Application Note: JN-AN-1246

DK006: JN5189 / JN5188 / K32W061 / K32W041

Zigbee 3.0 Sensors

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

This Application Note provides example applications for sensors in a Zigbee 3.0 network that employs the NXP DK006 Zigbee 3.0 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 DK006 Development 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 DK006 platform.

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-3131]* and the clusters used by the devices are detailed in the *Zigbee Cluster Library User Guide [JN-UG-3132]*. 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-3130]* for a general introduction.

2 Development Environment

2.1 Software

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

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

The MCUXpresso software and installation instructions are described in the *Zigbee 3.0 Getting Started Application Note [JN-AN-1260]*. The *JN-AN-1246 Release Notes* (included in this folder) indicate the versions of MCUXpresso and SDK that you will need to work with this Application Note.

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

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

2.2 Hardware

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

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

This kit supports the NFC commissioning of network nodes (see NFC Hardware Support for hardware configuration details).

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
LightSensor	Light Sensor	Control Bridge
OccupancySensor	Occupancy Sensor	Control Bridge
LightTemperatureOccupancySensor	Light, Temperature & Occupancy Sensor	Control Bridge

Table 1: Example Applications and Device Types

The Control Bridge is described in in the Application Note *Zigbee 3.0 Control Bridge [JN-AN-1247]*.

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

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

3.1 NFC Hardware Support

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

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

4 Running the Demonstration Application

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

4.1 Loading the Applications

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

Note that the light sensor used on the OM15081 Lighting/Sensor Expansion Board changed between revision 1 and 2 the supplied binaries currently are available for both. Not all variants are provided with the Application Note but they can be built, if required, by setting the DR_REV variable on the command line to make in the MCUXpresso project settings. The appropriate board revision is indicated in the binary filename.

DK006 Hardware	JN5189 Binary File
OM15076-3 Carrier Board OM15081-2 Lighting/Sensor Expansion Board	**LightSensor_OM15081R2** **LightSensor_NtagIcode_Ota_OM15081R2_V1** **LTOSensor_OM15081R2** **LTOSensor_NtagIcode_Ota_OM15081R2_V1**
OM15076-3 Carrier Board OM15082-2 Generic Expansion Board	**OccupancySensor_OM15082** **OccupancySensor_NtagIcode_Ota_OM15082_V1**

Table 2: Application Binaries and Hardware Components

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

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

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

1. Connect a USB port of your PC to the USB Mini B port marked “FTDI USB” on the Carrier Board (next to the 34-pin header) using a ‘USB A to Mini B’ cable. At this point, you may be prompted to install the driver for the USB virtual COM port.

2. Determine which serial communications port on your PC has been allocated to the USB connection.

3. Put the DK006 device into programming mode by holding down the ISP button while pressing and releasing the RESET button.

4. Run the batch (**.bat**) file provided alongside the binary (**.bin**) file to erase the contents of the flash memory including the persistent data stored by the PDM and program the binary file into flash memory. The batch file will prompt for the COM port number to use.

5. Once the download has successfully completed, reset the board or module to run the application.

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

4.2 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 DK006 module on a Carrier Board fitted with the OM15081 Lighting/Sensor Expansion Board, as described in Loading the Applications.

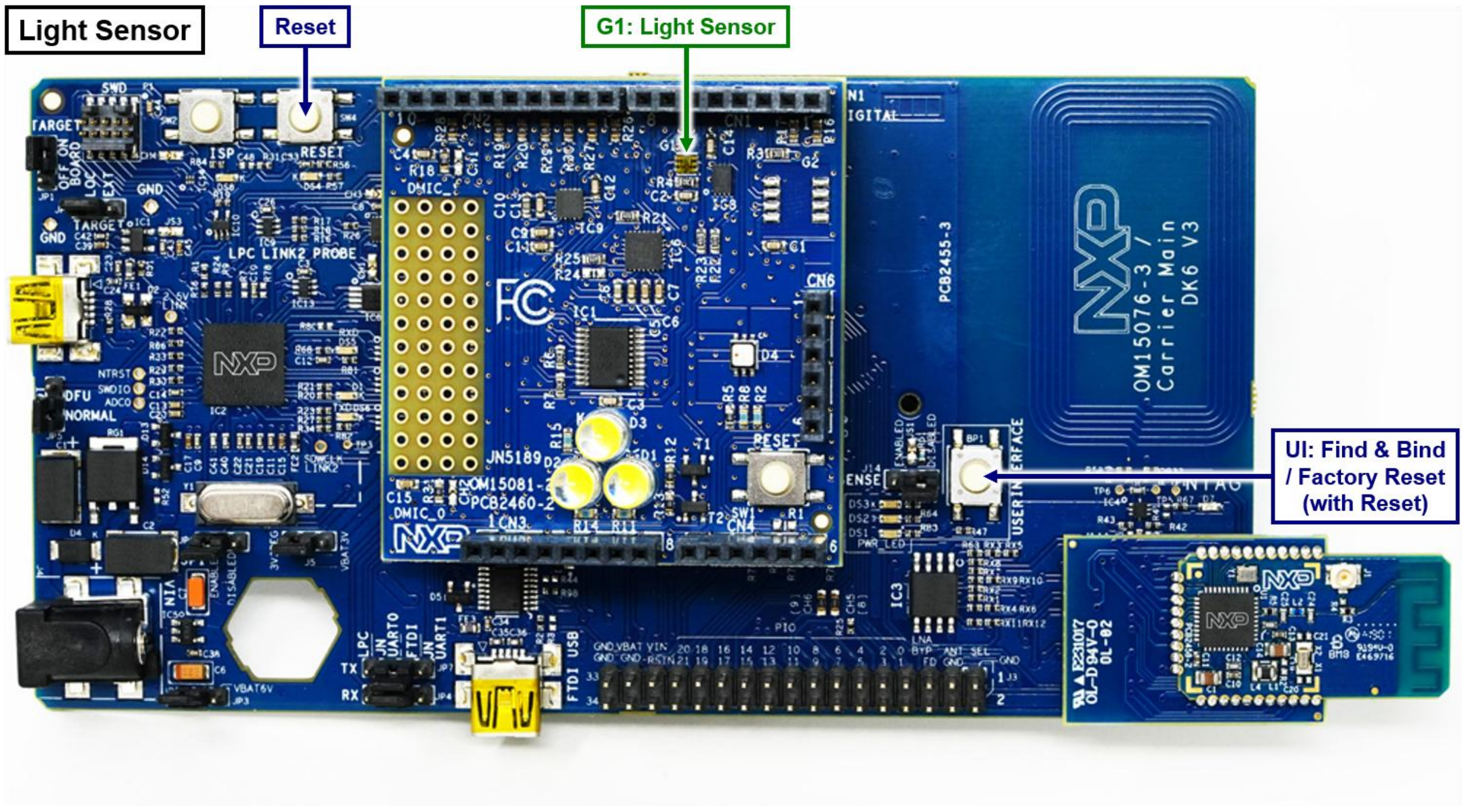
- **LightSensor_OM15081R2.bin**
- **LightSensor_NtagIcode_Ota_OM15081R2_V1.bin**

Note that the pre-built binaries are for revision 2 of the OM15081 Lighting/Sensor Expansion Board as indicated in the filename. To build for older revision 1 boards set DR_REV=1 on the command line in the MCUXpresso project.

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.



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 **RESET** button while holding down the **USER INTERFACE** button (both buttons are on the Carrier Board).

Alternatively erasing the whole flash memory when programming will clear the context data.

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 Network Commissioning Operations. To incorporate the device into a Zigbee 3.0 network, work through Network Commissioning Operations 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 to any bound devices. 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 to any bound devices 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 are only detected 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. Return the device to normal sleeping mode using a single reset.

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 3.0 Cluster Library User Guide [JN-UG-3132]*.

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

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 DS2 on the Carrier 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.

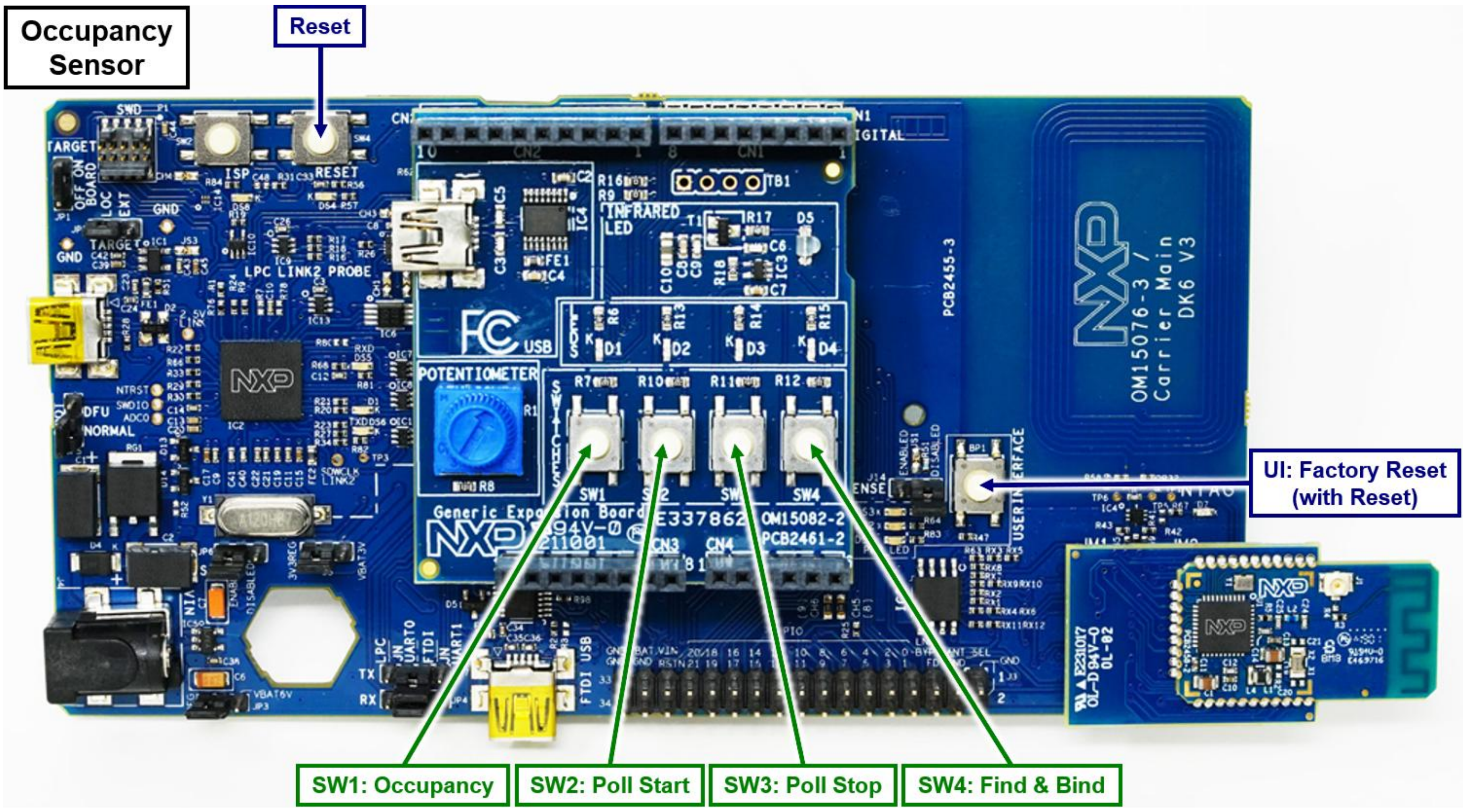
4.3 Using the OccupancySensor Application

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 DK006 module on a Carrier Board fitted with the DR1199 Generic Expansion Board, as described in Loading the Applications:

- **OccupancySensor_OM15082.bin**
- **OccupancySensor_NtagIcode_Ota_OM15082_V1.bin**

4.3.1 Occupancy Sensor Device Functionality

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



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 **RESET** button while holding down the **USER INTERFACE** button (both buttons are on the Carrier Board).

Alternatively erasing the whole flash memory when programming will clear the context data.

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 Network Commissioning Operations. To incorporate the device into a Zigbee 3.0 network, work through Network Commissioning Operations 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 JN518x module is awake for sampling or re-joining.

4.3.4.1 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 (this is the default).

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.

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.

4.3.4.2 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).

Note 1: The number of button-presses required can be customised using the `APP_OCCUPANCY_SENSOR_UNOCCUPIED_TO_OCCUPIED_THRESHOLD` macro.

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.

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`.

4.3.4.3 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’

4.3.4.4 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 bound devices 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 0 seconds, disabling this feature.

4.3.5.1 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 – this is the default setting. 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.

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.

Note: Since the Occupancy Sensor is a sleepy End Device, there may be a delay in sending out the attribute reports.

4.3.5.2 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
LED2 flashes 1 second ON, 1 second OFF: Identifying
LED3 flashes 1 second ON, 1 second OFF: Joining or re-joining the network
LED3 flashes 0.5 second ON, 0.5 second OFF: Initiator mode active
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

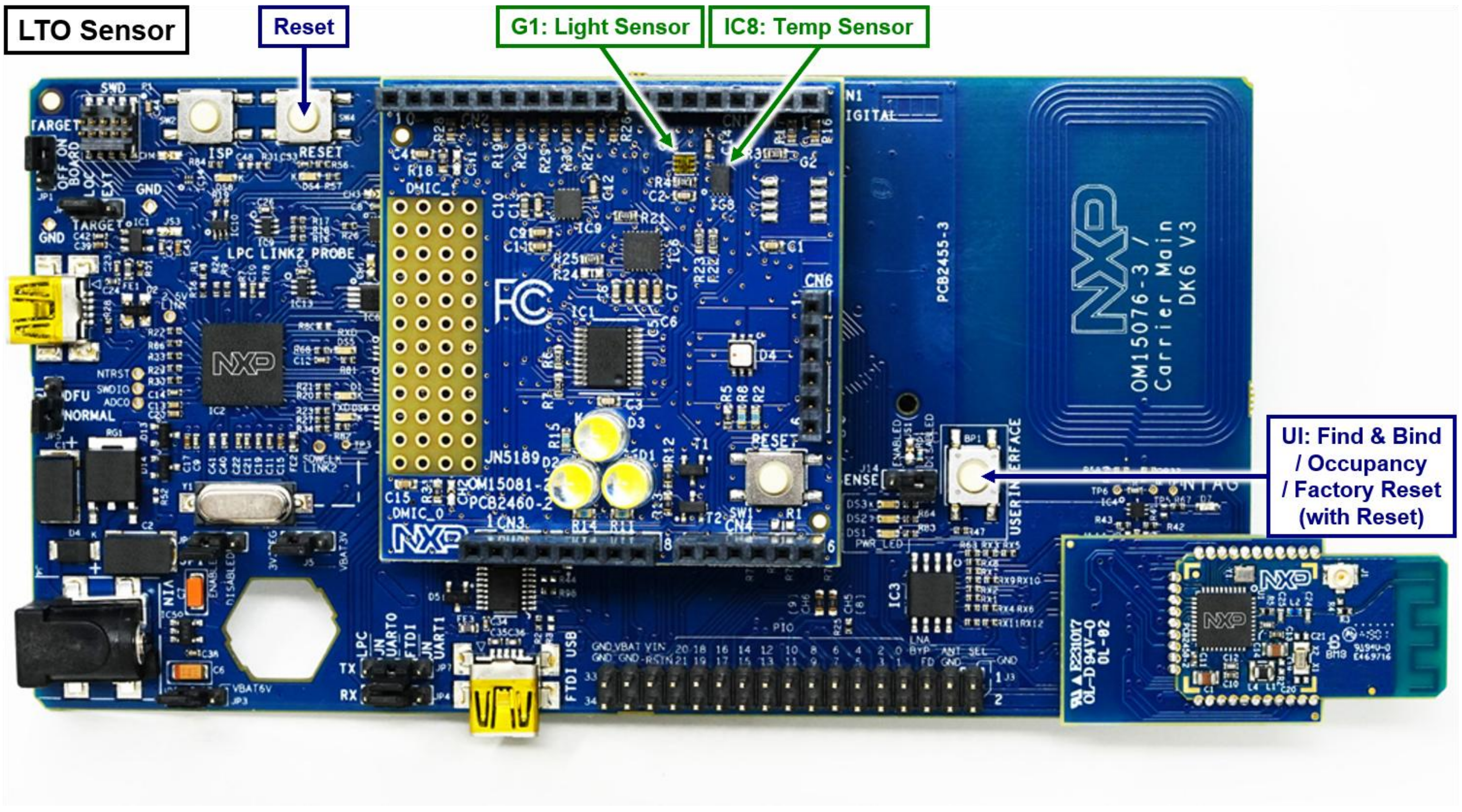
This section describes how to commission and operate the LightTemperatureOccupancySensor application in a Zigbee 3.0 network. To use this application, you must have programmed the relevant application binary into the DK006 module on a Carrier Board fitted with the OM15081 Lighting/Sensor Expansion Board, as described in Loading the Applications:

- **LTOSensor_OM15081R2.bin**
- **LTOSensor_NtagIcode_Ota_OM15081R2_V1.bin**

Note that the pre-built binaries are for revision 2 of the OM15081 Lighting/Sensor Expansion Board as indicated in the filename. To build for older revision 1 boards set DR_REV=1 on the command line in the MCUXpresso project.

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 **RESET** button while holding down the **USER INTERFACE** button (both buttons are on the Carrier Board).

Alternatively erasing the whole flash memory when programming will clear the context data.

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 Network Commissioning Operations . To incorporate the device into a Zigbee 3.0 network, work through Network Commissioning Operations 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`).

  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 **USER INTERFACE** 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 (this is the default).

Simulating Unoccupied to Occupied Event:

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

Simulating Occupied to Unoccupied Event:

Once in the occupied state, to simulate an 'unoccupied' event, release the button **USER INTERFACE**. If no further occupancy event is simulated by pressing the **USER INTERFACE** 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 **USER INTERFACE** 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 **USER INTERFACE** 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 the SE98A driver on the Lighting/Sensor Expansion Board is 125C and the minimum temperature level is -40C.

4.4.5 LED Indication of Different States

The following table lists the different behaviours of LED Ds2 on the Carrier 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.

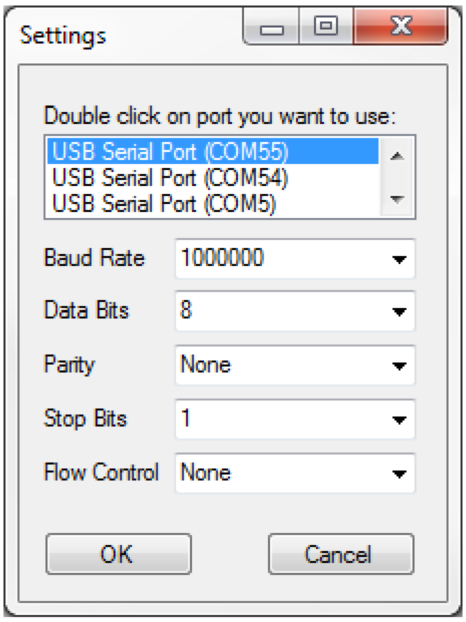
4.5.1 Forming and Joining a Network

To create a network with the Control Bridge as the Coordinator and add one sensor node to it, follow the procedure below.

1. Plug a USB Dongle or Carrier Board into your PC and program the dongle with the Control Bridge binary (such as **ControlBridge_Full_GpProxy_1000000.bin**), supplied in the Application Note *Zigbee 3.0 Control Bridge [JN-AN-1247]*.

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.


Settings

Double click on port you want to use:

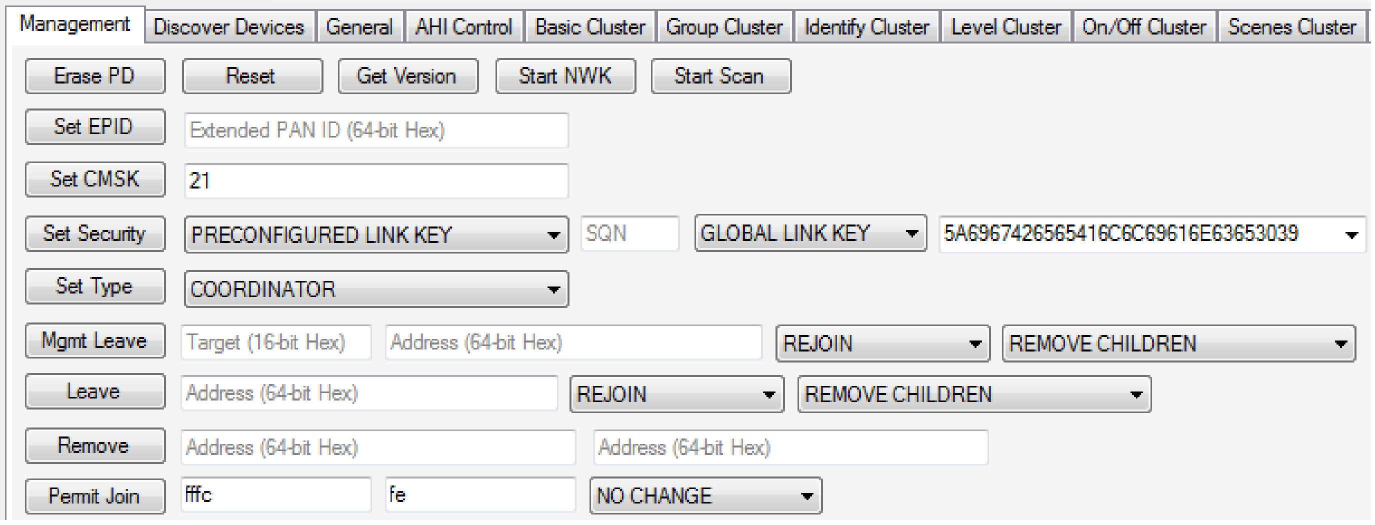
USB Serial Port (COM55)
USB Serial Port (COM54)
USB Serial Port (COM5)

Baud Rate	1000000
Data Bits	8
Parity	None
Stop Bits	1
Flow Control	None

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

5. Connect to the Control Bridge device by clicking **Open Port** (top-left on the menu bar).

6. Ensure the Management tab is selected as the next steps will use these controls.



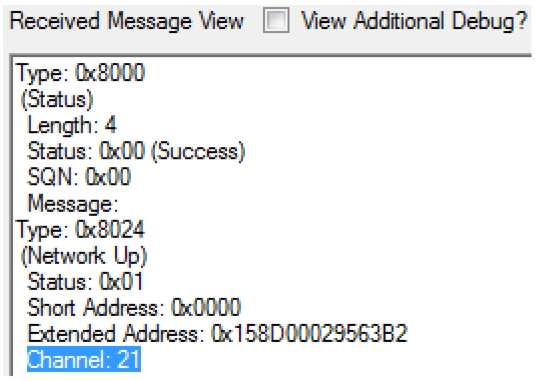
7. If the Control Bridge has been used previously it may be helpful to erase the persistent data by clicking the **Erase PD** button. This will close down the network and remove knowledge of any devices that were in the previous network formed by the Control Bridge. Data will scroll in the **Raw Data** and **Received Message View** multi-line edit boxes at the bottom of the window until the process is complete, all the commands result in updates to these edit boxes.

8. Reset the Control Bridge device by clicking the **Reset** button on the **Management** tab. Data will scroll in the multi-line edit boxes at the bottom of the window until the process is complete.

9. The Control Bridge can be limited to operate on only a single channel, this is useful when working with a sniffer utility. To do this enter the channel in the edit box next to the **Set CMSK** button, then click the button to apply the setting.

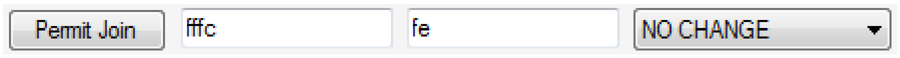
10. Set the Control Bridge device type to COORDINATOR in the dropdown next to the **Set Type** button, then click the button to apply the setting.

11. Clear the multi-line editboxes using the Clear buttons (so the network formation can be easily observed. 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 including confirmation of the selected channel:

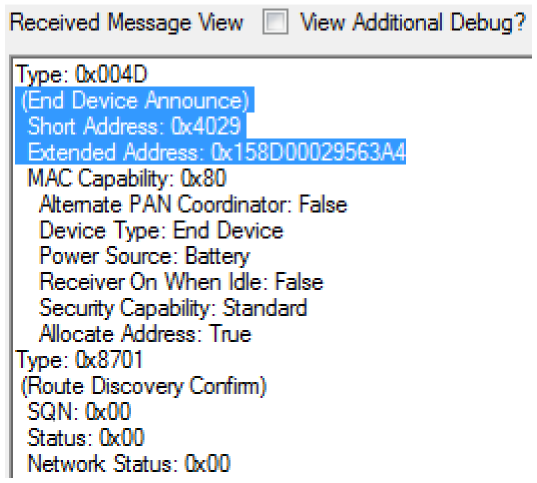


```
Type: 0x8000
 (Status)
  Length: 4
  Status: 0x00 (Success)
  SQN: 0x00
  Message:
Type: 0x8024
 (Network Up)
  Status: 0x01
  Short Address: 0x0000
  Extended Address: 0x158D00029563B2
  Channel: 21
```

12. Open the network for devices to join by completing the controls next to the **Permit Join** button as shown below. This opens the network for all device types to join for a period of time.



1. Then power on 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. This includes both the network allocated short address and fixed extended address.



```
Type: 0x004D
 (End Device Announce)
  Short Address: 0x4029
  Extended Address: 0x158D00029563A4
  MAC Capability: 0x80
    Alternate PAN Coordinator: False
    Device Type: End Device
    Power Source: Battery
    Receiver On When Idle: False
    Security Capability: Standard
    Allocate Address: True
Type: 0x8701
 (Route Discovery Confirm)
  SQN: 0x00
  Status: 0x00
  Network Status: 0x00
```

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

- The *NcIcode* build of the Zigbee IoT Control Bridge, described in the Application Note *Zigbee 3.0 IoT Control Bridge (JN-AN-1247)*
- The Zigbee IoT Gateway Host, described in the Application Note *Zigbee IoT Gateway Host with NFC (JN-AN-1222)*

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

Ensure the hardware is set up for NFC as described in NFC Hardware Support.

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

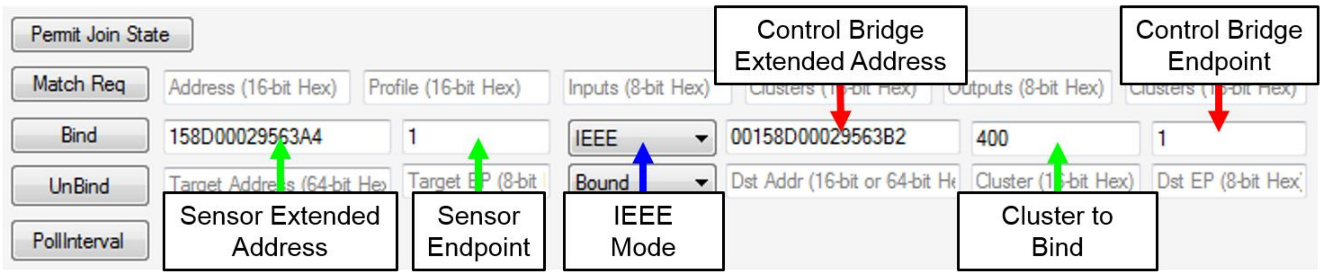
4.5.3 Allowing Other Nodes to Join

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 steps 12 and 13 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. Use Cluster ID 400 for Illuminance Measurement, 402 for Temperature Measurement, 406 for Occupancy Sensing.

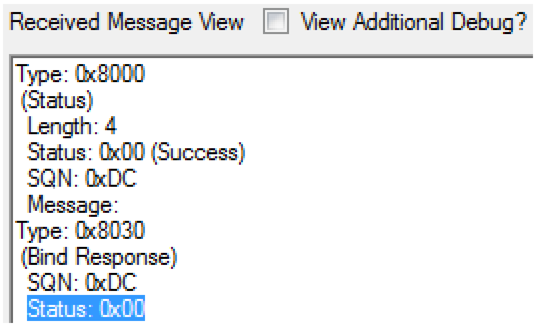


158D00029563A4 | 1 | IEEE | 00158D00029563B2 | 400 | 1

Bound

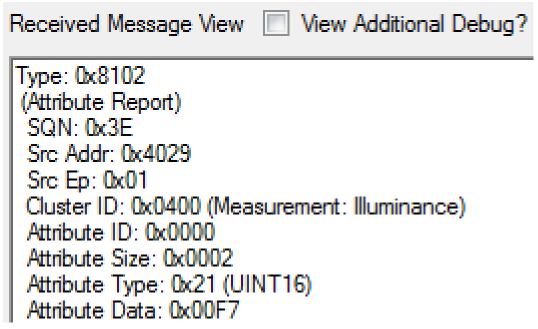
2. Place the sensor node in the persistent poll/keep-alive mode to ensure that it is able to receive a Bind Request from the Control Bridge.

3. Initiate binding by clicking the **Bind** button. If successful, the result will be shown in the **Received Message View** pane success, is indicated with a Status of 0x00:



```
Type: 0x8000
(Status)
 Length: 4
 Status: 0x00 (Success)
 SQN: 0xDC
 Message:
Type: 0x8030
(Bind Response)
 SQN: 0xDC
 Status: 0x00
```

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 a Light Sensor reporting a change in the state of its u16MeasuredValue attribute.



```
Type: 0x8102
(Attribute Report)
 SQN: 0x3E
 Src Addr: 0x4029
 Src Ep: 0x01
 Cluster ID: 0x0400 (Measurement: Illuminance)
 Attribute ID: 0x0000
 Attribute Size: 0x0002
 Attribute Type: 0x21 (UINT16)
 Attribute Data: 0x00F7
```

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 **USER INTERFACE** 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 **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 **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 **USER INTERFACE** and then press/release the **RESET** button, both of which are on the underlying Carrier Board.

5 Over-The-Air (OTA) Upgrade

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

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

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

5.1 Overview

Support for the OTA Upgrade cluster as a client has been included for the Light Sensor, Occupancy Sensor and LightTemperatureOccupancy Sensor 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 DK006 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 initial client binaries to be programmed into the JN5189 device are V1.bin files:

- **LightSensor_NtagIcode_Ota_OM15081R2_V1.bin**
- **LTOSensor_NtagIcode_Ota_OM15081R2_V1.bin**
- **OccupancySensor_NtagIcode_Ota_OM15082_V1.bin**

The Application Note *Zigbee 3.0 IoT Control Bridge [JN-AN-1247]* has OTA server functionality built into it. A device called OTA_server is provided to host the upgrade images that the clients will request.

The included ZGWUI tool provides an easy to use interface to serve OTA images to client devices. The OTA images are V2/V3.ota files:

- **LightSensor_NtagIcode_Ota_OM15081R2_V2.ota**
- **LightSensor_NtagIcode_Ota_OM15081R2_V3.ota**
- **LTOSensor_NtagIcode_Ota_OM15081R2_V2.ota**
- **LTOSensor_NtagIcode_Ota_OM15081R2_V3.ota**
- **OccupancySensor_NtagIcode_Ota_OM15082_V2.ota**
- **OccupancySensor_NtagIcode_Ota_OM15082_V3.ota**

5.2 OTA Upgrade Operation

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

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

5.3 Image Credentials

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

- **Manufacturer Code:** This is a 16-bit number that is a Zigbee-assigned identifier for each member company. In this application, this number has been set to 0x1037, which is the identifier for NXP. In the final product, this should be changed to the identifier of the manufacturer. The OTA client will compare the Manufacturer Code in the advertised image with its own and the image will be downloaded only if they match.
- **Image Type:** This is a manufacturer-specific 16-bit number in the range 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 sensor - for example, 0x0400 for a Light Sensor or 0x1400 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.

5.4 Encrypted and Unencrypted Images

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

5.5 Upgrade and Downgrade

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

6 Developing with the Application Note

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

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

These are the resources that you should use to develop DK006 Zigbee 3.0 applications, respectively. They are available free-of-charge to authorised users via the MCUXpresso website.

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

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

6.1 Compilation for Specific Chips/SDKs

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

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

6.2 Code Common to All Sensors

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.

6.3 NFC Folder (Zigbee Installation Code Format)

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

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

6.4 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 Rebuilding the Applications.

6.4.1 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.

6.4.2 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**

6.4.3 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).

6.4.4 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             25
```

6.5 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 Rebuilding the Applications.

6.5.1 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.

6.5.2 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**.

6.5.3 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).

6.5.4 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
```

6.6 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 Rebuilding the Applications.

6.6.1 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.

6.6.2 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**

6.6.3 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).

6.6.4 Configuration Macros

The following macros, which can be found in **App_LightTemperatureOccupancySensor.h**, are available for configuring the sensor. The light is measured in Lux, the temperature is measured in increments of 0.01C.

```
#define LIGHT_SENSOR_MINIMUM_MEASURED_VALUE                     0x0001
#define LIGHT_SENSOR_MAXIMUM_MEASURED_VALUE                     0xFAF
#define LIGHT_SENSOR_MINIMUM_REPORTABLE_CHANGE                  25


#define TEMPERATURE_SENSOR_MINIMUM_MEASURED_VALUE               -4000
#define TEMPERATURE_SENSOR_MAXIMUM_MEASURED_VALUE               12500
#define TEMPERATURE_SENSOR_MINIMUM_REPORTABLE_CHANGE            25
```

6.7 Debugging the Demonstration Application

6.7.1 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.

7 Advanced User Information

7.1 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.

7.2 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 relevant SDK.

8 Related Documents

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

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

Important Notice

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 23-Dec-2020
Document identifier: JN-AN-1246