



# Application Note: JN-AN-1171

## ZigBee Light Link Solution

---

This Application Note demonstrates a typical ZigBee Light Link (ZLL) network based on the NXP JN516x wireless microcontroller. The solution employs the following device types from the ZigBee Light Link Profile Specification version 1.0:

- Lighting devices
- Controller devices

The accompanying software uses the ZLL clusters to transfer data between the devices in a wireless network in order to control Lighting devices from Controller devices. The hardware for the devices is implemented using components from the NXP JN516x-EK001 Evaluation Kit. The device software was developed using NXP Application Programming Interfaces (APIs).

---

## 1 Introduction

This Application Note provides a ZigBee Light Link (ZLL) wireless network solution which uses the NXP JN516x-EK001 Evaluation Kit. The demonstration allows the user to control ZLL Lighting devices from ZLL Controller devices. From the evaluation kit:

- the Lighting/Sensor Expansion Boards (DR1175) are used as ZLL Lighting devices
- the Remote Control Unit (DR1159) is used as a ZLL Controller device

The evaluation kit and its components are described in the *JN516x-EK001 Evaluation Kit User Guide (JN-UG-3093)*.

This Application Note provides implementations of the ZLL Lighting devices and ZLL Controller devices, as specified in the *ZigBee Light Link Profile Specification v1.0 (11-0037-10)*. Information on these devices, such as the clusters that they support, is also provided in NXP's *ZigBee Light Link User Guide (JN-UG-3091)*, which you are advised to study in order to familiarise yourself with ZLL concepts before using this Application Note.

Pre-built application binaries for implementing the ZLL devices on the JN516x-EK001 Evaluation Kit hardware are supplied with this Application Note. You will need to program the binaries into the appropriate boards, as indicated in Section 3. Instructions for running the demonstration are provided in Section 4. Instructions for re-building the application binaries are provided in Section 7.3.

The device software was developed using the following NXP Application Programming Interfaces (APIs): ZigBee PRO APIs, JenOS APIs, ZLL API and JN516x Integrated Peripherals API. These APIs are described in their own User Guides.

### 1.1 System Overview

This example ZigBee Light Link (ZLL) network consists of a combination of ZLL Controller and ZLL Lighting devices. The sub-sections below provide a brief introduction to each device type. Advanced user information is provided in Section 5.

### 1.1.1 ZLL Controller Device

The ZLL Controller device resides on the Remote Control Unit (from the evaluation kit) which acts as a Sleeping End Device in the network. The ZLL Controller device is used to commission the lighting network using Touchlink and to control the operation of the Lighting devices, once the network is formed. For the operational details, refer to Section 4 “Running the Demonstration”.

The ZLL Controller device can be any of the device types listed in the table below. Each device type includes mandatory and optional clusters to provide the functionality defined in the ZLL Profile Specification.

ZLL Controller Device	Clusters Implemented	Comments
Non-Colour Controller	Identify, Groups, On/Off, Level Control	This device is not capable of controlling scenes or the colour of Lighting devices.
Colour Controller	Identify, Groups, On/Off, Level Control, Colour Control	This device is not capable of controlling scenes on Lighting devices.
Non-Colour Scene Controller	Identify, Groups, On/Off, Scenes, Level Control	This device is not capable of controlling the colour of Lighting devices.
Colour Scene Controller	Identify, Groups, On/Off, Scenes, Level Control, Colour Control	Sleeping and non-sleeping build options are provided for this device.
On/Off Sensor	Identify, Groups, On/Off, Scenes, Level Control, Colour Control	-

### 1.1.2 ZLL Lighting Devices

The ZLL Lighting devices reside on permanently powered nodes that act as Routers in the network. On receiving Touchlink commands (from the Controller device), a ZLL Lighting device can start a network or join an existing network.

A ZLL Lighting device can be any of the device types listed in the table below. Each device type includes mandatory and optional clusters to provide the functionality defined in the ZLL Specification.

ZLL Lighting Device	Clusters Implemented	Comments
On/Off Light	Groups, Scenes, Identify, On/Off, Level Control	This device is not capable of colour or level control (Level Control cluster is included for a consistent user experience so that Level Control “with On/Off” commands can be interpreted when the device is in a group with dimmable lights).
On/Off Plug-in Light	Groups, Scenes, Identify, On/Off, Level Control	This device is not capable of colour or level control (Level Control cluster is included for a consistent user experience so that Level Control “with On/Off” commands can be interpreted when the device is in a group with dimmable lights). The device is typically included in nodes that contain a controllable mains plug or adaptor.
Dimmable Light	Groups, Scenes, Identify, On/Off, Level Control	This device is not capable of colour or level control.
Dimmable Plug-in Light	Groups, Scenes, Identify, On/Off, Level Control	This device is not capable of colour or level control. It is typically included in nodes that contain a controllable mains plug or adaptor.
Colour Light	Groups, Scenes, Identify, On/Off, Level Control, Colour Control	This device supports manipulation of the lamp colour using hue and saturation, enhanced hue, colour loop and CIE XY coordinates. It does not support colour temperature.

Extended Colour Light	Groups, Scenes, Identify, On/Off, Level Control, Colour Control	This device supports manipulation of the lamp colour using hue and saturation, enhanced hue, colour loop, CIE XY coordinates and colour temperature.
Colour Temperature Light	Groups, Scenes, Identify, On/Off, Level Control, Colour Control	This device supports manipulation of the lamp colour using colour temperature.

## 2 Compatibility

The software provided with this Application Note is intended for use with the following evaluation kit and SDK (Software Developer's Kit) versions:

Product Type	Part Number	Version or Build
Evaluation Kit	JN516x-EK001	-
JN516x ZLL/HA SDK	JN-SW-4168	1620
'BeyondStudio for NXP' Toolchain	JN-SW-4141	1308

## 3 Loading the Application

Table 1 below lists the application binary files supplied with this Application Note and indicates the JN516x-EK001 Evaluation Kit components on which the binaries can be used. For the Light devices, binaries are provided for JN5168 and JN5169 – in the table below, <x> can be 8 or 9.

Device Type / Application Binary	Expansion Board ( + Carrier Board )			Remote Control Unit
	Generic	LCD	Lighting/Sensor	
Light_ColorLight_JN516<x>			■	
Light_ColorTemperatureLight_JN516<x>			■	
Light_DimmableLight_JN516<x>			■	
Light_DimmablePlug_JN516<x> *			□	
Light_ExtendedColorLight_JN516<x>			■	
Light_OnOffLight_JN516<x>			■	
Light_OnOffPlug_JN516<x> *			□	
Controller_ColorController_JN5168				■
Controller_ColorSceneController_JN5168				■
Controller_NonColorController_JN5168				■
Controller_NonColorSceneController_JN5168				■
Controller_OnOffSensor_JN5168 **				□

**Table 1: Device Type – Evaluation Kit Compatibility Matrix**

**Key:** ■ – Preferred hardware □ – Reduced functionality

\* Not fully-functioning plug (same as 'Light' version except for Device ID)

\*\* Sensor functionality of **Controller\_OnOffSensor\_JN5168.bin** is not supported on the Remote Control Unit.

In practice, any controller can be used with any light, but the following limitations apply in the control of lights:

- A 'non-colour' controller can only perform on/off and level control
- A 'non-scene' colour controller can control colour but not save or restore scenes
- A Colour Scene Controller can perform all control operations

From the pre-built binaries supplied for the JN516x-EK001 Evaluation Kit (see Table 1), as a starting point and to fully use the features described, you are advised to program the:

- Lighting/Sensor Expansion Board with **Light\_ExtendedColorLight\_JN516<x>.bin**
- Remote Control Unit with **Controller\_ColorSceneController\_JN5168.bin**

The application binaries must be loaded into the corresponding evaluation kit boards using the JN516x Flash Programmer within BeyondStudio for NXP or the JN51xx Production Flash Programmer (JN-SW-4107).



**Caution:** If loading this application for the first time, the persistent data must be cleared in each of the devices using the 'Erase EEPROM' option in the JN516x Flash Programmer.



**Note:** The supplied software can also be built for NXP's LED bulb reference designs that relate to boards DR1190, DR1192 and DR1221. In this case, use the build configuration appended with DR1190, DR1192 or DR1221, as required, within BeyondStudio for NXP.

## 4 Running the Demonstration

This section describes how to demonstrate the ZLL application. You should have programmed binaries into JN516x evaluation kit boards as described in Section 3. As stated, you are advised to use a Colour Scene Controller and an Extended Colour Light.

### 4.1 Forming the ZLL Network

The primary method of forming a ZLL network is by 'Touchlink' commissioning, which is introduced in the *ZigBee Light Link User Guide (JN-UG-3091)*. Touchlink uses the inter-PAN protocol, described in the *ZigBee PRO Stack User Guide (JN-UG-3101)*. Touchlinking is performed at reduced RF power, so the devices need to be brought into close proximity. As part of Touchlinking, there are two roles that devices can play: Initiator and Target. A Controller can be both an Initiator and a Target, while Lights can only be Targets.

In order to form a new ZLL network, you will need a factory-new Controller device (Remove Control Unit) and a Lighting device. It does not matter whether the Lighting device is factory-new, as its network settings will acquire values for the new network as part of the Touchlink commissioning.

1. Power up the Lighting device (Lighting/Sensor Expansion Board). The light will attempt to classically join any open ZigBee Light Link (or ZigBee Home Automation) network that it finds. When this fails, it will randomly pick one of the ZLL primary channels (11, 15, 20 or 25) and wait with its receiver on for Touchlink commissioning.



**Note:** To reset a Lighting device to the factory-new state, press and hold down the button 'DIO8', and then press and release button 'RST' on the Carrier Board.

2. Power up the Controller device (Remote Control Unit) by inserting batteries.



**Note 1:** The Controller device goes into sleep mode after a while. To wake up the Remote Control Unit, press hard on the button ●.



**Note 2:** The Controller device is in the factory-new state if no LEDs blink when buttons on the capacitive-touch keypad are pressed. If the Controller device is not in the factory-new state on first use then the device must be reset to the factory-new state.



**Note 3:** To reset the Controller device to the factory-new state, press the button \* repeatedly until both of the LEDs illuminate and then enter the button sequence –, +, –.

3. Perform Touchlink commissioning.



**Note:** Before performing Touchlink commissioning, make sure that the Controller device (Remote Control Unit) is not in the sleep state.

- a) For successful commissioning, make sure the Controller device is close to the target Lighting device (approximately 10 cm away).
- b) Now press the # button to start the commissioning.

The Controller device will issue inter-PAN scan requests across the ZLL primary channels 11, 15, 20 and 25. Any target devices within range will respond with a scan response and the Controller device will select the light in closest proximity to be the target of the Touchlinking. If there are no suitable responses from the scan of the primary channels then the secondary channels will be scanned for a suitable target. Once a Touchlink target has been selected, an Identify command will be sent to it (the identify action of a target will be device-specific: a colour light will go red, a monochrome light will flash). After this identification, the Controller device will send a Device Information Request to the target device which will respond with the information that is required to control the operation of the light (Device ID, Profile ID, endpoint number). The Controller device will then send a Network Start Command to the target, containing the network address that the light must use and a randomly generated network key to use. This key is encrypted using the ZLL certification key. The target light will set up its network parameters and start as a ZigBee Router on the chosen channel and PAN. Two seconds after this, the Controller device will issue a ZigBee Rejoin Request to join the Router just started. The network is now formed.

- c) The Controller device will now add the device, profile, endpoint number and address of the new light to its light database. It will also send an Add Group command to the Lighting device to assign the light to its control group, and finally it will send an 'Identify with Okay effect' command to the light to signal completion of the Touchlinking.

After successful commissioning, the Controller device can control the Lighting device with the button mapping described in the next section.

## 4.2 Adding More Lights to the Network

To add more lights to the network, repeat the Touchlink procedure above. The Controller device will assign a network address and pass on the network parameters to the new lights using the inter-PAN Network Join Router command.

## 4.3 Adding More Controllers to the Network

Touchlinking can be used to add more Controller devices to the network. The new Controller device must be in the factory-new state - it is not possible to commission Controller devices that are not factory-new.

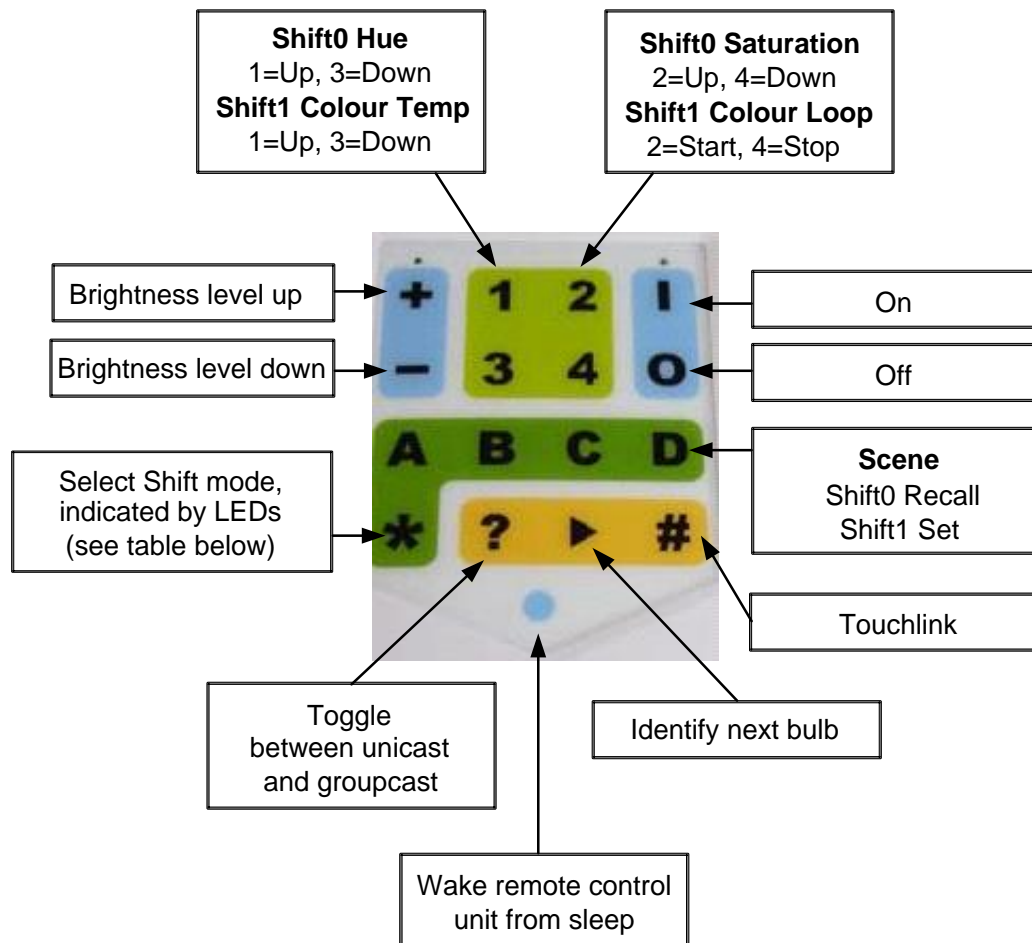
1. Bring the new Controller device and existing Controller device within close proximity of each other, and away from other ZLL devices.
2. Ensure that both devices are not sleeping and simultaneously press the # keys on the two devices to start the commissioning.
3. Both Controller devices will issue inter-PAN scan requests and each will respond to the other's scan. However, the factory-new device will abandon its scanning and defer to the non-factory-new device to become the target for Touchlink.
4. The new Controller device will be passed its network parameters and address in a Network Join End Device command in order to become part of the network. Since Controller devices are 'address assignment capable', the existing Controller device will split the range of network and group addresses that it holds for assignment (to new devices) - it retains half for itself and passes the other half to the new Controller device for use in future Touchlinking.
5. The new Controller device will now issue a ZigBee Rejoin Network command and join a suitable parent Router.
6. After successfully joining the network, the original Controller device will send an Endpoint Information command to the new Controller device to inform it of its endpoint number. The new Controller device can now send a Get Endpoint List command to the original Controller device, which will then respond with the contents of its light database. A Get Group Identifiers command will also be used to inform the new Controller device of the Group IDs used by the original device.
7. The new Controller device will now be able to control the lights in the network.

## 4.4 Touchlinking to Lights Already in the Network

Touchlinking can be used to gather endpoint and device type information from lights already in the same network. This is useful in the situation where there are two Controller devices and the first device is used to add a new light to the network - the second device will have no knowledge of the existence of this new light. By using Touchlink, the second device can gather the device address and endpoint of the new light and add this information to its light database. In this case, there is no need to exchange network parameters, just device information.

## 4.5 ZLL Controller Device (Remote Control Unit) Operation

The button functions on the Controller device (Remote Control Unit) are as shown below.



**Note 1:** When the Controller device is in the factory-new state, only the buttons # and + are available (for commissioning ZLL devices into the network).



**Note 2:** The Controller device goes to sleep after a while. Hence, if the Remote Control Unit LEDs do not blink on pressing a touch-button, press hard on the button • to wake up the unit.



**Note 3:** The Controller device can operate the Lighting device(s) using one of two addressing modes – unicast or groupcast. Unicast mode allows the user to operate a single Lighting device. Groupcast mode allows the user to operate a group of Lighting devices simultaneously. Use the button '?' to toggle between these two modes.

The Controller device can operate in four Shift modes (0, 1, 2 and 3) to accommodate maximum functionality. The Shift mode is indicated by a combination of two LEDs on the Remote Control Unit, as shown in the table below. You can press button \* to move to the next Shift mode.



Shift Mode	Left LED	Right LED
Shift0	Off	Off
Shift1	On	Off
Shift2	Off	On
Shift3	On	On

**Table 2: Shift Modes**

The four tables below summarise the button functions in the four Shift modes.

Shift0 Mode Operation	Button
<b>On:</b> Send a command to switch on the light(s). The transmission mode will depend on the current mode selected.	I
<b>Off with Effect:</b> Send a command to switch off the light(s) using the 'Off with Effect' option - this has the effect of saving the current settings as the Global Scene.	O
<b>Increase Brightness:</b> Increase the brightness level of the light(s). If the light is off, this will switch on the light and then increase its level. The brightness will stop increasing when the button is released.	+
<b>Decrease Brightness:</b> Decrease the brightness level of the light(s). The brightness will stop decreasing when the button is released.	-
<b>Move Hue Up:</b> Send a command to move the hue of the light(s) up. The movement will stop when the button is released.	1
<b>Move Hue Down:</b> Send a command to move the hue of the light(s) down. The movement will stop when the button is released.	3
<b>Increase Saturation:</b> Send a command to move the saturation of the light(s) up. The movement will stop when the button is released.	2
<b>Decrease Saturation:</b> Send a command to move the saturation of the light(s) down. The movement will stop when the button is released.	4
<b>Recall Scene 1:</b> Groupcast a Recall Scene command to restore scene 1.	A
<b>Recall Scene 2:</b> Groupcast a Recall Scene command to restore scene 2.	B
<b>Recall Scene 3:</b> Groupcast a Recall Scene command to restore scene 3.	C
<b>Recall Scene 4:</b> Groupcast a Recall Scene command to restore scene 4.	D
<b>Shift Menu:</b> Cycle through the four Shift modes (0 → 1 → 2 → 3 → 0 etc).	*
<b>Groupcast/Unicast:</b> Toggle between groupcast and unicast transmission modes. On waking from sleep, this mode will always be groupcast. After Touchlinking to a light, the mode will always be unicast with that light selected.	?
<b>Select next light:</b> Select the next light in the light database to be controlled by unicast. An Identify command will be sent to the relevant light, and unicast transmission mode will be selected.	►
<b>Touchlink:</b> Start Touchlink commissioning to add new devices to the network or to gather endpoint information about existing devices in the network.	#

**Table 3: Button Functions in Shift0 Mode**

Shift1 Mode Operation	Button
<b>On:</b> Send a command to switch on the light(s). The transmission mode will depend on the current mode selected.	I
<b>Off with Effect:</b> Send a command to switch off the light(s) using the 'Off with Effect' option - this has the effect of saving the current settings as the Global Scene.	O
<b>Increase Brightness:</b> Increase the brightness level of the light(s). If the light is off, this will switch on the light and then increase its level. The brightness will stop increasing when the button is released.	+
<b>Decrease Brightness:</b> Decrease the brightness level of the light(s). The brightness will stop decreasing when the button is released.	-
<b>Increase Colour temperature:</b> Send a command to increase the colour temperature of the light(s). The increase will stop when the button is released.	1



<b>Decrease Colour temperature:</b> Send a command to decrease the colour temperature of the light(s). The decrease will stop when the button is released.	<b>3</b>
<b>Set the Colour Loop:</b> Send a command to the light(s) to start a colour loop. The light will start cycling through colours.	<b>2</b>
<b>Stop Colour Loop:</b> Send a command to the light(s) to stop the colour loop. A colour loop must be stopped before other colour control commands will be accepted by colour lights.	<b>4</b>
<b>Store Scene 1:</b> Groupcast a Store Scene command to save the current settings as Scene 1. Note that following the saving of a scene, the menu level will automatically revert to Shift0.	<b>A</b>
<b>Store Scene 2:</b> Groupcast a Store Scene command to save the current settings as Scene 2. Note that following the saving of a scene, the menu level will automatically revert to Shift0.	<b>B</b>
<b>Store Scene 3:</b> Groupcast a Store Scene command to save the current settings as Scene 3. Note that following the saving of a scene, the menu level will automatically revert to Shift0.	<b>C</b>
<b>Store Scene 4:</b> Groupcast a Store Scene command to save the current settings as Scene 4. Note that following the saving of a scene, the menu level will automatically revert to Shift0.	<b>D</b>
<b>Shift Menu:</b> Cycle through the four Shift modes (0 → 1 → 2 → 3 → 0 etc).	<b>*</b>
<b>Groupcast/Unicast:</b> Toggle between groupcast and unicast transmission modes. On waking from sleep, this mode will always be groupcast. After Touchlinking to a light, the mode will always be unicast with that light selected.	<b>?</b>
<b>Select next light:</b> Select the next light in the light database to be controlled by unicast. An Identify command will be sent to the relevant light, and unicast transmission mode will be selected.	<b>▶</b>
<b>Touchlink:</b> Start Touchlink commissioning to add new devices to the network or to gather endpoint information about existing devices in the network.	<b>#</b>

**Table 4: Button Functions in Shift1 Mode**

<b>Shift2 Mode Operation</b>	<b>Button</b>
<b>On:</b> Send a command to switch on the light(s). The transmission mode will depend on the current mode selected.	<b>I</b>
<b>Off with Effect:</b> Send a command to switch off the light(s) using the 'Off with Effect' option - this has the effect of saving the current settings as the Global Scene.	<b>O</b>
<b>Increase Brightness:</b> Increase the brightness level of the light(s). If the light is off, this will switch on the light and then increase its level. The brightness will stop increasing when the button is released.	<b>+</b>
<b>Decrease Brightness:</b> Decrease the brightness level of the light(s). The brightness will stop decreasing when the button is released.	<b>-</b>
No function assigned	<b>1</b>
No function assigned	<b>3</b>
No function assigned	<b>2</b>
No function assigned	<b>4</b>
No function assigned	<b>A</b>
No function assigned	<b>B</b>
<b>Permit Join:</b> Broadcast a ZigBee Management command to the network to instruct Routers to set their 'permit joining' state to TRUE for 120 seconds. This opens the network to classical joining.	<b>C</b>
<b>Channel Change:</b> Broadcast a ZigBee Management command to change the operational channel to one of the other ZLL primary channels, selected at random.	<b>D</b>
<b>Shift Menu:</b> Cycle through the four Shift modes (0 → 1 → 2 → 3 → 0 etc).	<b>*</b>
<b>Groupcast/Unicast:</b> Toggle between groupcast and unicast transmission modes. On waking from sleep, this mode will always be groupcast. After Touchlinking to a light, the mode will always be unicast with that light selected.	<b>?</b>

<b>Select next light:</b> Select the next light in the light database to be controlled by unicast. An Identify command will be sent to the relevant light, and unicast transmission mode will be selected.	►
<b>Touchlink:</b> Start Touchlink commissioning to add new devices to the network or to gather endpoint information about existing devices in the network.	#

**Table 5: Button Functions in Shift2 Mode**

Shift3 Mode Operation	Button Sequence
<b>On:</b> Send a command to switch on the light(s). The transmission mode will depend on the current mode selected.	I
<b>Off with Effect:</b> Send a command to switch off the light(s) using the 'Off with Effect' option - this has the effect of saving the current settings as the Global Scene.	O
<b>Factory Reset:</b> Factory reset the Remote Control Unit, restoring the application and stack persistent data to its factory-new state.	- + -
No function assigned	1
No function assigned	3
No function assigned	2
No function assigned	4
<b>Identify Effect Blink:</b> Send a command to the light(s) to trigger the 'Blink' effect.	A
<b>Identify Effect Breathe:</b> Send a command to the light(s) to trigger the 'Breathe' effect.	B
<b>Identify Effect Okay:</b> Send a command to the light(s) to trigger the 'Okay' effect.	C
<b>Identify Effect Channel Change:</b> Send a command to the light(s) to trigger the 'Channel Change' effect.	D
<b>Shift Menu:</b> Cycle through the four Shift modes (0 → 1 → 2 → 3 → 0 etc).	*
<b>Groupcast/Unicast:</b> Toggle between groupcast and unicast transmission modes. On waking from sleep, this mode will always be groupcast. After Touchlinking to a light, the mode will always be unicast with that light selected.	?
<b>Select next light:</b> Select the next light in the light database to be controlled by unicast. An Identify command will be sent to the relevant light, and unicast transmission mode will be selected.	►
<b>Touchlink:</b> Start Touchlink to send a Factory New (reset) command to the target device. This button cannot be used to add new devices to the network.	#

**Table 6: Button Functions in Shift3 Mode**

## 4.6 Light Link and Classical Joining

As an alternative to Touchlink commissioning, it is possible for a ZLL device to classically join either a ZigBee Light Link (ZLL) or ZigBee Home Automation (ZHA) network. This is done using the standard 'discover then associate' used by ZigBee. Joiners will then be issued with a randomly generated network address and the network key sent using a Transport Key Command.

### 4.6.1 Classically Joining a ZLL Network

In a ZLL network, there is no Co-ordinator device to act as the trust centre, so each Router is capable of taking the trust centre role and transporting the network key to a new device, encrypted using the ZLL key. In order to allow a new ZLL device to classically join a ZLL network, the network must first be opened to allow joining. This is done by sending a Management Permit Joining command from the Controller device (Shift2 mode, button **C**) which will open the network for joining for 120 seconds.

- To join a Lighting device to the network, start or power-cycle a factory-new Lighting device, which will attempt classical joining - success will be indicated by a short identify effect on the light.

- To join a Controller device to the network, the device must be factory-new. Start the joining process by pressing the **+** button. Following the join by this method, the Controller device will need to use Touchlinking to gather device and endpoint information about the lights in the network.

### 4.6.2 Classically Joining a ZHA Network

In a ZHA network, there is a trust centre which will authenticate joiners and issue the network key. This will be encrypted with the ZHA key, which is known to the ZLL devices. In order to join a ZHA network, the network must first be opened for joining with a Permit Joining command. How to do this will depend on the implementation of the ZHA network.

- To join a Lighting device to the network, start or power-cycle a factory-new Lighting device, which will attempt classical joining - success will be indicated by a short identify effect on the light.
- To join a Controller device to the network, the device must be factory-new. Start the joining process by pressing the **+** button. Following the join by this method, the Controller device cannot be used to add new devices into the network using Touchlink, as it is not allowed to bypass the security functions of the trust centre. However, Touchlinking can still be used to gather endpoint information etc.

## 5 Advanced User Information

### 5.1 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 the 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.

### 5.2 Clearing Network Context

During the demonstration and development, it is often necessary to clear the context data. To clear context on any ZLL Lighting device (JN516x-EK001 Lighting/Sensor Expansion Board (DR1175) on Carrier Board (DR1174)), hold down the button labelled '**DIO8**' on the Carrier Board while resetting the device.

In order to clear context data on the ZLL Controller device (JN516x-EK001 Remote Control Unit (DR1198)), follow the appropriate instructions from those below:

- When the Controller device is in the factory-new state or not commissioned to any Lighting device (no LED blinks on the Remote Control Unit when any touch-button is pressed) then the context data can be cleared by pressing the touch-button **I** on the keypad.
- When the Controller device has previously been commissioned into a network (LED blinks when a touch-button is pressed) then the context data can be cleared by means of the following steps:
  - a. Press the button **\*** repeatedly until both of the LEDs illuminate.
  - b. Now enter the button sequence **-, +, -**.

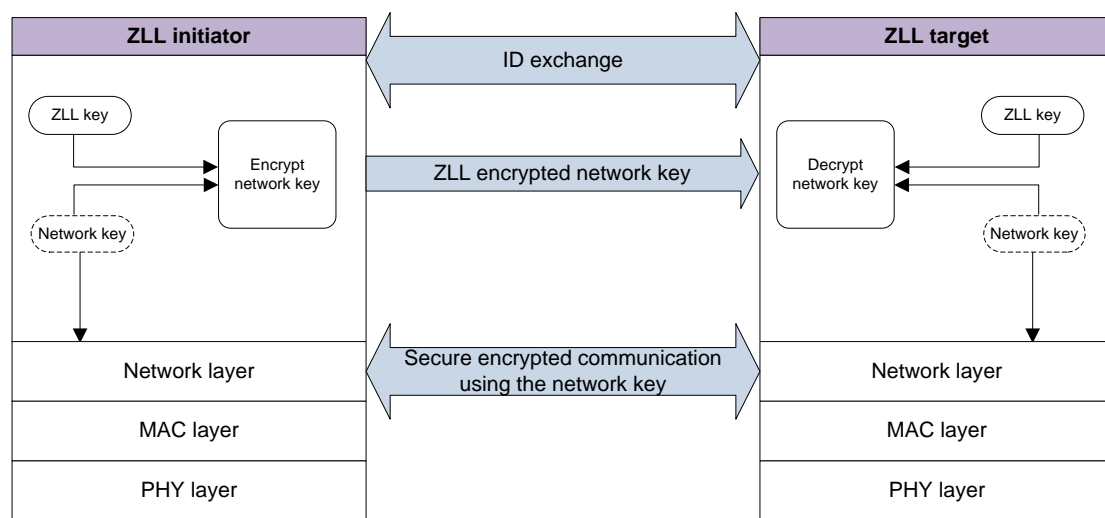
## 5.3 Security Keys and MAC Addresses

The application uses the MAC address pre-programmed in the devices.

All the nodes in a ZLL system use ZigBee network layer security to generate a network key for the communication with the network security enabled devices. The network key is generated randomly by the initiator that starts the new network. All ZLL devices use the ZLL key to encrypt/decrypt the exchanged network key.

ZLL provides Do-It-Yourself (DIY) installation and commissioning. It is required to have a common yet secured technique to transfer the network keys during classical joining methods or Touchlink commissioning. For ease of use during development, this key data has been hardcoded within the application.

Figure 1 illustrates the architecture that is used for transferring the encrypted network key. Refer to the following sub-sections for more information.



**Figure 1: Overview of ZLL Security**

The encryption and decryption are done by the following functions:

- **PRIVATE uint8 eEncryptKey()**
- **PRIVATE uint8 eDecryptKey()**

Please refer to ZigBee Light Link Specification for more details.

### 5.3.1 Certification Key

Prior to the successful completion of certification, a certificate key should be used to allow testing of the security mechanisms as specified in ZigBee Light Link Specification.

### 5.3.2 Master Key

The ZLL master key is a secret key shared by all certified ZLL devices. It will be distributed only to certified manufacturers by the ZigBee Alliance.



**Note:** The software provided in this Application Note is NOT supplied with the ZLL master key. As such, it is not capable of completing Touchlink commissioning with commercial 'off-the-shelf' ZigBee Light Link products.

## 5.4 Adding More Groups to a Device

Each Lighting device is configured with a group addressing table of size 5. This allows each light to be a member of 5 different groups. To increase this group addressing table size, increase the value of the "Group Addressing Table Size" in the ZPS configuration editor. Also update the `CLD_GROUPS_MAX_NUMBER_OF_GROUPS` macro in the `zcl_options.h` file to match the updated value of "Group Addressing Table Size" in the ZPS configuration editor.

Similarly, for the Controller device, the number of group records is configured to 4. To increase this number, edit the `NUMBER_GROUP_RECORDS` and `GROUPS_REQUIRED` macros in the `zcl_options.h` file for the respective device.

## 5.5 Adding More Scenes to a Lighting Device

Each Lighting device is configured to have 9 scenes, of which one scene, scene 0, is reserved for use as the Global Scene. This allows 8 scenes to be added in the device. To increase the number of scenes, edit the `CLD_SCENES_MAX_NUMBER_OF_SCENES` macro in the `zcl_options.h` file for the respective device. Care should be taken when adding more scenes to ensure that the EEPROM remains capable of saving all the required data. The Persistent Data Manager (PDM) handles this, but there must be enough free sectors of EEPROM to save the largest record. If this condition is met then the PDM will be 'save safe'. The Occupancy and Capacity parameters of the PDM may be examined to give information about the state of the PDM. A callback function can be enabled to give application indications of PDM error conditions. The use of these is recommended during development.

```
DBG_vPrintf(TRACE_APP|1, "PDM: Capacity %d\n",
u8PDM_CalculateFileSystemCapacity() );
DBG_vPrintf(TRACE_APP|1, "PDM: Occupancy %d\n",
u8PDM_GetFileSystemOccupancy() );
PDM_vRegisterSystemCallback(vPdmEventHandlerCallback);
```

Both of these feature are present in the application under the `TRACE_APP` build option.

# 6 Over-The-Air (OTA) Upgrade

## 6.1 Overview

Support for the OTA Upgrade cluster and OTA clients has been included for the Extender Colour Light, Colour Light, Colour Temperature Light, Dimmable Light and On/Off Light devices. In order to build with these options, add `OTA=1` to the command line before building. This will add the relevant functionality to the lights and invoke post-build processing to create a bootable image and two upgrade images. The produced binaries will be stored in the **OTA\_build** directory. By default, unencrypted binaries will be produced. In order to build encrypted binaries, add the `OTA_ENCRYPTED=1` option to the command line before building.

- If built for the JN5168 device then external Flash memory will be used to store the upgrade image before replacing the old one.
- If built for the JN5169 device then the internal Flash memory will be used to store the upgrade image.

A device called `OTA_server` is provided to host the upgrade images that the clients will request. This OTA Upgrade server must be implemented on a JN5168 device (and cannot be implemented on a JN5169 device).

## 6.2 OTA Upgrade Operation

To implement an OTA upgrade:

1. Build the light application with `OTA=1` in the makefile to enable OTA upgrade (this option is not enabled by default). Also comment out the line `CFLAGS += -DDEBUG_APP_OTA` in the makefile so that the upgrade can be seen over the UART of the light node.

The binary files for the light are created in the **OTABuild** folder – bootable binaries have the extension **.bin** and upgrade binaries have the extension **.ota** (the latter is intended to be loaded into external Flash memory of the OTA Upgrade server using the JN51xx Production Flash Programmer (JN-SW-4107), as described in Step 7). There are three encrypted binaries and three non-encrypted binaries. The three binaries in each set are different versions with different headers so that we can test the upgrading of the light.

2. Program one of the binary files from the **OTABuild** folder into the internal Flash memory of the JN516x device on the DR1174 Carrier Board of the light node - for example, **Light\_ExtendedColorLight\_JN5168\_DR1175.bin**. You can do this using the JN516x Flash Programmer within BeyondStudio for NXP, as described in the *BeyondStudio for NXP Installation and User Guide (JN-UG-3098)*. Alternatively, you can use the JN51xx Production Flash Programmer (JN-SW-4107) described in the *JN51xx Production Flash Programmer User Guide (JN-UG-3099)*.
3. Build the OTA server application. The resulting binary file, **server.bin**, will be created in the **OTABuild** directory.
4. Program the file **server.bin** into the internal Flash memory of the JN5168 device on the DR1198 USB Dongle, so that the dongle becomes the OTA Upgrade server. You can do this using the JN516x Flash Programmer within BeyondStudio for NXP or the JN51xx Production Flash Programmer (JN-SW-4107).
5. Form a network with a light node and the DR1159 Remote Control Unit in the normal way using Touchlink (see Section 4.1).
6. Open the network for classical joining from the Remote Control Unit by pressing the button sequence **\* \* C** (if both LEDs on the unit are off). After pressing **\* \***, the Remote Control Unit will be in Shift2 mode (see Section 4.5) and, to indicate this, the left LED will be Off and the right one will be On. Pressing **C** broadcasts a command to the network nodes to enable 'Permit Joining' on the nodes for 120 seconds.

The OTA Upgrade server device will now classically join the network.

7. Load a **.ota** upgrade image (V2 or V3) into the external Flash memory of the JN5168 device on the DR1198 USB Dongle using the JN51xx Production Flash Programmer (JN-SW-4107) - the required command line will be similar to the following:

```
Jn51xxProgrammer -S external -s COM<port> -f <filename>
```

Alternatively, you can load a **.bin** upgrade image (V2 or V3) into the external Flash memory of the JN5168 device on DR1198 USB Dongle using the obsolete JN51xx Flash Programmer v1.8.9.

8. When viewing the UART output from the light node, the upgraded image should be found and the light node upgraded.

Any devices with OTA clients in the network will periodically send Match Descriptor Requests in order to find an OTA server. Once a server responds, it will then be sent an IEEE Address Request in order to confirm its address details. After this, the clients will periodically send OTA Image Requests to determine whether the server is hosting an image for that client device. In response to the Image Request, the server will return details of the image that it is currently hosting - Manufacturer Code, Image Tag and Version Number. The client will check these credentials and decide whether it requires this image. If it does not, it



will query the server again at the next query interval. If the client does require the image, it will start to issue Block Requests to the server to get the new image. Once all blocks of the new image have been requested and received, the new image will be verified, the old one invalidated, and the device will reboot and run the new image. The client will resume periodically querying the server for new images

## 6.3 Image Credentials

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

- **Manufacturer Code:** This is a 16-bit number that is a ZigBee-assigned identifier for each member company. In this application, this number has been set to 0x1037, which is the identifier for NXP. In the final product, this should be changed to the identifier of the manufacturer. The OTA client will compare the Manufacturer Code in the advertised image with its own and the image will be downloaded only if they match.
- **Image Type:** This is a manufacturer-specific 16-bit number in the range 0x000 to 0xFFBF. Its use is for the manufacturer to distinguish between devices. In this application, the Image Type is set to the ZigBee Device Type of the bulb - for example, 0x0210 for an Extended Colour Light or 0x1210 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, it is possible (through a build option) for the OTA client to compare the string in the advertised image with its own string before accepting an image for download. If the strings match then the image will be accepted. In this way, the string can be used to provide extra detail for identifying images, such as hardware sub-types.

## 6.4 Encrypted and Unencrypted Images

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

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

```
#define INTERNAL_ENCRYPTED
```

## 6.5 Upgrade and Downgrade

The decision to accept an image following a query response is under the control of the application. The code, as supplied, will accept an upgrade or a downgrade. As long as the notified image has the right credentials and a version number which is different from the current version number, the image will be downloaded. For example, if a client is running a V3 image and a server is loaded with a V2 image then the V2 image will be downloaded. If it is required that the client should only accept upgrade images (V2 > V3 > V5), or only accept

sequential upgrade images (V2 > V3 > V4 > V5) then the application callback function that handles the Image Notifications in the OTA client will need to be modified.

## 7 Developing with the Application Note

This section provides additional information that may be useful when developing with this Application Note.

### 7.1 Useful Documents

Before commencing a ZigBee Light Link development, you are recommended to familiarise yourself with the following documents:

- [R1] - JN-UG-3101 ZigBee PRO User Guide
- [R2] - JN-UG-3075 JenOS User Guide
- [R3] - JN-UG-3091 ZigBee Light Link User Guide
- [R4] - JN-UG-3103 ZigBee Cluster Library User Guide
- [R5] - JN-UG-3087 JN516x Integrated Peripherals API User Guide
- [R6] - ZigBee Light Link (ZLL) Profile Specification
- [R7] - ZigBee Cluster Library (ZCL) Specification

The latest versions of [R1] to [R5] can be obtained from the [Wireless Connectivity](#) area of the NXP web site, while [R6] and [R7] can be found on the ZigBee Alliance web site, <http://www.zigbee.org>.

### 7.2 Debugging the Demonstration Application

#### 7.2.1 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:

<b>BAUD Rate</b>	115200
<b>Data</b>	8 bits
<b>Parity</b>	None
<b>Stop bit</b>	1 bit
<b>Flow Control</b>	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.

#### 7.2.2 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)*.

### 7.2.3 On-Air Packets

The demonstration uses the following pre-configured link key and channel mask, in case you wish to capture on-air data packets with a protocol analyser (such as Ubiqua from Ubilogix):

Pre-configured Link Key	0xd0d1d2d3d4d5d6d7d8d9daddbdcdddedf
Channel Mask	11, 15, 20 and 25

## 7.3 Building and Downloading the Application

This section provides application build instructions. If you simply wish to use the supplied application binaries, refer to Section 3.

### 7.3.1 Pre-requisites and Installation

Before you start to build and load the application, please ensure that you have following installed on your development PC:

- BeyondStudio for NXP (JN-SW-4141)
- JN516x ZigBee Light Link SDK (JN-SW-4168)



**Note:** For the installation instructions, please refer to *BeyondStudio for NXP Installation and User Guide (JN-UG-3098)* and the Release Notes supplied with the JN516x ZigBee Light Link SDK (JN-SW-4168).

In order to build the application, this Application Note (JN-AN-1171) must be unzipped into the directory:

**<BeyondStudio for NXP installation root>\workspace**

where **<BeyondStudio for NXP Installation root>** is the path into which BeyondStudio for NXP was installed (by default, this is **C:\NXP\bstudio\_nxp**). The **workspace** directory is automatically created when you start BeyondStudio for NXP.

All files should then be located in the directory:

**...\workspace\JN-AN-1171-ZigBee-LightLink-Demo**

There is a sub-directory for each application, each having **Source** and **Build** sub-directories.

### 7.3.2 Build Instructions

The software provided with this Application Note can be built for the JN5169, JN5168 or JN5164 device (JN5169 is the default chip for Lights, JN5168 is the default chip for Controllers and the OTA Server).

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

- To build using makefiles, refer to Section 7.3.2.1.
- To build using BeyondStudio for NXP, refer to Section 7.3.2.2.

### 7.3.2.1 Using Makefiles

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

To build an application and load it into a JN516x device, follow the instructions below.



**Note:** The make commands given below will build the application according to the default build options in the makefile (e.g. device type). To use alternative build options, these must be specified in the make command. The required options for different builds can be obtained from the build configurations provided in BeyondStudio for NXP.

1. Ensure that the project directory is located in  
**<BeyondStudio for NXP 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 follow the instructions below for your chip type:

**For JN5169:**

At the command prompt, simply enter:

```
make JENNIC_CHIP=JN5169 clean all <non-default make arguments>
```

**For JN5168:**

At the command prompt, enter:

```
make JENNIC_CHIP=JN5168 clean all <non-default make arguments>
```

**For JN5164:**

At the command prompt, enter:

```
make JENNIC_CHIP=JN5164 clean all <non-default make arguments>
```

In all the above cases, the binary file will be created in the **Build** directory, the resulting filename indicating the chip type (e.g. **5168**) for which the application was built.


4. Load the resulting binary file into the device. You can do this from the command line using the JN51xx Production Flash Programmer (described in the *JN51xx Production Flash Programmer User Guide (JN-UG-3099)*).

### 7.3.2.2 Using BeyondStudio for NXP

This section describes how to use BeyondStudio for NXP to build the demonstration application.

To build the application and load it into JN516x devices, follow the instructions below:

1. Ensure that the project directory is located in  
**<BeyondStudio for NXP installation root>\workspace**
2. Start the BeyondStudio for NXP and import the relevant project as follows:
  - a) In BeyondStudio, 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 and click **Finish**.
3. In the makefile(s) for application(s) to be built, ensure that the JN516x chip on which the application is to run is correctly specified in the line beginning JENNIC\_CHIP. For example, in the case of the JN5169 device, this line should be:  

```
JENNIC_CHIP=JN5169
```
4. Build an application. To do this, ensure that the project is highlighted in the left panel of BeyondStudio 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.
5. Load the resulting binary files into the devices. You can do this using the integrated Flash programmer, as described in the *BeyondStudio for NXP Installation and User Guide (JN-UG-3098)*.

## 7.4 Application Start-up

This section describes the typical start-up flow of an NXP ZigBee PRO device. Note that not all devices sleep, hence the 'Warm Start' path is not always applicable.

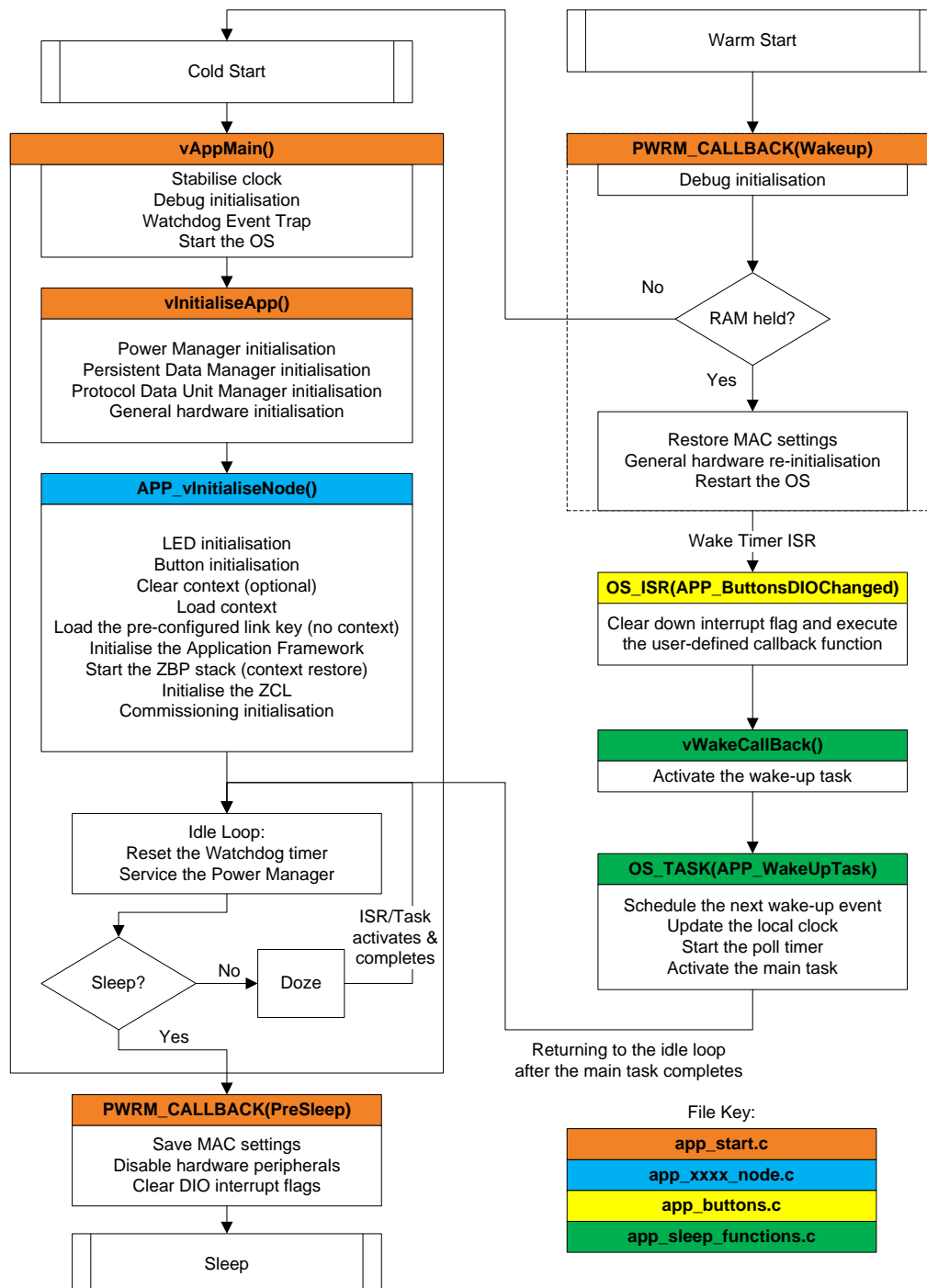


Figure 2: Typical Start-up Flow



## 7.5 ZLL Device Start-up

The start-up flow for different ZLL devices (Lighting or Controller) with respect to different states (Factory-New or Non-Factory-New) are described in the sub-sections below.

### 7.5.1 Factory-New Lighting Device

A ZLL Lighting device acts as a Router in the ZigBee network. On power-up, a factory-new Lighting device performs association discovery on the ZLL primary channels (11, 15, 20 and 25). The Lighting device tries to join any open network through MAC association when a network is found during the discovery.

If no network is found or the association on the primary channel is unsuccessful, the factory-new Light device scans the secondary channels and attempts MAC association when a network is found.

If the light successfully joins a network, it will indicate this by self-identifying for a few seconds.

Whenever the above procedure fails, the Router selects a random primary channel number from the set of primary channels and a random PAN ID. At this point, it switches its radio receiver on. An initiator device will then be able to 'Touchlink' to it.

### 7.5.2 Non-Factory-New Lighting Device

Once the Lighting device has been commissioned into a network, it stores its own state as persistent data. When it powers up again, the device restores the network and application settings, and starts as a Router. In line with the ZLL specification, it sends out three Device Announce commands.

### 7.5.3 Factory-New Controller Device

The Controller device is an End Device in the ZLL network and is termed a network 'initiator'. Hence, on power-up, the device waits for a user action to initiate a Touchlink action or classical commissioning.

In this application, the classical joining is initiated by pressing the button '+'. For the Router, 'Permit Join' must be set to TRUE. This can be achieved on the Lighting device by pressing the button labelled DIO8 on the Carrier Board (DR1174).

### 7.5.4 Non-Factory-New Controller Device

On power-up, the non-factory-new Controller device attempts to join a network that was initiated or commissioned earlier. The device also restores the stack settings from the non-volatile memory and attempts a network rejoin. If successful, the Controller device then resumes normal operation.

In the case of a failure to rejoin a network on power-up, the Controller device waits for a configurable time of 20 seconds to allow the user to use the Controller to 'Touchlink' to another device. After this, it enters sleep with RAM on for another configurable time of 20 seconds before entering deep sleep mode to conserve battery power.

## 7.6 Memory Allocation from Application

The application is responsible for allocating memory for the ZigBee Cluster Library (ZCL). The sub-sections below describe the important memory allocations.

### 7.6.1 Device Table

The application must allocate the device table of type **tsCLD\_ZIIDevice** for storing the device records for its own endpoints. The ZCL accesses this table through the API function **psGetDeviceTable()**.

### 7.6.2 Endpoint Record Table

The remote control application must hold an endpoint information table of type **tsZIIEndpointInfoTable**. This table holds the network address, device type, profile ID and endpoint number of each light of which it controls. This information is gathered as part of the Touchlinking process. The ZCL accesses this table through the API function **psGetEndpointRecordTable()**.

### 7.6.3 Group ID Table

The application must allocate the groups record table of type **tsZIIGroupInfoTable**. The ZCL accesses this table through the API function **psGetGroupRecordTable()**.

## 7.7 Commissioning

In general, all the ZLL devices can join a ZLL network or non-ZLL network using either of the following methods:

- ZLL Touchlink commissioning
- Classical joining

### 7.7.1 ZLL Touchlink Commissioning

A ZLL device can be in either of two states for the Touchlink operation – Factory New (FN) or Non-Factory New (NFN). This state will determine how a device responds to 'Touchlinking'.

In this application, a Controller device can be both a Touchlink initiator and target while in the FN state. The Controller device in the NFN state cannot be a target and needs to be factory reset before it can be commissioned to any network. The Lighting devices are always considered as targets in both the FN and NFN states.

The Touchlink implementation is present in the **app\_remote\_comission\_task.c** file for the Controller device and **app\_light\_comission\_task.c** file for the Lighting device. Both of these files contain a commissioning state machine as an OS task, **APP\_Commission\_Task**. All the Touchlink commands are performed from this task.

A timer task, **APP\_CommissionTimerTask**, is associated with **APP\_Commission\_Task** for the periodic activities in commissioning such as transaction ID expiration.

During Touchlink commissioning, there is a requirement to reduce the transmission power. Setting the **ADJUST\_POWER** flag to **TRUE** will enable the following function for transmission power adjustment.

```
eAppApiPlmeSet(PHY_PIB_ATTR_TX_POWER, TX_POWER_LOW);
```

The normal power is restored by calling the following function:

```
eAppApiPlmeSet(PHY_PIB_ATTR_TX_POWER, TX_POWER_NORMAL);
```

The minimum Link Quality Indication (LQI) to complete the Touchlink process is set by the **ZLL\_SCAN\_LQI\_MIN** macro.

The Touchlink process comprises the phases outlined in the sub-sections below. For more details, refer to the ZLL Specification.

### 7.7.1.1 Device Discovery

The device discovery phase is initiated by the Controller device. The initiator broadcasts a Touchlink Scan Requests on the primary channel set (11, 15, 20 and 25) as inter-PAN messages to the devices that are present within Touchlink proximity (for this application, around 10cm or closer). In this process, 5 scan requests are sent out on channel 11, followed by one request on each of channels 15, 20 and 25. In between each broadcast scan request, there is a waiting time of *aplcScanTimeBaseDuration* (250 ms) to receive any responses.

If the primary channel scan does not get any responses, the Controller device will send out similar scan requests on each of the secondary channels (12, 13, 14, 16, 17, 18, 19, 21, 22, 23, 24 and 26) with a waiting time of *aplcScanTimeBaseDuration*.

On receiving any response that has a valid transaction ID, the initiator selects as its target the closest device, as determined by the received signal strength.

### 7.7.1.2 Identify

The Touchlink initiator issues an Inter-PAN Identify command as an optional command to identify the device from which it has received a response - this is mainly to confirm the device identity through a user-defined effect. If the identifying device is not the user's desired Touchlink target, by pressing # again that target will be dropped and ignored in the new scan sequence that will start. This can be repeated 3 times.

### 7.7.1.3 Device Information Request Command

The remote control unit will send a Device Information Request to the target to obtain information about the endpoint(s) of the target device.

### 7.7.1.4 Starting a Network

Once the discovery phase is complete and the target is identified, if the initiator is in the Factory New (FN) state and the target is a Router then the initiator sends out a Start Network request to the target with the required network settings. The Router then starts the network and sends out a Start Network response. After receiving a response from the Router indicating a successful network start, the initiator rejoins the network.

### 7.7.1.5 Joining a Light Device to the Network

After the discovery and identify phases, if the initiator is in the Non-Factory New (NFN) state and the target is a Router then it sends a 'Join network as Router' command to the target.

The above sections allow a target Lighting device to be in any state, either FN or NFN. In the case of a Lighting device in the NFN state, the device first issues a network leave command to the existing network before it can become part of the initiator's network. This is termed "stealing" a Lighting device from an existing network by an initiator (of another network) through Touchlink.

### 7.7.1.6 Joining an End Device to the Network

After the discovery and identify phases, if the initiator is already part of a network and the target is an FN Controller device then the initiator sends a 'Join as End Device' command to the target. However, it is not possible to add a NFN Controller device to any network.

This procedure allows Controller-to-Controller device commissioning. In such a case, the newly joined Controller device needs to acquire the network database, containing information such as Endpoint IDs, Endpoint List and Group IDs, from the initiator by issuing Commissioning Utility commands.

### 7.7.1.7 Network ID Update

As part of the processing of the Touchlink scan responses, for any responders that are in the same network as the initiator the returned Network Update ID will be examined. If these IDs are not identical, one of two actions will be taken:

- If the Network Update ID of the target is older than that of the initiator, the initiator will send the target a Network Update Request command containing the logical channel, Pan ID and Network Update ID. The target can then update its network parameters.
- If the Network Update ID of the initiator is older than that of the target, the initiator will update its channel, PAN ID and Network Update ID from the details in the scan response. Then in the case of an End Device, Touchlink will be ended and a ZigBee Network Rejoin initiated.

### 7.7.2 Classical Joining

A ZLL device can join a ZLL or non-ZLL (ZHA) network by the classical commissioning method. In this method, the device joins the network through MAC association and obtains the network key that is encrypted with the pre-configured link key. For more information, refer to the ZLL Specification.

In this demonstration, the Lighting device (if on the DR1175 hardware) enables 'Permit Join' for 60 seconds when the button labelled '**DIO8**' is pressed on the Carrier Board.

Alternatively, the remote control unit can be used to broadcast a Management Permit Join command to the network. During this time, other devices can request MAC associations to join.

If joining a ZLL network, the Transport Key command uses the ZLL key to encrypt the network key. If joining a ZHA network then the ZHA key is used to encrypt the network key.

### 7.7.3 Network Address Assignment

Network address assignment in a ZLL network is done by an 'address assignment capable device' as an incremental address starting from the minimum network address of 0x0001. When an initiator forms a network, the application specifies an address range for the network in terms of minimum and maximum network addresses.

- When a device that is not 'address assignment capable' is added to the network, the initiator issues a network address to the target.
- When a device that is 'address assignment capable' is added to the network, it receives its address from the initiator. It also receives a range of addresses for it to use later when Touchlinking devices onto the network

If a device cannot be assigned a network address, it cannot join the network.

If a device is added as part of the classical join method, the network address is not assigned by the initiator and will be a random address. The device must then be 'touchlinked' to the initiator, so that the Controller device can update its End Device table with the address, device ID and endpoint ID of the new Lighting device.

### 7.7.4 Group Address Assignment

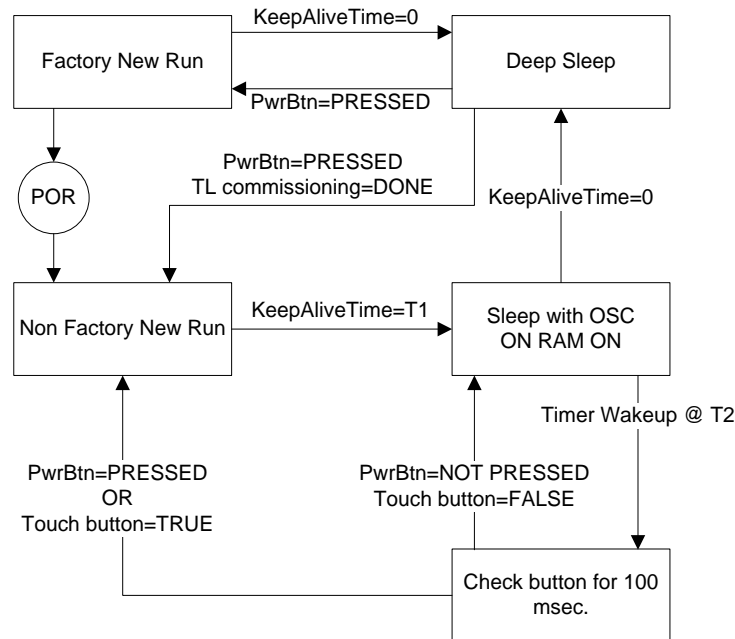
The number of group IDs is specified by an application in the device information table. The total number of groups required by a target is indicated in the network response. The target receives its free group identifier range during commissioning. A network initiator that is 'network address assignment capable' will also be a 'group address assignment capable' device. The group address can range from 0x0001 to 0xFF00. The default group 0x0000 is used by the Scenes cluster.

When a Controller-to-Controller touchlink exchange occurs, the new Controller device that is added to the network must send out Commissioning Utility commands to obtain information such as an endpoint list and group ID list, to allow the initiator to control the commissioned devices in the network.

## 7.8 Sleep Wake-up Cycle for Remote Control Unit

The Remote Control Unit in a ZLL network can typically be a Sleeping End Device, in order to conserve battery power. This application demonstrates sleep in two modes – Sleep with RAM ON and Deep Sleep.

The diagram below illustrates the sleep state machine.



**Figure 3: Sleep state machine**

The following sub-sections describe the different states.

### 7.8.1 Power On Reset

A 'Power On Reset' can be performed on either a Factory New or Non-Factory New device. The *KeepAliveTime* value is loaded with T seconds.

### 7.8.2 Factory New Run

In the 'Factory New Run' state, the *KeepAliveTime* value will decrement and then enter into Deep Sleep mode when there is no touch activity on the keypad. Pressing the wake-up button will cause the unit to wake and continue to run.

### 7.8.3 Non Factory New Run

In the 'Non-Factory New Run' state, the Remote Control Unit will be reset the timer while there is activity from touch-button presses on the keypad. Otherwise, the *KeepAliveTime* value is decremented. When there is no activity for *T1* seconds, the unit enters the 'Sleep with RAM ON' state.

### 7.8.4 Sleep With RAM ON

In the 'Sleep with RAM ON' state, a wake timer is configured to wake up the device every second. The Remote Control Unit checks for touch-button activity in order to come out of sleep and set the *KeepAliveTime* value to T seconds. On a touch-button press, the device enters into OS restart mode, continues with the network and keeps polling.

If there is no button press, the device keeps decrementing the *KeepAliveTime* value and enters Deep Sleep mode when *KeepAliveTime* reaches zero.

### 7.8.5 Deep Sleep

When in Deep Sleep mode, the unit can be woken by pressing the wake-up button. The unit then rejoins the network.

### 7.8.6 PollTask

PollTask is a task that performs the following:

- Runs at the rate of once per second and issues a poll request.
- Decrements *KeepAliveTime*
- When *KeepAliveTime* expires, the device stops all the timers using the function **vStopAllTimers()**, re-initialises the power manager for Deep Sleep mode and eventually enters into Deep Sleep mode.
- Declares the *SleepWithRamOn* as TRUE, so that the **PreSleep()** callback will be able to configure the appropriate wake-up signal.

### 7.8.7 PreSleep Callback

The **PreSleep()** callback function is entered when all activities have been shut down. The function does the following.

- Saves the MAC settings
- If *SleepWithRamOn* is TRUE, schedules the wake-up input as a 1-second wake timer.

### 7.8.8 WakeUp Callback

If *SleepWithRamOn* is TRUE, the **WakeUp()** callback function restores the MAC settings, wakes up the touch-button interface, restarts the OS and activates APP\_ScanInputTask.

### 7.8.9 APP\_ScanInputTask

This task runs every 10 ms for 10 cycles and scans the touch-button interface for any input. If an appropriate input is detected, it activates PollTask and the button scan timer task. In the case of no input detected, the task returns the unit to sleep by stopping all the timers.

## 7.9 Guidelines for Modifying the Remote

This section highlights the key areas of interest within the code, in case the developer wishes to alter the Remote Control Unit's functional states or switch to a different user interface.

### 7.9.1 Operational State Machine

The operational state machine (**sDeviceDesc.eNodeState**) is located within **zll\_remote\_node.c**. Additional states must be added to this switch statement if further operational modes are required.



## 7.9.2 Handling a Key Press and Release

The function that handles a key press and release is located in **zll\_remote\_node.c** as part of APP\_ZLL\_RemoteTask. The events related to key input are processed as in the task with the sAppEvent.eType event. Any changes to the key handling should be made within the corresponding switch statement. The file also contains the key press handler function, **APP\_vHandleKeyPress()**. Any alteration to the key map to allow different functionality should go in this function.

Similarly, the **APP\_vHandleKeyRelease()** function can be altered to add a function that is called upon release of a key.

## 7.10 Guidelines for Modifying the Light Identify Effects

This section highlights the key areas of interest within the code, in case the developer wishes to alter the identify effect on the Lighting device.

The file **app\_zcl\_light\_task.c** contains an endpoint callback function **APP\_ZCL\_cbEndpointCallback()** in which the identify and trigger effect commands are translated into an effect that a Lighting device can use to visually identify itself. The RGB components and levels for identification can be set here to achieve a different effect, if desired.

Similarly, in the function **vldEffectTick()**, the level step for the effect can be set to the desired value.

## 8 Release Details

### 8.1 New Features

ID	Feature	Description
<b>Version 1.16</b>		
<b>Version 1.15</b>		
<b>Version 1.14</b>		
lpsw8157	Lights: Allow scans for classic join while waiting for Touchlink	At power on the lights scan all channels attempting to classically join (unchanged). If a classic join does not complete the lights listen for Touchlink commands on channel 11 (previously a random primary channel was chosen). Every 1.25s whilst waiting to be Touchlinked a single channel is scanned to attempt classic joining resulting in each channel being scanned once every 20s (new).  The previous algorithm can be restored by building with SCAN_IN_TOUCHLINK defined to TRUE in zpr_light_node.c.
<b>Version 1.13</b>		
<b>Version 1.12</b>		
<b>Version 1.11</b>		
<b>Version 1.10</b>		
N/A	OTA download of same image	Same version of an OTA image can now be downloaded from the OTA server to a client.

### 8.2 Known Issues

There are no known issues in this release.

### 8.3 Bug Fixes

ID	Description
<b>Version 1.16</b>	
<b>Version 1.15</b>	
<b>Version 1.14</b>	
lpsw8158	Extended Color Light: Revert endpoints back to standard values.  In Extended Color Light the endpoints have been altered to match the other light applications: Light endpoint is 1, Commission endpoint is 2.
<b>Version 1.13</b>	
lpsw7797	CCLD_SCENES_SUPPORT_ZLL_ENHANCED_COMMANDS name corrected.
lpsw7789	Stack and heap sizes defined by SDK. LD files have been removed from application.
lpsw7788	eOTA_UpdateClientAttributes() function updated to include image stamp value.
<b>Version 1.12</b>	
lpsw7777	Deleted scenes on a light node were not properly removed and were reinstated following a power-cycle (failed ZigBee certification test 3.13).
<b>Version 1.11</b>	
<b>Version 1.10</b>	
lpsw7320	OTA needs to take into account the remapping of Flash by the bootloader.
lpsw7444	Some ZLL bulbs identify as OTA servers as well as clients. This has been corrected such that OTA clients on bulbs do not advertise as OTA servers.
lpsw7445	Touchlink handling of scan response fails if number of endpoints is not 1.

lpsw7448	OTA images did not include any chip type in the headers, so a device built for JN5168 could accept an image built for JN5169, or vice versa. This has been corrected such that OTA clients only accept images for the correct chip type.
----------	--

## Appendix A - Source File Descriptions

### Automatically Generated Files

Each device has several files that are automatically generated at build-time from the JenOS and ZPS configuration diagrams. These files are not generally used by the developer but are located in the respective device's **\Source** folders, in case they are of interest.

### Common Files

A number of common files are used across all device types and are located within the **\Common\Source** folder. The following table gives a brief description of each of the common files.

Filename	Description
app_common.h	Contains the common macro definitions used in the project. It contains the conditional inclusion of the appropriate device headers.
app_events.h	Contains global definitions for the events in the ZLL application.
ecb_decrypt.c	Contains the AES128 decryption functions.
eventStrings.c/h	Contains strings for the ZPS and application events.
os_msg_types.h	Contains all the include files required.
PDM_ID.h	Contains the PDM_IDs that are used in PDM load and restore.
app.zpscfg	This is the ZPS configuration diagram, which is used to configure generic network and node parameters. This includes profile, cluster, endpoint and RF channel configurations. For more information, refer to the "ZPS Configuration Editor" chapter of the <i>ZigBee PRO Stack User Guide (JN-UG-3048)</i> .

### Common Controller Files

The following table gives a brief description of the common files used by all the Controller devices, and are located in the **\Common\_Controller\Source** folder.

Filename	Description
app_captouch_buttons.c/h	Capacitive-touch remote button task and event handler. It also defines the DIO ISR that is used for capacitive touch.
app_remote_commission_task.c	Commissioning task and state machine handler.
app_start_remote.c	Start-up module with vAppMain and sleep wake-up callbacks. All the initialisation for start-up and wake-up is available in this module.
app_zcl_remote_task.c/h	ZCL event handler for the node - the endpoint callback function is part of this module. This also has the node task and functional state machine.
DriverCapTouch.c/h	Capacitive-touch driver module.
zll_remote_node.c/h	Contains the ZLL application command send/receive functions - key map for the Remote Control Unit. It also handles the initialisation of ZLL states by calling the appropriate PDM store and retrieve descriptors.
App_ZLL_Remote_JN516x.oscf gdiag	This is the JenOS configuration diagram, which is used to configure certain application building blocks, such as pre-emptive tasks, software timers, mutexes and Interrupt Service Routines. For more information, refer to the <i>JenOS User Guide (JN-UG-3075)</i> .

## Common Light Source Files

The following table gives a brief description of the common files used by all the Lighting devices, and are located in the **\Common\_Light\Source** folder.

Filename	Description
app_buttons.c/h	Contains application buttons mask and ISR routines.
app_light_commission_task.c	Contains commissioning task and state machine handle for the Lighting devices.
app_start_light.c	Start-up module with vAppMain and sleep wake-up callback functions. All the initialisation for start up and wake up is available in this module.
app_zcl_light_task.c/h	ZCL event handler for the node - the endpoint callback function is part of this module. This also has the note task and functional state machine.
zpr_light_node.c/h	Contains ZLL application command send/receive functions. It also handles initialisation of ZLL states by calling appropriate PDM store and retrieve descriptors.
App_ZLL_Light_JN516x.oscfg diag	This is the JenOS configuration diagram, which is used to configure certain application building blocks, such as pre-emptive tasks, software timers, mutexes and Interrupt Service Routines. For more information, refer to the <i>JenOS User Guide (JN-UG-3075)</i> .

## Device-specific Source Files

The following table gives a brief description of the device-specific source files, located in the **\Controller\_<DeviceType>\Source** or **\Light\_<DeviceType>\Source** folder.

Filename	Description
App_Controller_<DeviceType>.c/h	Controller device-specific module that has the device definition and registration functions for each device type. It also handles device-specific initialisation.
App_Light_<DeviceType>.c/h	Lighting device-specific module that has the device definition and registration functions for each device type. It also handles device-specific initialisation.

## Appendix B – Pre-processing Macros Description

Compile-time macros to manipulate ZigBee Cluster Library functionality are defined in the **zcl\_options.h** file for the respective device. Other than these ZCL-specific macros, the demonstration application uses the following macros for ease of development and testing.

Macro	Description
TEST_RANGE	Enables code for testing low and high ranges during development
ZLL_PRIMARY	Uses primary channels 11,15, 20 and 25
ZLL_PRIMARY_PLUS3	Uses primary channel set, offset by 3, for use during development.
FIX_CHANNEL	Allows the application to use fixed channels, as specified in the channel mask.
RAND_KEY	A random network key is generated when the RAND_KEY macro is defined as TRUE. A fixed network key of 0XXXXXXXXXXXXXXXXXXXXXXXXXXXX is generated when the RAND_KEY macro is defined as FALSE. The fixed key is useful during the development/verification stage.
SHOW_KEY	Allows application to print the keys to the debug terminal to aid debugging.
HALT_ON_EXCEPTION	Stops execution in the case of an exception. Otherwise, allows the application to continue after a reset following an exception.
OS_STRICT_CHECKS	OS strict check for task handlers.
SLEEP_ENABLE	Enables sleep for the Remote Control Unit.
NEVER_DEEP_SLEEP	If set to TRUE, the Remote Control Unit will never enter Deep Sleep. It will, however, sleep with memory held and wake at one second intervals.
CLASSIC_JOIN	Allows a classic join by pressing the + button on a Remote Control Unit in the FN state.
BUTTON_MAP_DR1175	Button mapping for DR1175 (Lighting/Sensor Expansion Board).

## Appendix C - Build File Descriptions

### Common Build Files

The following table gives a brief description of the build files, located in the **\Common\_<Light>or<Controller>\Build** folder.

Filename	Description
Makefile	This common Makefile is used to build the binary for the specific ZLL device based on input flags (e.g., make REMOTE=Controller_ColorSceneController -f Makefile)

### Device-specific Linker Files

The following table gives a brief description of the device-specific linker files, located in the **\Controller\_<DeviceType>\Build\** or **\Light\_<DeviceType>\Build\** folder.

Filename	Description
APP_stack_size_JN5168.ld	Linker command file defining the default application stack size. Can adjust _stack_size for the desired stack size.

## Appendix D - Known Issues

ID	Severity	Description
-	-	-

## Appendix E - Application Code Size Statistics

The demonstration application of this Application Note has the following memory footprint, using the JN516x ZLL SDK (JN-SW-4168).

Components	Chip	Text Size (In Bytes)	Data Size (In Bytes)	BSS Size (In Bytes)
<b>Controller_NonColorController</b>	JN5168	121790	1896	20917
<b>Controller_ColorController</b>	JN5168	123190	1912	21313
<b>Controller_NonColorSceneController</b>	JN5168	122734	1912	21045
<b>Controller_ColorSceneController</b>	JN5168	125474	1928	21557
<b>Controller_OnOffSensor</b>	JN5168	124142	1928	21349
<b>Light_OnOffLight</b>	JN5169	138882	1976	22321
<b>Light_OnOffPlug</b>	JN5169	138826	1976	22305
<b>Light_DimmableLight</b>	JN5168	138938	2040	22801
	JN5169	139438	2048	22825
<b>Light_DimmablePlug</b>	JN5169	139462	2048	22825
<b>Light_ColorLight</b>	JN5168	158348	2416	22665
	JN5169	158789	2412	22685
<b>Light_ExtendedColorLight</b>	JN5168	159688	2416	23097
	JN5169	160200	2416	23097
<b>Light_ColorTemperatureLight</b>	JN5168	151686	2416	22649
	JN5169	152158	2416	22665



## Revision History

Version	Notes
1.0	First release
1.1	General updates and improvements, new functionality added to the remote control unit
1.2	Note about master key added and application build sizes updated
1.3-1.5	General updates and improvements made. Shift LED issue fixed
1.6	Default interpolation points set to stop bulbs flashing when switched off following first use
1.7	Updated for new SDK and 'BeyondStudio for NXP' toolchain, added OTA Upgrade support and other minor updates/corrections made
1.8	Added support for JN5169 device
1.9	Binaries rebuilt on JN-SW-4168 SDK v1279 for improved radio settings
1.10	Updated with the new features and bug fixes detailed in Section 8. Binaries rebuilt on JN-SW-4168 SDK v1455.
1.11	Binaries rebuilt on JN-SW-4168 SDK v1461.
1.12	Updated with bug fix detailed in Section 8. Binaries rebuilt on JN-SW-4168 SDK v1470.
1.13	Updated with bug fixes detailed in Section 8.
1.14	Updated with bug fixes detailed in section 8. Binaries rebuilt on JN-SW-4168 SDK v1595
1.15	Updated with bug fixes detailed in section 8. Binaries rebuilt on JN-SW-4168 SDK v1611
1.16	Updated with bug fixes detailed in section 8. Binaries rebuilt on JN-SW-4168 SDK v1620

## 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](http://www.nxp.com)