change in the state of its `u8Occupancy` attribute.
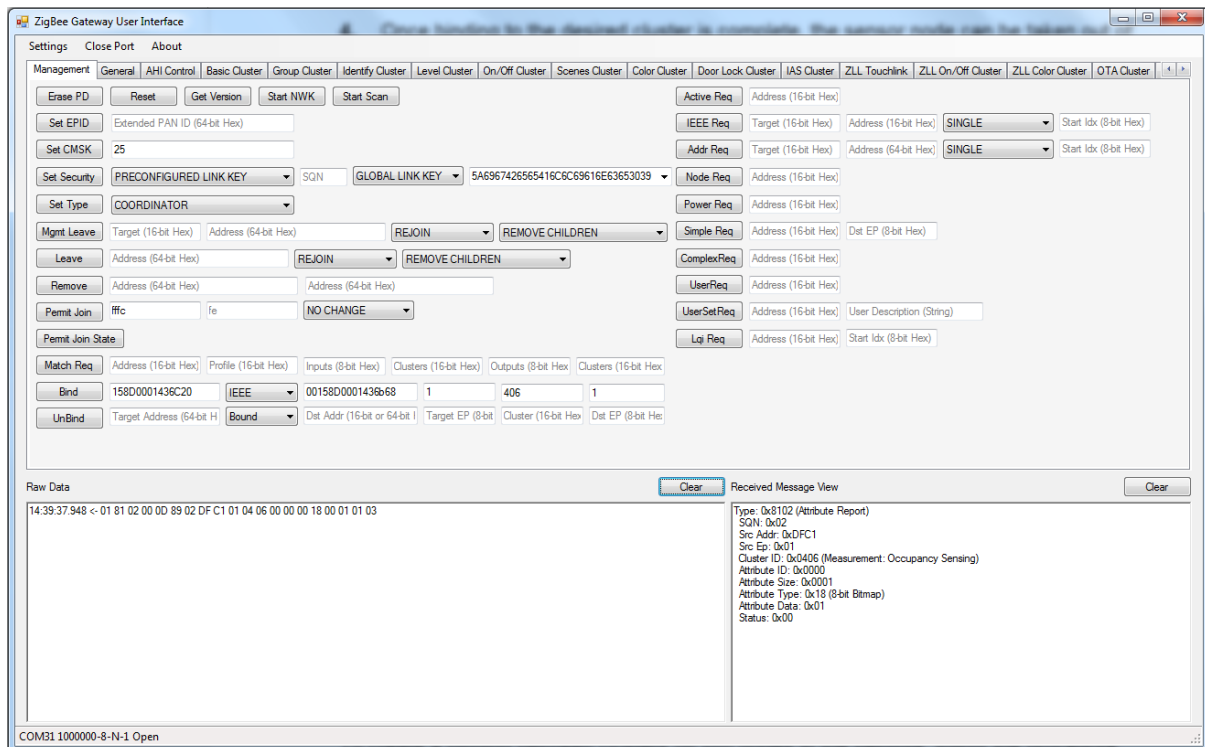
4.5.5 Binding Nodes (Finding and Binding)

To create a binding between clusters on two nodes in the network, follow the procedure below.

1. Put the target node (to which you wish to bind the sensor node) into identify mode.

2. Put the sensor node into 'Finding and Binding' mode as follows:

    - If a Light Sensor, hold down the button **DIO8/GPIO4** on the Carrier Board until the target device stops identifying (the sensor node will indicate that it is in 'Finding and Binding' mode by flashing an LED).

    - If an Occupancy Sensor, hold down the button **SW4** on the Generic Expansion Board until the target device stops identifying (the sensor node will indicate that it is in 'Finding and Binding' mode by flashing an LED).

    - If a Light, Temperature & Occupancy Sensor, press the **RST/RESET** button on the Carrier Board 5 times (the sensor node will indicate that it is in 'Finding and Binding' mode by flashing a red LED). The button-presses must be done at one-second intervals, as failure to do this will not allow the right task to be triggered at the sensor. Once the target device stops identifying, press the **RST/RESET** button on the sensor node again to come out of Finding and Binding mode.

4.5.6 Performing a Factory Reset

To return the sensor node to its factory-new state, hold down the button **DIO8/GPIO4** and then press/release the **RST/RESET** button, both of which are on the underlying Carrier Board.

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 7.

5.1 Application Start-up

The diagram below illustrates the typical start-up flow of an NXP ZigBee 3.0 device.

Cold Start

vAppMain()

Stabilise clock
Debug initialisation
Watchdog Event Trap

APP_vInitResources()

Initialise Timers
Create Timers
Create Queues

APP_vSetUpHardware()

Set up interrupts

vInitialiseApp()

Power Manager initialisation
Persistent Data Manager initialisation
Protocol Data Unit Manager initialisation
General hardware initialisation

APP_vInitialiseNode()

LED initialisation
Button initialisation
Clear context (optional)
Load context
Load the pre-configured link key (no context)
Initialise the Application Framework
Start the ZigBee stack (context restore)
Initialise the ZCL
BDB initialisation

APP_vMainLoop()

Idle Loop:
Run ZPS task
Run BDB task
Run Ztimer task
Run Application task
Reset the Watchdog timer
Check if sleep is possible
Service the Power Manager

Sleep?

No → Doze → ISR/Task activates & completes

Yes

Save MAC settings
Disable hardware peripherals
Clear DIO interrupt flags
Stop Ztimer operation

Sleep

PWRM_CALLBACK(PreSleep)

Warm Start

PWRM_CALLBACK(Wakeup)

Debug initialisation

No ← RAM held? → Yes

Restore MAC settings
General hardware re-initialisation
Set up interrupts
Resume Ztimer operation

DIO / Wake Timer ISR

vISR_SystemController()

Clear down interrupt flag and execute
the user-defined callback function

vWakeCallBack()

Restart timers

Return to the idle loop

File Key:

app_start.c
app_main.c
app_xxxx_node.c
app_xxxx_buttons.c
app_sleep_handler.c

5.2 Code Common to All Sensors

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.

5.3 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.

5.4 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.

5.5 Light Sensor Application Code

This section describes the application code for the LightSensor, 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 5.8.

5.5.1 Operational State Machine

The operational state machine (**sDeviceDesc.eNodeState**) is located within **app_light_sensor_state_machine.c**. If further operational modes are required, additional states must be added to this switch statement.

5.5.2 Button Press and Release Handling

Buttons are debounced in **APP_cbTimerButtonScan()** contained in **app_light_sensor_buttons.c**. Any button events are then passed into the **PP_msgEvents** queue for processing in **APP_ZLO_vSensor_Task()** in **app_zlo_sensor_node.c**, where the button event is eventually handled in **vAppHandleAppEvent()** contained in the file **app_event_handler.c**

5.5.3 Sleeping

The application makes a call to **vAttemptToSleep()** contained in the file **app_sleep_handler.c** every time it passes around the main processing loop. This function checks the state of a number of application timers. If all timers considered to be "non-sleep preventing" are stopped, the function will stop any remaining timers and decide whether the device should sleep with memory held (with a wake timer to wake up after a pre-determined sleep period) or enter deep sleep mode (which requires external influence in the form of a DIO state change or a device reset to wake from).

5.5.4 Configuration Macros

The following macros, which can be found in **App_LightSensor.h**, are available for configuring the Light Sensor:

```
#define LIGHT_SENSOR_MINIMUM_MEASURED_VALUE                     0x0001
#define LIGHT_SENSOR_MAXIMUM_MEASURED_VALUE                     0xFAF
#define LIGHT_SENSOR_MINIMUM_REPORTABLE_CHANGE                  0x01
#define LIGHT_SENSOR_SAMPLING_TIME_IN_SECONDS                   5
```

The sampling time is the rate at which the `u16MeasureValue` attribute will be changed.

5.6 Occupancy Sensor Application Code

This section describes the application code for the OccupancySensor, 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 5.8.

5.6.1 Operational State Machine

The operational state machine (**sDeviceDesc.eNodeState**) is located within **app_occupancy_sensor_state_machine.c**. If further operational modes are required, additional states must be added to this switch statement.

5.6.2 Button Press and Release Handling

Buttons and occupancy sensor inputs are debounced in **APP_cbTimerButtonScan()** contained in **app_occupancy_buttons.c**. Any button events are then passed into the **APP_msgEvents** queue for processing in **APP_ZLO_vSensor_Task()** in **app_zlo_sensor_node.c**, where the button event is eventually handled in **vAppHandleAppEvent()** contained in the file **app_event_handler.c**.

5.6.3 Sleeping

The application makes a call to **vAttemptToSleep()**, contained in the file **app_sleep_handler.c**, every time it passes around the main processing loop. This function checks the state of a number of application timers. If all timers considered to be "non-sleep preventing" are stopped, the function will stop any remaining timers and decide whether the device should sleep with memory held (with a wake timer to wake up after a pre-determined sleep period) or enter deep sleep mode (which requires external influence in the form of a DIO state change or a device reset to wake from).

5.6.4 Configuration Macros

The following macros, which can be found in **App_OccupancySensor.h**, are available for configuring the sensor:

```
#define APP_OCCUPANCY_SENSOR_UNOCCUPIED_TO_OCCUPIED_DELAY        10
#define APP_OCCUPANCY_SENSOR_TRIGGER_THRESHOLD                   5
#define APP_OCCUPANCY_SENSOR_OCCUPIED_TO_UNOCCUPIED_DELAY        180
```

5.7 Light, Temperature and Occupancy Sensor Application Code

This section describes the application code for LightTemperatureOccupancySensor, 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 5.8.

5.7.1 Operational State Machine

The operational state machine (**sDeviceDesc.eNodeState**) is located within **app_sensor_state_machine.c**. If further operational modes are required, additional states must be added to this switch statement.

5.7.2 Button Press and Release Handling

Buttons and occupancy sensor inputs are debounced in **APP_cbTimerButtonScan()** contained in **app_sensor_buttons.c**. Any button events are then passed into the **APP_msgEvents** queue for processing in **APP_ZLO_vSensor_Task()** in **app_zlo_sensor_node.c**, where the button event is eventually handled in **vAppHandleAppEvent()** contained in the file **app_event_handler.c**

5.7.3 Sleeping

The application makes a call to **vAttemptToSleep()**, contained in the file **app_sleep_handler.c**, every time it passes around the main processing loop. This function checks the state of a number of application timers. If all timers considered to be "non-sleep preventing" are stopped, the function will stop any remaining timers and decide whether the device should sleep with memory held (with a wake timer to wake up after a pre-determined sleep period) or enter deep sleep mode (which requires external influence in the form of a DIO state change or a device reset to wake from).

5.7.4 Configuration Macros

The following macros, which can be found in **App_LightTemperatureOccupancySensor.h**, are available for configuring the sensor:

```
#define LIGHT_SENSOR_MINIMUM_MEASURED_VALUE                  0x0001
#define LIGHT_SENSOR_MAXIMUM_MEASURED_VALUE                  0xFAF
#define LIGHT_SENSOR_MINIMUM_REPORTABLE_CHANGE              0x01
#define LIGHT_SENSOR_SAMPLING_TIME_IN_SECONDS              5


#define TEMPERATURE_SENSOR_MINIMUM_MEASURED_VALUE          0x0001
#define TEMPERATURE_SENSOR_MAXIMUM_MEASURED_VALUE          0xFAF
#define TEMPERATURE_SENSOR_MINIMUM_REPORTABLE_CHANGE       0x01
#define TEMPERATURE_SENSOR_SAMPLING_TIME_IN_SECONDS        5
```

The sampling time is the rate at which the `u16MeasureValue` attribute will be changed.

5.8 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.

5.8.1 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-1219] 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-1220-Zigbee-3-0-Sensors

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.

5.8.2 Build Instructions

The software provided with 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 5.8.2.1.
- To build using the IDE, refer to Section 5.8.2.2.

5.8.2.1 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

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.

5.8.2.2 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 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.

5.9 Debugging the Demonstration Application

5.9.1 Serial Debug

Each node in the demonstration prints out debug information via the UART port based on the debug flags set in the makefile. This debug information can be viewed using terminal emulator software, e.g. Tera Term. Connect the node of interest to a PC using the Mini-USB cable (supplied in the evaluation kit) and configure the terminal emulator's COM port as follows:

BAUD Rate	115200
Data	8 bits
Parity	None
Stop bit	1 bit
Flow Control	None

Debug can be disabled for production by setting the 'Trace' flag in the relevant node's makefile to zero. The makefile also defines a subset of debug flags that allows localised debug statements to be collectively enabled or disabled, e.g. TRACE_START.

By default, there are certain debug print lines left in the Application Note code to trace any issues.

5.9.2 JTAG Debug

The application on a node can be debugged from BeyondStudio for NXP via a JTAG connection. This method requires additional hardware to form the JTAG interface on the node, including a JTAG expansion board and JTAG adaptor/dongle. JTAG debugging is fully described in the Application Note *JN516x JTAG Debugging in BeyondStudio (JN-AN-1203)*.

6 Advanced User Information

6.1 Saving Network Context

All device types are protected from losing their network configuration during a power outage by means of context saving. The required network parameters are automatically preserved in non-volatile memory by the ZigBee PRO Stack (ZPS). On restart, the radio channel, Extended PAN ID (EPID) and security keys are restored.

Application-specific information can also be preserved in the non-volatile memory, which is most commonly used to preserve the application's operating state.

6.2 Security Key

Security policy and default security keys are defined in the ZigBee Base Device Behaviour (BDB) Specification. Pre-configured link keys are provided in the ZigBee Base Device file **bdb_link_keys.c**, included in the JN516x/7x ZigBee 3.0 SDK.

7 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 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

    JN-AN-1220 (v1003) 27-Mar-2017