AN1257

Zigbee 3.0 Developing Clusters

Rev. 2.8 — 12 December 2023 **Application note**

Document information

Information	Content
Keywords	DK006 K32W148EVK
Abstract	This Application Note applies to the JN5189, JN5188, K32W061,K32W041,K32W041A and K32W041AM Zigbee 3.0 wireless microcontrollers used with the DK006 (Development Kit) platform, it also applies to the K32W1480 wireless microcontroller used with the K32W148EVK (Development Kit) platform. These microcontrollers will be referred to as the DK006 microcontrollers throughout this document. The Zigbee 3.0 Getting Started Application Note [JN-AN-1260] contains instructions for installing MCUXpresso, DK006 microcontroller SDKs and other required tools to develop with this Application Note. This Application Note describes how to develop a Zigbee 3.0 Window Covering Device using the JN-AN-1243 Base Device Template Router Device application as a starting point. This Application Note can be used in two ways: 1. As a starting point for creating a Window Covering device using the functional example created in the final step. 2. As a guide to creating devices and clusters not included in the NXP ZCL implementation including manufacturer-specific devices and cluster. The Window Covering device described in this Application Note is based on Zigbee device types from the Zigbee Closures and Zigbee Cluster Library Specifications. It 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 Window Covering device on the NXP DK006/K32W148EVK platform with an emphasis on how the Router template was adapted to create the Window Covering device including the creation of the Window Covering Zigbee device and cluster software.

- **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 your DK006 microcontroller:

- MCUXpresso IDE
- JN518x Zigbee 3.0 SDK
- K32W061/K32W041A/K32W041AM Zigbee 3.0/Bluetooth SDK
- K32W148 SDK

The MCUXpresso software and installation instructions are described in the *Zigbee 3.0 Getting Started Application Note [JN-AN-1260]*. The *JN-AN-1257 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 the *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 kits provide a platform for running these applications:

- JN518x-DK006 Development Kit, which features JN5189 devices
- K32W148EVK, which features K32W148 devices
- IoT_ZTB-DK006 Development Kit, which features K32W061 devices

This kit supports the NFC commissioning of network nodes (see Joining an Existing Network using NFC for details).

3 Application Note Overview

The example applications provided in this Application Note are listed in the following table.

Table 1. Example Applications and Device Types

Application	Device Type
WindowCovering	Router

For each application, source files and pre-built binary files are provided in the Application Note ZIP package. This Application Note divides the development of the Window Covering device into a number of steps. The source code and binaries for each step are in separate folders to allow easy viewing of the changes made in each step using a text difference application.

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 learn about the development of the Window Covering device from the Router template, refer to Developing the Window Covering.
- For an overview of the source files not covered in Developing the Window Covering and to aid in developing the application further, 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.

DK006 Development Kits for the development of Zigbee 3.0 applications provide the possibility of network commissioning through Near Field Communication (NFC).

4 Running the Demonstration Application

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

This section describes the operation of the finished Window Covering device at the end of Step 3 – Hardware Control. The source code for this step is located in the **3-WindowCovering** folder of the Application Note (with additional source code in the **3-Common** folder). The pre-built binary is located in the **Binaries/3-Window Covering_NtagNwk_NtagOta_GpProxy_Ota_OM15082** folder.

The basic functionality of the Window Covering device is identical to the functionality of the Router in the *Zigbee 3.0 Base Device Application Note [JN-AN-1243]* upon which it is based. The section Developing the Window Covering works through the development of the Window Covering device from the template Router.

4.1 Loading the Applications

The table below lists the application binary files, created in step 3, 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.

Table 2. Application Binaries and Hardware Components

Hardware	Binary Files
OM15076-3 Carrier Board OM15082-2 Generic Expansion Board	**WindowCovering_GpProxy_OM15082 WindowCovering_NtagNwk_NtagOta_ GpProxy_Ota_OM15082_V1**
K32W148EVK OM15082-2 Generic Expansion Board	**WindowCovering_GpProxy_Ota_K32W148EVK_V1**

A binary file can be loaded into the Flash memory of a DK006 device using the *DK6 Production Flash Programmer [JN-SW-4407]*. 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]*. For K32W148, the batch file requires the installation of JLINK, see *Zigbee 3.0 Getting Started Application Note [JN-AN-1260]*.

4.2 Window Covering Functionality

The functionality of the Window Covering application is described and illustrated below.

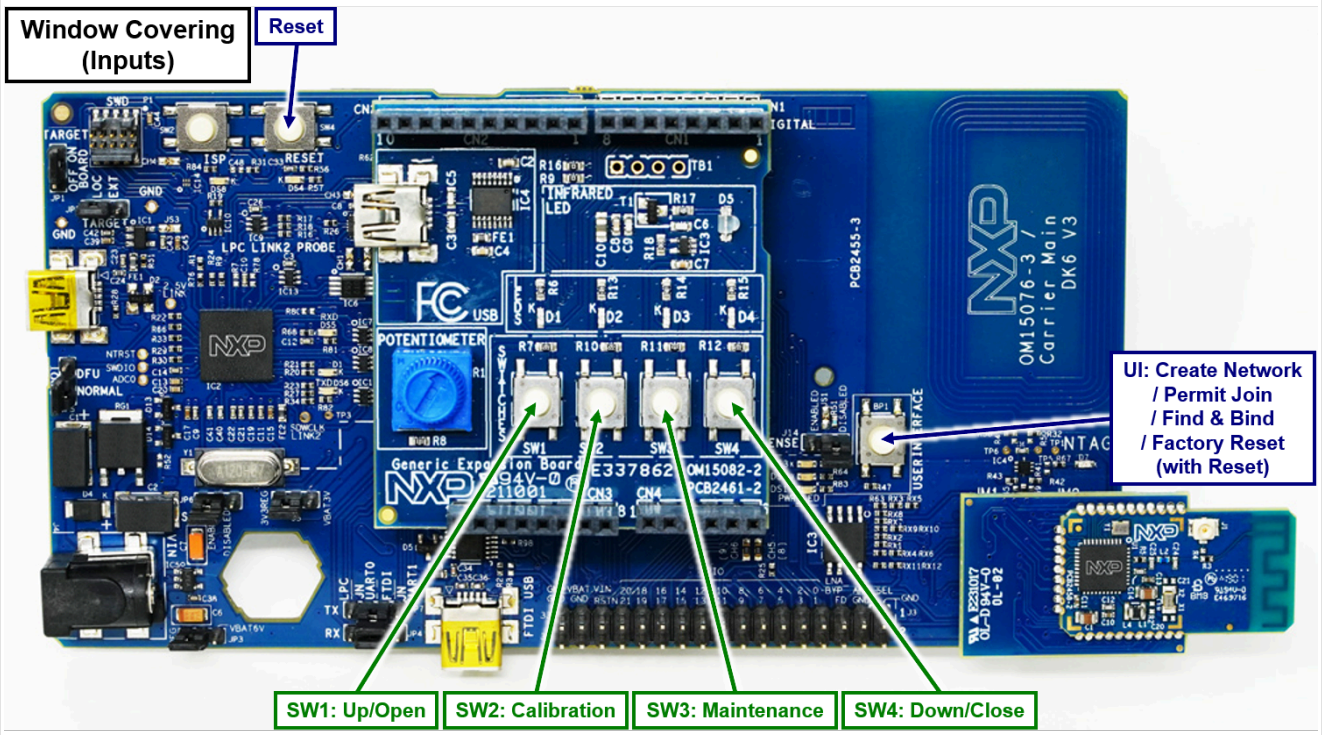

Figure 1. Window Covering Board

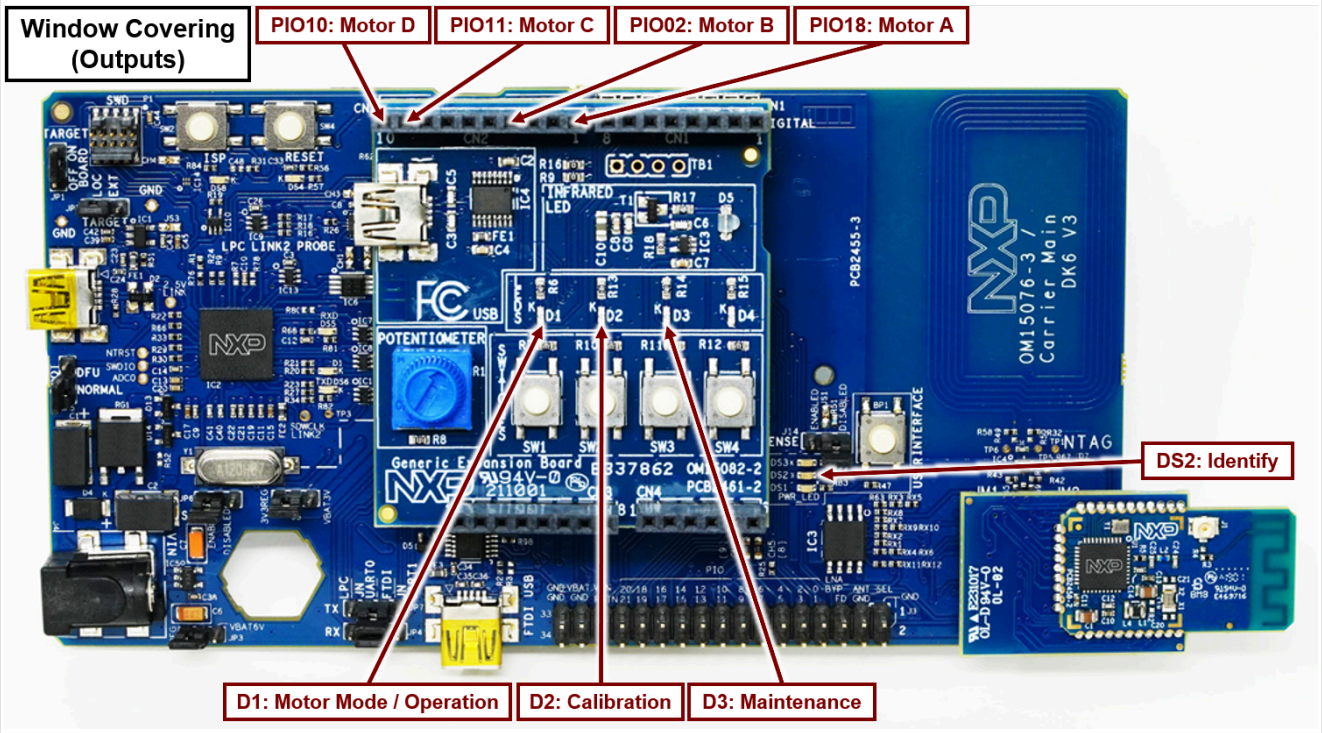

Figure 2. Window Covering Board

The Window Covering operates as a Zigbee Router and can either join an existing network or decide to form a distributed network itself for other nodes to join. For details of the differences between a Centralised Trust Centre Network and a Distributed Network, refer to the *Zigbee Devices User Guide [JN-UG-3114]*. The Window Covering supports the mandatory clusters and features of a Window Covering device and also many optional attributes, commands and features.

This Window Covering implementation operates as a Rollershade Window Covering Type with Lift control only (so Tilt operations are not included but the necessary device and cluster code can be enabled for other Window Covering Types). The lift control operates in open-loop mode where the Zigbee specification states that certain features are only for closed-loop mode they have been implemented as if the Window Covering device were operating in closed-loop mode in order to provide a more fully-featured example application.

The table below shows the clusters implemented in the Window Covering Device. Note the the OTA Upgrade client cluster is only included in the **WindowCovering_NtagNwk_NtagOta_GpProxy_Ota_OM15082_V1.bin** build:

Table 3. Clusters

Server (Input) Side	Client (Output) Side
Mandatory	
Basic	*None*
Identify	
Groups	
Scenes	
Window Covering	
Optional	
None	OTA Upgrade

The following table shows the Window Covering Cluster attributes. The greyed attributes are available for use in the cluster and device code but are disabled in this implementation (mostly because Tilt functionality is not included but some others are optional).

Attributes marked M* are only mandatory if the Lift or Tilt operation is included in the device in closed-loop mode (bits in the ConfigStatus attribute indicate this). Even through this Window Covering device operates in open-loop mode the M* attributes have been included for the Lift operations.

The Scenes functionality uses the actual lift position percentage despite operating in open-loop mode.

The Mode attribute Zigbee default (shown in the table below) sets the device into maintenance mode where the motor is not allowed to operate, and any LEDs are disabled. This implementation instead uses a default value of 0b00001000 allowing the motor to run and any LEDs are enabled (for ease of demonstration).

Table 4. Attributes

ID	Name	Type	Access	Default	M/O
0x0000	WindowCoveringType	enum8	Read	0x00	M
0x0001	PhysicalClosedLimitLift	uint16	Read	0x0000	O
0x0002	PhysicalClosedLimitTilt	uint16	Read	0x0000	O
0x0003	CurrentPositionLift	uint16	Read	0x0000	O
0x0004	CurrentPositionTilt	uint16	Read	0x0000	O
0x0005	NumberOfActuationsLift	uint16	Read	0x0000	O
0x0006	NumberOfActuationsTilt	uint16	Read	0x0000	O

Table 4. Attributes*...continued*

ID	Name	Type	Access	Default	M/O
0x0007	ConfigStatus	map8	Read	0b00000011	M
0x0008	CurrentPositionLiftPercentage	uint8	Read / Scene / Report	0x00	M*
0x0009	CurrentPositionTiltPercentage	uint8	Read / Scene / Report	0x00	M*
0x0010	InstalledOpenLimitLift	uint16	Read	0x0000	M*
0x0011	InstalledClosedLimitLift	uint16	Read	0xffff	M*
0x0012	InstalledOpenLimitTilt	uint16	Read	0x0000	M*
0x0013	InstalledClosedLimitTilt	uint16	Read	0xffff	M*
0x0014	VelocityLift	uint16	Read / Write	0x0000	O
0x0015	AccelerationTimeLift	uint16	Read / Write	0x0000	O
0x0016	DecelerationTimeLift	uint16	Read / Write	0x0000	O
0x0017	Mode	map8	Read / Write	0b00000100	M
0x0018	IntermediateSetpointsLift	octstr	Read / Write	“1,0x0000”	O
0x0019	IntermediateSetpointsTilt	octstr	Read / Write	“1,0x0000”	O

The following table shows the Window Covering Cluster commands. The greyed commands are available for use in the cluster and device code but are disabled in this implementation (because Tilt functionality is not included). The included optional commands have been included despite Lift control using open-loop mode.

Table 5. Cluster Commands

ID	Description	M/O
0x00	Up / Open	M
0x01	Down / Close	M
0x02	Stop	M
0x04	Go To Lift Value	O
0x05	Go To Lift Percentage	O
0x07	Go To Tilt Value	O
0x08	Go To Tilt Percentage	O

4.2.1 Joining a Network

4.2.1.1 Joining an Existing Network using Network Steering

A factory-new Window Covering can join an existing network once the network has been opened to accept new joiners (Network Steering for a device on a network). This is achieved as follows:

1. Trigger Network Steering on one of the devices already on the network.
2. Then reset (using the RESET button) or power-on the Window Covering device.

This will cause the Window Covering to start a network discovery and the associate process. Association is followed by an exchange of security materials and an update of the Trust Centre Link Key (if joining a Centralised Trust Centre Network).

If the join is unsuccessful, it can be retried by power-cycling again.

4.2.1.2 Joining an Existing Network using NFC

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

- The *NciIcode* build of the Zigbee IoT Control Bridge, described in the Application Note *Zigbee 3.0 IoT Control Bridge [JN-AN-1247],* this is recommended for use with the Window Covering device as the other options do not provide any functionality to control the Window Covering device.
- The *NciIcode* build of the Coordinator template, described above in the Application Note *Zigbee 3.0 Zigbee 3.0 Base Device [JN-AN-1243]*
- 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.

Instructions for this process are included in the above Application Notes (JN-AN-1247, JN-AN-1243 or JN-AN-1222).

4.2.2 Allowing Other Devices to Join the Network

Once the Window Covering is part of a network, the network must be opened to allow other devices to join (Network Steering while on a network). To do this:

- Press the USER INTERFACE button on the Carrier Board (the same button is also used to start Finding and Binding, described in Binding Devices).

The Router will then broadcast a Management Permit Join Request to the network to open the ‘permit join’ window for 180 seconds. The Network Steering process (for devices not on a network) can now be triggered on the devices that are to join the network.

4.2.3 Binding Devices

The Window Covering device supports the Window Covering cluster as a server and implements the Finding and Binding process as a target. To trigger Finding and Binding as a target, do the following:

1. Press the USER INTERFACE button on the Carrier Boards of all the target devices (the same button also is also used to start Network Steering, described in Allowing Other Devices to Join the Network).

2. Start Finding and Binding on the Initiator device.

This will cause the Window Covering to self-identify for 180 seconds, while the Initiator will try to find the identifying devices, query their capabilities and create bindings on those with matching operational clusters. As part of this process, the Window Covering may receive an Add Group Command and/or a Binding Request Command.

Reporting is a mandatory feature in Zigbee 3.0. The Window Covering device supports the Window Covering cluster as a server and the Current Position Lift Percentage and Current Position Tilt Percentage attributes of this cluster are reportable attributes as defined in the Zigbee Base Device Behaviour Specification. The Window Covering holds a default configuration for reporting the state of the Current Position Lift Percentage attribute. Once a device wishing to receive these periodic and on-change reports has created a remote binding, the Router will start to send reports to this bound device. The frequency of the reports depends on the default report configuration of the individual target device, 61 seconds in this case or whenever a change occurs. The device receiving the reports can request that this is changed by sending a Report Configuration command.

4.2.4 Operating the Device

The operational functionality of this device in this demonstration is provided by the Window Covering cluster. Since the device supports the Window Covering cluster server, its operation is passive and it responds to

Window Covering cluster commands sent by other devices. It responds to these commands by operating the four motor control digital output lines to drive a 4-phase stepper motor.

The Window Covering device will also respond appropriately to writes to the Mode attribute used to set specific operating modes.

The Window Covering device saves the current position to Flash memory 5 seconds after it stops moving to reduce wear on the Flash memory. If the Window Covering is power-cycled during this time the current position of the device may become unsynchronised. Changes to the Mode attribute are saved immediately.

As there is no Window Covering client device (such as a Window Covering Controller device) implemented the JN-AN-1247 Zigbee 3.0 IoT Control Bridge can be used with the included ZGWUI to exercise these features as follows.

4.2.4.1 ZGWUI Raw Data Commands

As the ZGWUI does not include a Window Covering Client Cluster interface commands for the Window Covering cluster should be issued from the General tab using the Raw Data button. These commands all share a common set of parameters shown below (all values are entered in hexadecimal). The image first shows the unedited fields followed by fields with appropriate values, in the ZGWUI application all the controls are on a single line:

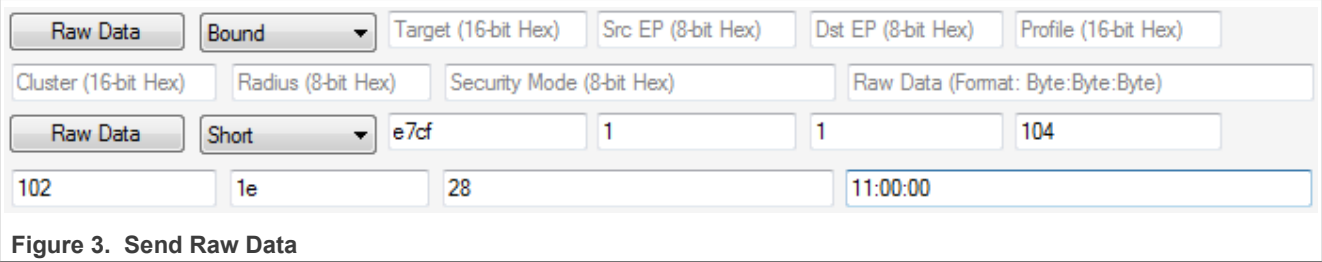

Figure 3. Send Raw Data

The parameters are as follows:

- Addressing Mode (Short) – the drop down allows the addressing mode to be selected. Short (16-bit) unicast addressing is shown above the other available modes may also be used including Group mode if the Window Covering is placed into a group.
- Address (e7cf) – this is the unicast or group address to send the command to.
- Src EP (1) – this is the source endpoint for the sending device. The Control Bridge uses 1 as its application endpoint.
- Dst EP (1) – this is the destination endpoint for the receiving device. The Window Covering uses 1 as its application endpoint.
- Profile (104) – this is the Zigbee Profile ID the command is addressed to. 0x104 is used for both Zigbee Home Automation and Zigbee 3.0.
- Cluster (102) – this is the Zigbee Cluster ID the command is addressed to. 0x102 is used for the Window Covering cluster.
- Radius (1e) – this is the maximum number of hops through the network the packet can take. The commonly used value of 0x1e is recommended.
- Security Mode (28) – these are the Network Security Control bits. The commonly used value of 0x28 is recommended which uses the network key with an extended nonce.
- Raw Data (11:00:00) – this is the raw data to be transmitted as a series of bytes. The bytes are as follows:
  - Byte 0 (11) – these are the Frame Control bits the commonly used value of 0x11 is used here to indicate the command is cluster specific and disable the default response.
  - Byte 1 (00) – is the Transaction Sequence Number this can be set to any value and is used to match responses to requests.
  - Byte 2 (00) – is the Command ID the value of 0x00 used here is the Up/Open command.

– Bytes 3 onwards (not shown) – any additional bytes required by the command payload can be added here.

4.2.4.2 Up/Open, Down/Close and Stop Commands

These basic commands are grouped together and do not have a payload. Enter the following in the Raw Data edit-box for each command (the last byte is the Command ID):

- Up/Open – “11:00:00”
- Down / Close – “11:00:01”
- Stop – “11:00:02”

4.2.4.3 Go To Lift Percentage Command

This command uses Command ID 0x05 and includes a 1-byte parameter specifying the lift percentage to move to (the value 0x32 shown below will move the Window Covering to 50%):

- Go To Lift Percentage (50%) – “11:00:05:32”

The percentage is between the Installed Limit attribute values when in normal mode and between the Physical Limit attribute values when in calibration mode.

An Invalid Value response will be returned for any values greater than 100%.

4.2.4.4 Go to Lift Value Command

This command uses the Command ID 0x04 and includes a 2-byte parameter specifying the lift value to move to using little-endian order (the value 0x0032 shown below, in little-endian order, will move to a position of 50cm):

- Go To Lift Value (50cm) – “11:00:04:32:00”

The position is measured from the physical open position.

When in normal mode an Invalid Value response will be returned to any values not within the Installed Limit attribute values. When in calibration mode an Invalid Value response will be returned for any values not within the Physical Limit attribute values.

4.2.4.5 Reading Attributes

The values of Window Covering attributes can be read using the Read Attrib button on the ZGWUI. The example below illustrates reading the Current Position Lift attribute (ID 0x0003) other attributes can be read by using their Attribute IDs:

Figure 4. Reading Attributes

As the ZGWUI does not contain a Window Covering client it does not display the values of the read attributes but these can be observed on-air using sniffer software such as Ubiqua.

4.2.4.6 Writing Attributes

The values of the Window Covering attributes can be written using the Write Attrib button on the ZGWUI. The example below illustrates writing the Mode attribute (ID 0x0017) which is an 8-bit bitmap (Type 0x18) with a value of 0x0a (which enables calibration mode and LED display):

Figure 5. Writing Attributes

The bits of the Mode attribute are as follows:

Table 6. Mode bits

Bit	Meaning	Description
0	0 = motor direction is normal 1= motor direction is reversed	Disables (0) or Enables (1) the reversal of the motor rotating direction associated with an Up/Open command. Should be set to that an Up/Open command matches moving the Window Covering physically in that direction.
1	0 = run in normal mode 1 = run in calibration mode	Disables (0) or Enables (1) placing the Window Covering into Calibration Mode where the limits are either setup using physical tools or limits are learned by the controller based on physical setup or the Window Covering by an installer.
2	0 = motor is running normally 1 = motor is running in maintenance mode	Disables (0) or Enables (1) placing the motor into Maintenance Mode where the motor cannot be moved over the network or by a switch connected to a Local Switch Input.
3	0 = LEDs are off 1 = LEDs will display feedback	Disables (0) or Enables (1) the display of any feedback LEDs resident especially on the packaging of an endpoint where they may cause distraction to the occupant.

4.2.4.7 Group Control

The Window Covering may be placed into a Group using the Add Group button on the Group Cluster tab to allow control of a number of devices using Group addressing. The example below illustrates placing the Window Covering into Group ID 0x1111:

Figure 6. Group Control

4.2.4.8 Scene Control

The Window Covering may be placed into a Scene and when the Scene is recalled the Window Covering will move to the position set or saved for that Scene. Scenes are associated with a Group so the Window Covering must be placed into a Group before configuring Scenes. The Scene commands are located on the Scenes Cluster tab of the ZGWUI.

The example below illustrates storing the current position of the Window Covering to Scene ID 0x1111 with Group ID 0x11 unicasting the Store Scene command to the Window Covering's short address:

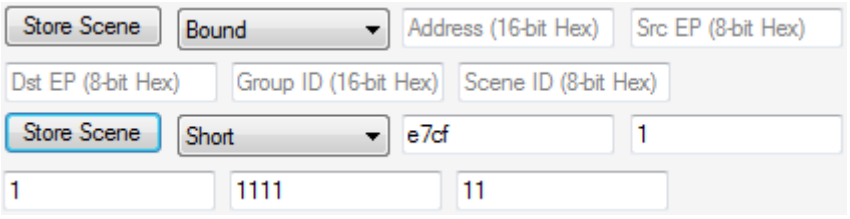

Figure 7. Store Scene

The example below illustrates recall the previously saved scene, group-casting the command to Group ID 0x1111:

Figure 8. Recall Scene

4.2.4.9 Local Control

The switches on the OM15082 Generic Expansion Board provide local control over the Window Covering as follows:

Table 7. Switch Operations

Switch	Operation	Description
SW1	Press	Move Up/Open
	Release	Stop
SW2	Press	Toggle calibration mode
SW3	Press	Toggle maintenance mode
SW4	Press	Move Down/Close
	Release	Stop

The LEDs on the OM15082 Generic Expansion Boards provide local indications when LEDs are enabled in the Mode attribute as follows:

Table 8. LED Indication

LED	Indication	Description
D1	Off (with motor stopped)	Motor is in normal mode

Table 8. LED Indication*...continued*

LED	Indication	Description
	On (with motor stopped)	Motor is in reversed mode
	Flashing	Motor is operating
D2	Off	Window Covering is in normal mode
	On	Window Covering is in calibration mode
D3	Off	Motor is allowed to operate
	On	Motor is not allowed to operate (maintenance mode)

4.2.4.10 Calibration Mode

When in calibration mode the Window Covering allows movement between the maximum physical limits of the device. Whilst in this mode it records the minimum and maximum positions the Window Covering is moved to and applies them as the Installed Limits when exiting calibration mode (there must be at least 10cm between the recorded points).

When in normal mode the Window Covering will only be allowed to move between the previously recorded Installed Limits.

To easily calibrate the device from normal mode:

1. Move the Window Covering to a point between the two desired Installed Limits
2. Enter calibration mode (by writing to the Mode attribute or using SW2)
3. Move the Window Covering to the minimum and maximum desired Installed Limits (by issuing commands or using SW1/SW4)
4. Exit calibration mode (by writing to the Mode attribute or using SW2)

If a previous calibration prevents moving the Window Covering to a point between the desired limits it may be necessary to calibrate to the Physical Limits and then re-calibrate to the desired Installed Limits.

4.2.5 Rejoining a Network

As a Router, when this device is restarted in a state which is not factory-new, it will just resume operation in its previous state. All application, binding, group and network parameters are preserved in non-volatile memory.

4.2.6 Performing a Factory Reset

The Window Covering can be returned to its factory-new state (erasing all persistent data except the outgoing network frame counter) as follows:

- Hold down the USER INTERFACE button and press the RESET button on the Carrier Board.

The Window Covering will then broadcast a Leave Indication on the old network, then delete all persistent data (except the outgoing network frame counter) and perform a software reset.

There are two supported over-the-air commands for removing a device from the network - these are:

- Network Leave Request without rejoin
- ZDO Management Network Leave Request without rejoin

The Reset command of the Basic cluster will cause the ZCL to be reset to its factory-new defaults, resetting all attributes and configured reports. This will not remove the device from the network - all network parameters, groups and bindings will remain in place.

4.3 Installation Codes

Zigbee allows for devices to join a network using unique install codes, rather than the well-known Default Link Key. This install code is only used for the initial join and is replaced by the Trust Centre immediately after the join with a new unique Link Key to secure future communication between the Trust Centre and the individual device. An installation code is 16 bytes in length.

To build the devices in this Application Note to use install codes for joining, edit the command line for each device to set the build option `ICODE=1`, then clean and rebuild each device.

Each joining device can have a random but not necessarily unique install code. How to program the install code into a device is described in more detail below. The install code should be provisioned to the Trust Centre via out-of-band means when the node is commissioned. It will be used to create a preconfigured Link Key.

To provision installation code in real devices, 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. On your PC, open a command window.
4. In the command window, navigate to the Flash Programmer directory: **C:\NXP \DK6ProductionFlashProgrammer**
5. Put the DK006 device into programming mode by holding down the ISP button while pressing and releasing the RESET button.
6. Write the installation code in the PSECT using a command with the following format at the command prompt: `DK6Programmer.exe -s <comport> -w install_code=<code>` where `<comport>` is the number of the serial communications port. E.g. COM3 `<code>` is the 16 bytes installation code(alphabetic character are not case-sensitive). E.g. 01367ABD61045893AFF4A0E1C814B901
7. To commit the install code to psector memory the DK006 microcontroller should needs to be reset either using the reset button or automatically by the flash programmer.

To read the installation code, enter a command with the following format at the command prompt, in Flash Programmer directory: `DK6Programmer.exe -s <comport> -r install_code` where `<comport>` is the number of the serial communications port. E.g. COM3

Before a device can join to the network using the install code, the IEEE/MAC address and the install code programmed into this device need to be provided to the Trust Centre of the network.

To provide the install code and MAC address to the Coordinator template (from JN-AN-1243), use the following command from a serial interface on a host terminal connected to the Coordinator hardware.

```
Code <MAC Address> <Install code>
```

where:

- `<MAC Address>` is the IEEE/MAC address of the device (MSB first, and alphabetic characters are not case-sensitive).
- `<Install code>` is the install code (MSB first, alphabetic characters are not case-sensitive, and the bytes may be separated by colons (‘:’), commas (‘,’) or nothing).

For a device with the above IEEE/MAC address, the command would be:

code 00158D000035C9B8 01:36:7A:BD:61:04:58:93:AF:F4:A0:E1:C8:14:B9:01

To find the MAC address of the joining device, a command with the following format can be entered at the command prompt, in Flash Programmer directory:

```
DK6Programmer.exe -s <comport>
```

where `<comport>` is the number of the serial communications port. E.g. COM3

After provisioning the install code and IEEE/MAC address in the Trust Centre, the normal procedure for joining a new device to the network can be followed. Details about forming a network and allowing other devices to the network can be found in Allowing Other Devices to Join the Network.

Once the install code has been used to join the new device, it is replaced with a new Trust Centre Link Key (the key derived from the install code is discarded and not stored for re-use). If a device is factory reset, it will not be able to re-associate with the network until the install code is re-provisioned in the Trust Centre.

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 Window Covering device. In order to build with these options, add `OTA=1` to the command line before building (this is disabled by default). This will add the relevant functionality to the device and invoke post-build processing to create a bootable image and two upgrade images. The produced binaries will be stored in the **OTABuild** 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.

- The internal Flash memory is be used to store the upgrade image by default.

The initial client binaries to be programmed into the DK006 devices are V1 .bin files:

- **WindowCovering_NtagNwk_NtagOta_GpProxy_Ota_OM15082_V1.bin**

The initial client binaries to be programmed into the K32W148 devices are V1 .bin files:

- **WindowCovering_GpProxy_Ota_K32W148EVK_V1.bin**

The Application Note *Zigbee 3.0 IoT Control Bridge (JN-AN-1247)* has OTA server functionality built into it. 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:

- **WindowCovering_NtagNwk_NtagOta_GpProxy_Ota_OM15082_V2.ota**
- **WindowCovering_NtagNwk_NtagOta_GpProxy_Ota_OM15082_V3.ota**
- **WindowCovering_GpProxy_Ota_K32W148EVK_V1.ota**
- **WindowCovering_GpProxy_Ota_K32W148EVK_V2.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 0x0000 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 for an Window Covering device (0x0202). For an encrypted image the most significant bit is set to distinguish the ota file from an unencrypted image (0x8202). 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.

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 the Window Covering

This section describes step-by-step how the Window Covering was created from the JN-AN-1243 Router template.

- For the purposes of this Application Note we are going to create separate source code folders for each step to allow easy comparisons. These are not necessary when creating a “real-world” application in this scenario just one instance of each source folder can be used, and the number folder prefixes omitted.

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           ?= K32W041A
#JENNIC_CHIP           ?= K32W041AM
#JENNIC_CHIP           ?= K32W061
#JENNIC_CHIP           ?= K32W041
#JENNIC_CHIP           ?= JN5189
#JENNIC_CHIP           ?= JN5188
#JENNIC_CHIP           ?= K32W1480
```

6.2 Step 1 – Creating the New Project

This step shows how to create and set up the new project in MCUXpresso starting from the JN-AN-1243 Application Note. The functionality of the starting device is not changed here, this initial step is concerned with creating a new project including renaming files appropriately for the Window Covering we are going to create.

The table below summarises the copied and renamed source code folders from the original JN-AN-1243 Application Note to aid comparisons with difference tools:

Table 9. Source Folder

Source Folder	Destination Folder
JN-AN-1243\Binaries\<subfolders>	***Deleted***
JN-AN-1243\Common	JN-AN-1257\1-Common
JN-AN-1243\Coordinator	*Deleted*
JN-AN-1243\EndDevice	*Deleted*
JN-AN-1243\mcux	JN-AN-1257\mcux
JN-AN-1243\NFC	JN-AN-1257\NFC
JN-AN-1243\Router	JN-AN-1257\1-WindowCovering

The next table summarises renamed files within the folders to aid comparisons with difference tools:

Table 10. Source File

Source File	Destination File
JN-AN-1243\Router\Source\ app_router_node.c	JN-AN-1257\1-WindowCovering\Source\ app_window_ covering.c
JN-AN-1243\Router\Source\ app_router_node.h	JN-AN-1257\1-WindowCovering\Source\ app_window_ covering.h

The following sections go through these changes including the changes made within individual files step-by-step.

6.2.1 Step 1.1 – Copy JN-AN-1243

The first step is to copy the **JN-AN-1243** folder to a new folder in the workspace and rename it to **JN-AN-1257**.

6.2.2 Step 1.2 – Delete Unused Folders

The following folders can be deleted from the JN-AN-1257 folder as they will not be used by the On/Off Sensor:

- **Binaries\<subfolders>** the subfolders of the Binaries folder can all be deleted. New, correctly named, folders will be created when the applications are compiled.
- **Coordinator**
- **EndDevice**

6.2.3 Step 1.3 – Rename Source Folders and Files

As part of this Application Note we are going to retain the source code for each step to allow the changes to be followed easily. As we are currently in Step 1 rename the folders as follows:

- **Common** to **1-Common**
- **Router** to **1-WindowCovering**
- In a “real-world” scenario, without separate folders for each step, the **Common** folder can be left as-is and the **Router** folder renamed to **WindowCovering**. This also applies to the following steps when development will continue in this single set of source file folders.

Whilst the NFC folder contains source code we are not going to alter it and so this folder does not need to be renamed.

In the **1-WindowCovering\Source** folder rename the source files as follows:

- **app_router_node.c** to **app_window_covering.c**
- **app_router_node.h** to **app_window_covering.h**

6.2.4 Step 1.4 – Makefiles

In **1-WindowCovering\Build\mcux\Makefile** change the TARGET variable from “Router” to “WindowCovering”:

```
# Application target name
TARGET = WindowCovering
```

We are going to enable UART debugging at the application and ZCL layer by uncommenting the debug flags below:

```
CFLAGS += -DDEBUG_APP
```

```
CFLAGS += -DDEBUG_ZCL
```

As we have renamed the source folders the Window Covering Makefile will need to be updated to access the source files from their new locations. Add the “1-“ to the folder name variables as shown below:

```
APP_BLD_DIR         = $(APP_BASE)/1-$(TARGET)/Build/mcux
APP_SRC_DIR         = $(APP_BASE)/1-$(TARGET)/Source
APP_COMMON_SRC_DIR  = $(APP_BASE)/1-Common/Source
BOARD_DIR           = $(APP_BASE)/1-Common/Source/board
AWK_SRC             = $(APP_BASE)/1-Common/Awk
```

- In a “real-world” scenario, without separate folders for each step the above numbered prefix changes should not be applied. This also applies to the following steps when development will continue in this single set of source file folders.

Change the **app_router_node.c** source file to **app_window_covering.c** (as we renamed it above):

```
APPSRC += app_window_covering.c
```

The makefile variables APP_OUT_DIR (used to specify the output folder for the binaries) and APP_OBJ_DIR (used to specify the folder for object files) need to be updated for the device’s folder name change:

```
APP_OUT_DIR = $(APP_BASE)/1-$(TARGET)/Build/$(TARGET_FULL)
APP_OBJ_DIR = $(APP_BASE)/1-$(TARGET)/Build/$(TARGET_FULL)/obj
```

The **mcux\Build\Makefile** is used to compile all the devices in the Application Note. The Coordinator and EndDevice devices from JN-AN-1243 can be removed The Router Device paths for the renamed folders also need to be updated by changing “Router” to “1-WindowCovering”:

```
all:
    rm -f ../../Doc/size.txt
    $(MAKE) -C ../../Coordinator/Build/mcux DR=DONGLE clean -r
    $(MAKE) -C ../../Coordinator/Build/mcux DR=DONGLE all -r
    $(MAKE) -C ../../Coordinator/Build/mcux DR=OM15082 clean -r
    $(MAKE) -C ../../Coordinator/Build/mcux DR=OM15082 all -r
    $(MAKE) -C ../../Coordinator/Build/mcux DR=OM5578
        APP_NCI_ICODE=1 APP_NCI_OTA=1 clean -r
    $(MAKE) -C ../../Coordinator/Build/mcux DR=OM5578
        APP_NCI_ICODE=1 APP_NCI_OTA=1 all -r
    $(MAKE) -C ../../1-WindowCovering/Build/mcux APP_NTAG_NWK=0
        OTA=0 OTA_ENCRYPTED=0 clean -r
    $(MAKE) -C ../../1-WindowCovering/Build/mcux APP_NTAG_NWK=0
        OTA=0 OTA_ENCRYPTED=0 all -r
    $(MAKE) -C ../../1-WindowCovering/Build/mcux APP_NTAG_NWK=1
        APP_NTAG_OTA=1 OTA=1 OTA_ENCRYPTED=0 clean -r
    $(MAKE) -C ../../1-WindowCovering/Build/mcux APP_NTAG_NWK=1
        APP_NTAG_OTA=1 OTA=1 OTA_ENCRYPTED=0 all -r
    $(MAKE) -C ../../EndDevice/Build/mcux APP_NTAG_NWK=0 OTA=0
        OTA_ENCRYPTED=0 clean -r
    $(MAKE) -C ../../EndDevice/Build/mcux APP_NTAG_NWK=0 OTA=0
        OTA_ENCRYPTED=0 all -r
    $(MAKE) -C ../../EndDevice/Build/mcux APP_NTAG_NWK=1
        APP_NTAG_OTA=1 OTA=1 OTA_ENCRYPTED=0 RAMOPT=1 clean -r
    $(MAKE) -C ../../EndDevice/Build/mcux APP_NTAG_NWK=1
```

```
        APP_NTAG_OTA=1 OTA=1 OTA_ENCRYPTED=0 RAMOPT=1 all -r
```

In the sample files the comments at the start of the makefiles have also been updated.

6.2.5 Step 1.5 – Source Files

In **1-WindowCovering\Source\app_window_covering.h** make the following changes:

- APP_ROUTER_NODE_H_ to APP_WINDOW_COVERING_H_

This #define is used to prevent multiple inclusions of the header file.

As we have renamed this header file the source files that include it will need to change “app_router_node.h” to “app_window_covering.h”. The affected files are:

- **1-Common\Source\app_ota_client.c**
  - In this file the “#ifdef Router” before the #include should be changed to “#ifdef WindowCovering”
- **1-WindowCovering\Source\app_main.c**
- **1-WindowCovering\Source\app_window_covering.c**
- **1-WindowCovering\Source\app_start.c**
- **1-WindowCovering\Source\app_zcl_task.c**

In Step 1.8 we will cause the name of the #define used to set the application endpoint to change from “ROUTER_APPLICATION_ENDPOINT” to “WINDOWCOVERING_APPLICATION_ENDPOINT”. The following source files which use this #define need to be updated:

- **1-WindowCovering\Source\app_window_covering.c**
- **1-WindowCovering\Source\app_start.c**
- **1-WindowCovering\Source\app_zcl_task.c**
- **1-WindowCovering\Source\app_reporting.c**

The start-up debug message “ROUTER RESET” is changed to “WINDOW COVERING RESET” in **app_start.c**.

In **app_window_covering.c** the UART debug #define is corrected from:

```
#ifndef DEBUG_APP
    #define TRACE_APP   TRUE
#else
    #define TRACE_APP   TRUE
#endif
```

To

```
#ifdef DEBUG_APP
    #define TRACE_APP   TRUE
#else
    #define TRACE_APP   FALSE
#endif
```

In the sample files the comments at the start of the source files have also been updated.

6.2.6 Step 1.6 – Project Name

In **mcux\.project** file change the project name from “JN-AN-1243-Zigbee-3-0-Base-Device” to “JN-AN-1257-Zigbee-3-0-Developing-Clusters”. This is an XML file that can be edited in an XML editor or a text editor. The

project name is being changed outside of MCUXpresso to avoid conflict if the JN-AN-1243 project has already been imported into MCUXpresso.

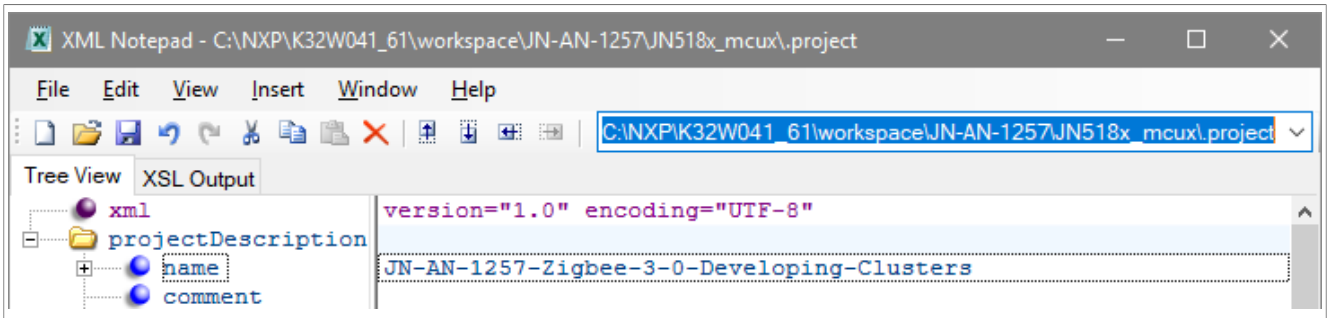

Figure 9. Project Name

6.2.7 Step 1.7 – MCUXpresso Setup

First import the JN-AN-1257 folder into MCUXpresso as a project.

Until Step 1.7 is completed MCUXpresso may generate the following error, simply click OK until they are cleared:

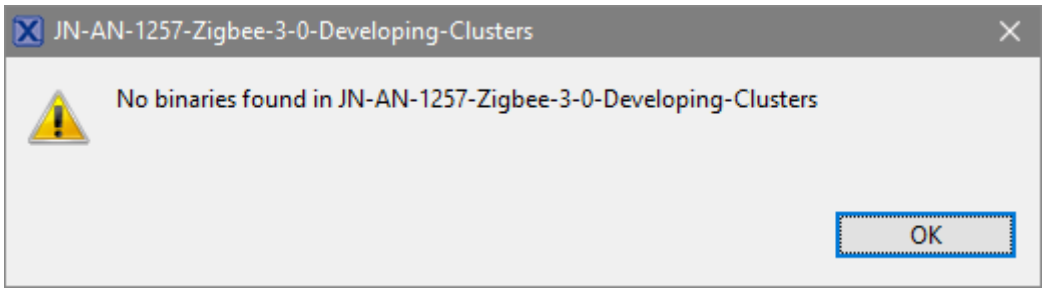

Figure 10. Error

Next the Common, Coordinator, EndDevice and Router folders can be deleted from the MCUXpresso project as they have either been deleted from the file system or renamed. Expand the tree under the JN-AN-1257-Zigbee-3-0-Developing-Clusters project in the Project Explorer select and delete these folders using the context menu or the delete key:

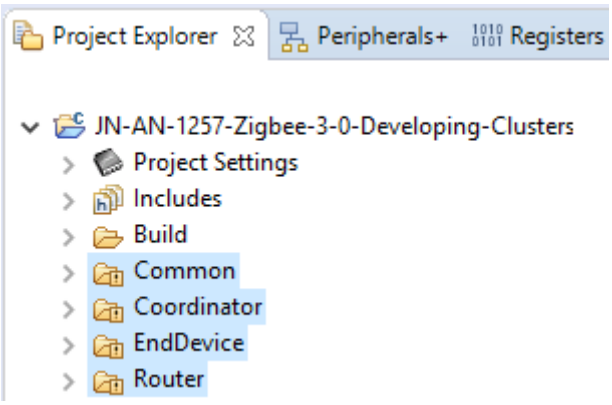

Figure 11. Project Explorer

With the JN-AN-1257-Zigbee-3-0-Developing-Clusters project selected in the Project Explorer on the Menu bar select Project > Build Configurations > Manage…

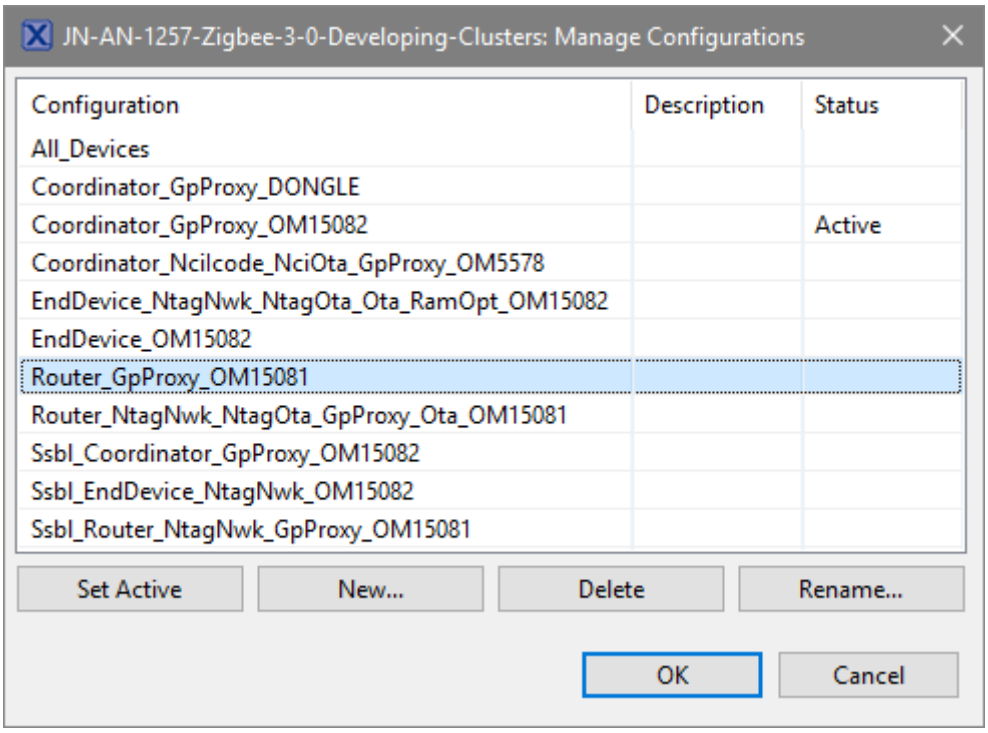

Figure 12. Build Configurations

The configurations beginning Coordinator, EndDevice and Ssbl configurations can be deleted to leave just the two configurations beginning Router plus the All Devices configuration. Theses Router configurations should be renamed to 1-WindowCovering:

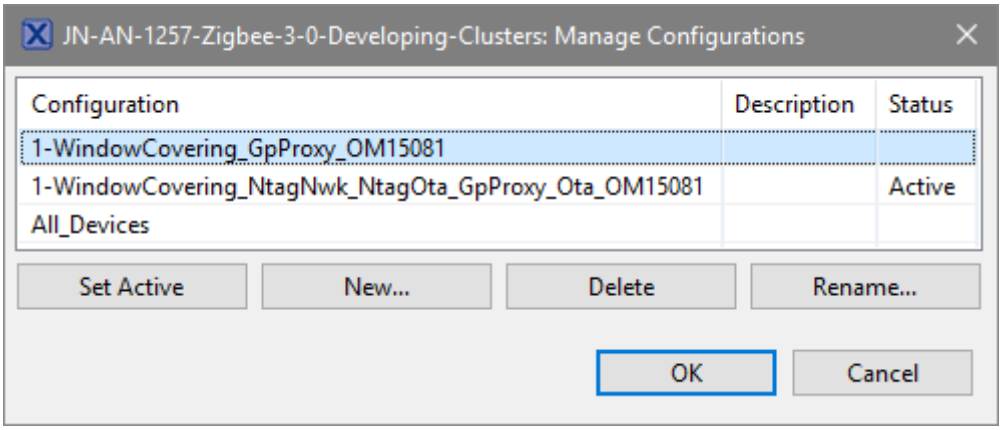

Figure 13. Rename

With the JN-AN-1257-Zigbee-3-0-Developing-Clusters project selected in the Project Explorer on the Menu bar select Project > Properties. In the Properties window select the C/C++ Build left in the panel on the left. Select the first WindowCovering configuration from the drop-down then edit the Build directory option to change the Router folder to 1-WindowCovering:

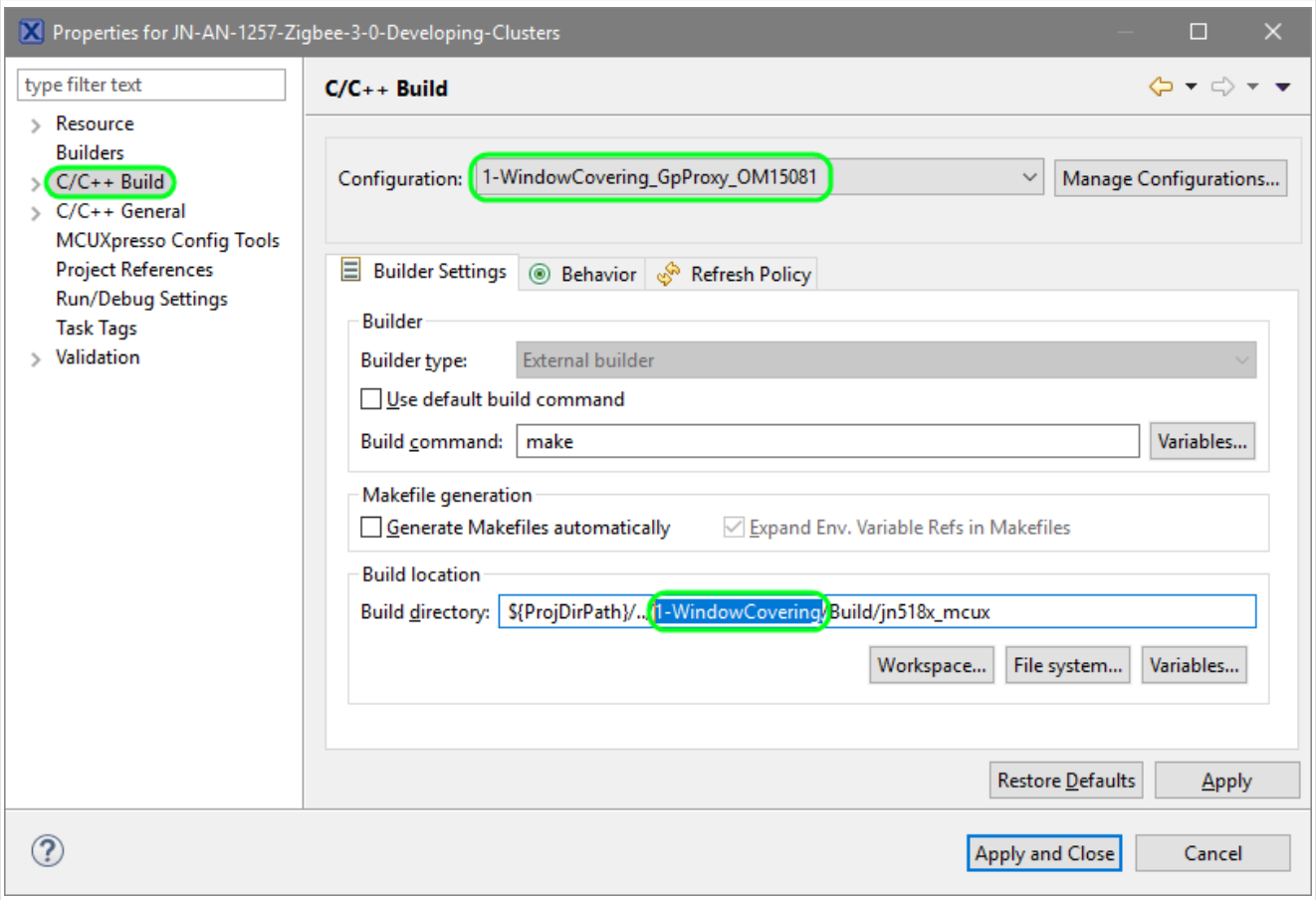

Figure 14. Edit Configurations

Click Apply to save the properties for the selected configuration. Then repeat with the second Router configuration and click Apply and Close to finish.

With the JN-AN-1257-Zigbee-3-0-Developing-Clusters project selected in the Project Explorer on the Menu bar select File > New > Other. In the window that opens expand the General part of the tree, select the Folder entry, then click the Next button:

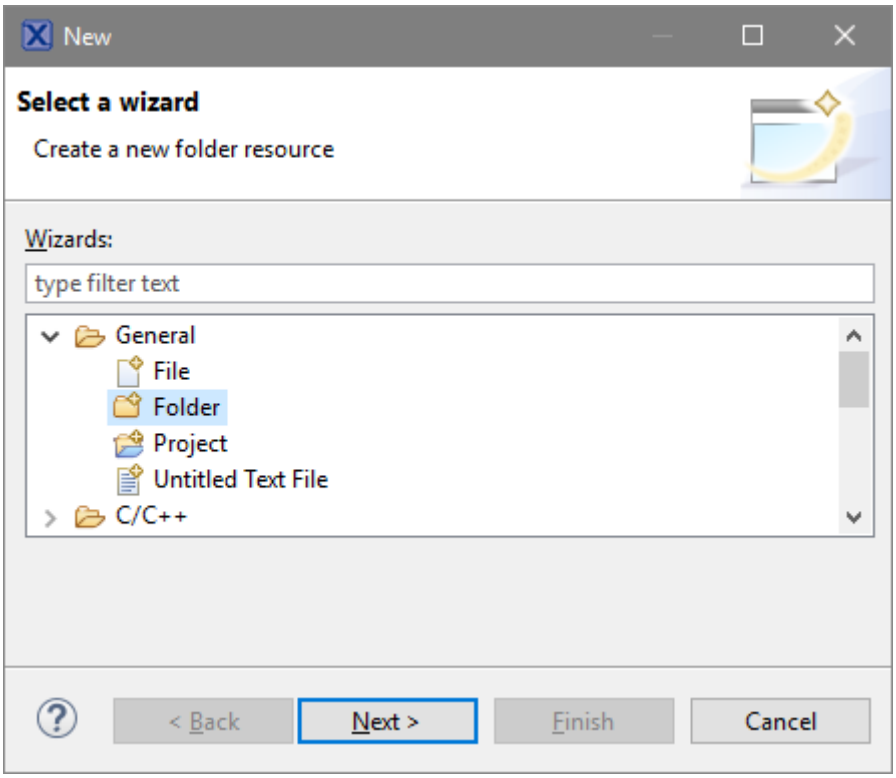

Figure 15. Create New Folder

In the next window select the JN-AN-1257-Zigbee-3-0-Developing-Clusters project in the upper list and click the Advanced >> button to show the additional controls. Select the Link to alternate location (Linked Folder) radio button and enter “PROJECT_LOC\..\1-Common” in the edit-line, then click the Finish button:

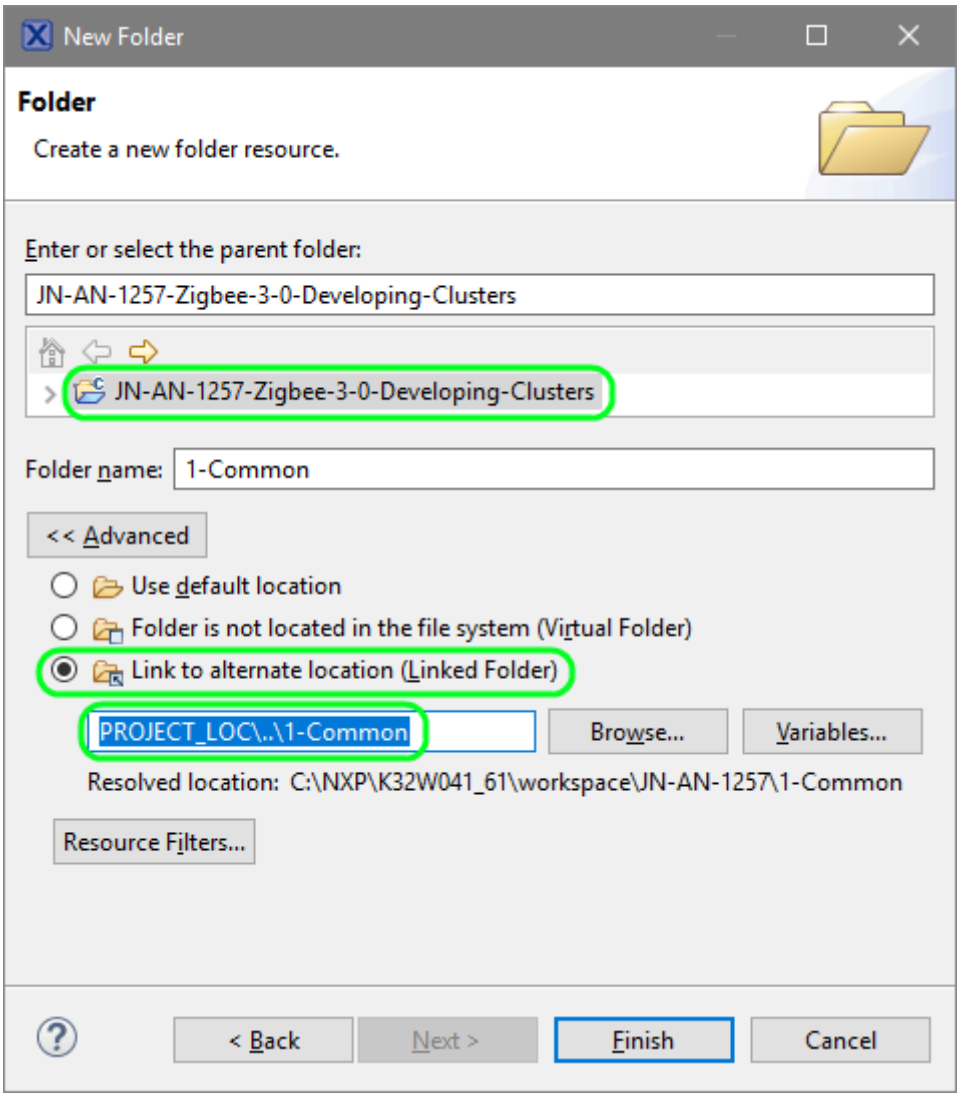

Figure 16. Link Folders

Repeat to add the **1-WindowCovering** folder to the project.

Linking the folders makes the source files for the project visible in the Project Explorer tree:

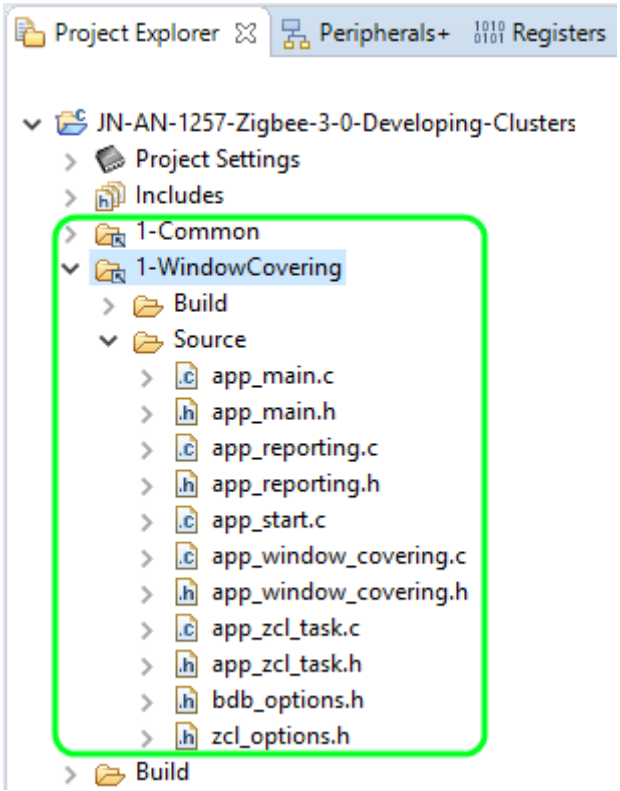

Figure 17. Project Explorer

6.2.8 Step 1.8 – app.zpscfg

In MCUXpresso navigate to the **1-Common\Source** folder under the project and double click the **app.zpscfg** file to open the Zigbee Configuration Editor plug-in. Expand the tree structure until the Coordinator, Router and Sleeping End Device nodes are displayed. Next select the Router node and in the panel below select the “Properties” tab:

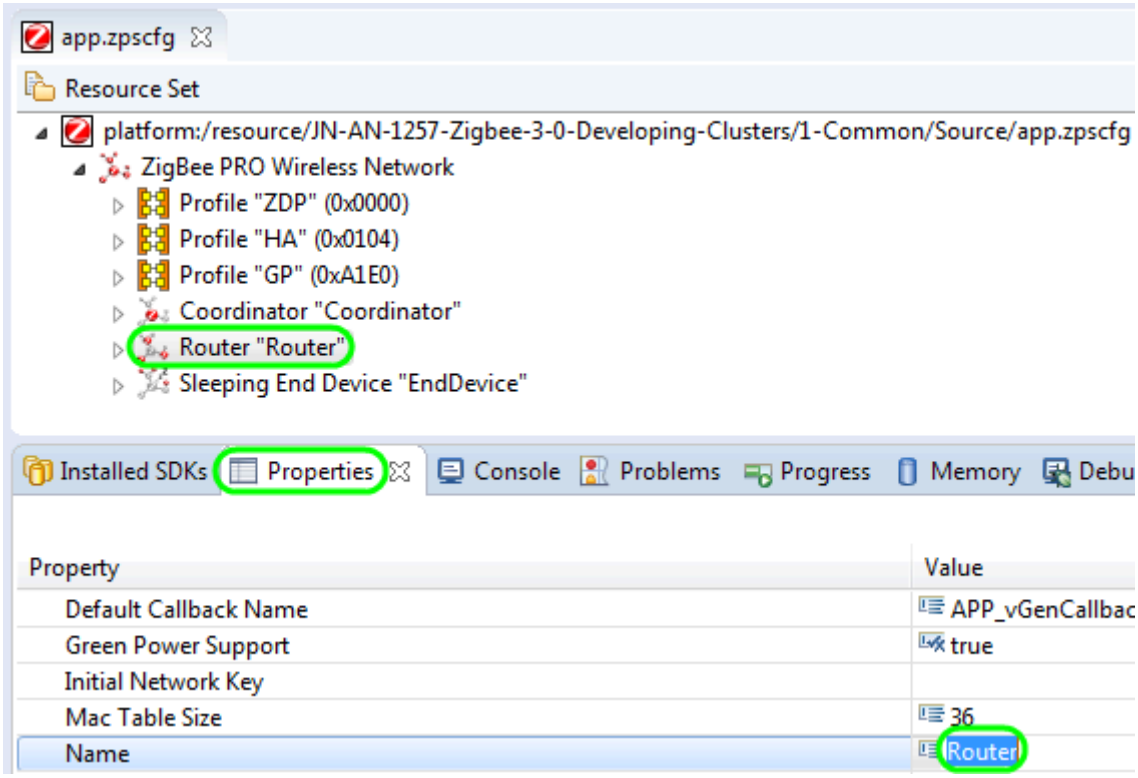

Figure 18. Edit ZPS config file

The name of the node can be changed from “Router” to “WindowCovering” and the file saved:

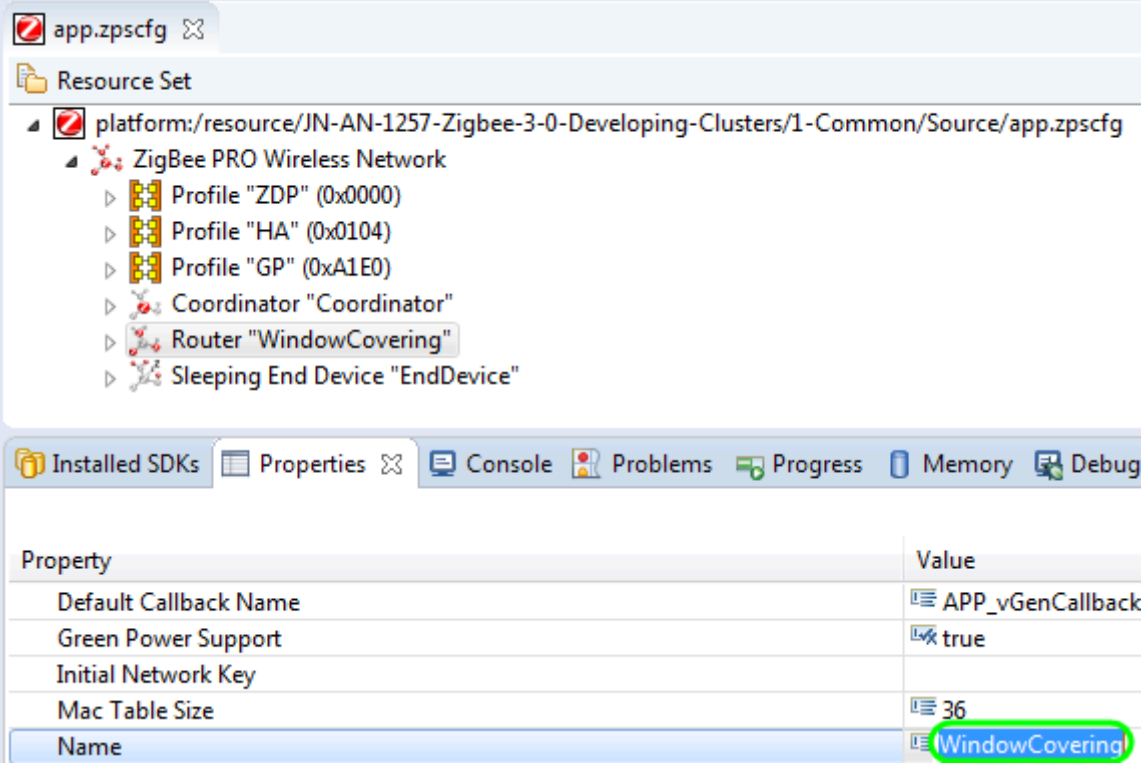

Figure 19. Edit ZPS config file

6.2.9 Step 1.10 – Compilation

It should now be possible to Clean and Build the WindowCovering applications from MCUXpresso.

These changes have not altered the functionality of the device it will still operate in exactly the same way as the JN-AN-1243 Template Router Device. They are simply made to ensure files are named appropriately for the Window Covering that is going to be built in the following steps.

6.3 Step 2 – Converting to a Window Covering Device

This step shows how to change the Zigbee device type from a Base Device type to a Window Covering type including the addition of the Window Covering device files and clusters.

The table below summarises the copied and renamed source code folders within the JN-AN-1257 Application Note to aid comparisons with difference tools:

Table 11. Source Folder

Source Folder	Destination Folder
JN-AN-1257\1-Common	JN-AN-1257\2-Common
JN-AN-1257\1-WindowCovering	JN-AN-1257\2-WindowCovering

The next table summarises new files added to the application and the original files in other Application Notes and the ZCL folder from which they are adapted to aid comparisons with difference tools:

Table 12. Source File

Source File	Destination File
JN-AN-1244\CommonLight\Source\app_scenes.c	JN-AN-1257\2-Common\Source\app_scenes.c
JN-AN-1244\CommonLight\Source\app_scenes.h	JN-AN-1257\2-Common\Source\app_scenes.h
JN-SW-4470\middleware\wireless\zigbee3.0\ZCL\ Devices\ZHA\Generic\Source\base_device.c	JN-AN-1257\2-Common\Source\ device_window_covering.c
JN-SW-4470\middleware\wireless\zigbee3.0\ZCL\ Devices\ZHA\Generic\Include\base_device.c	JN-AN-1257\2-Common\Source\ device_window_covering.h
JN-SW-4470\middleware\wireless\zigbee3.0\ZCL\ Clusters\General\Source\OnOff.c	JN-AN-1257\2-Common\Source\ cluster_window_covering.c
JN-SW-4470\middleware\wireless\zigbee3.0\ZCL\ Clusters\General\Source\OnOff_internal.h	JN-AN-1257\2-Common\Source\ cluster_window_covering_ internal.h
JN-SW-4470\middleware\wireless\zigbee3.0\ZCL\ Clusters\General\Source\OnOffCommandHandler.c	JN-AN-1257\2-Common\Source\ cluster_window_covering_ command_handler.c
JN-SW-4470\middleware\wireless\zigbee3.0\ZCL\ Clusters\General\Source\OnOffCommands.c	JN-AN-1257\2-Common\Source\ cluster_window_covering_ commands.c
JN-SW-4470\middleware\wireless\zigbee3.0\ZCL\ Clusters\General\Include\OnOff.h	JN-AN-1257\2-Common\Source\ cluster_window_covering.h

The following sections go through these changes including the changes made within individual files step-by-step.

6.3.1 Step 2.1 – Folders, Makefiles and Build Configurations

- This first sub-step creates copies of the Step 1 folders for further development into Step 2. It can be skipped if you are developing in a “real-world” scenario with just a single set of source file folders.

To create separate source file folders for this step, create copies of the existing folders as follows:

- **1-Common** to **2-Common**
- **1-WindowCovering** to **2-WindowCovering**

Next adjust the source code directories in **2-WindowCovering/Build/mcux/Makefile**:

```
APP_BLD_DIR         = $(APP_BASE)/2-$(TARGET)/Build/mcux
APP_SRC_DIR         = $(APP_BASE)/2-$(TARGET)/Source
APP_COMMON_SRC_DIR  = $(APP_BASE)/2-Common/Source
BOARD_DIR           = $(APP_BASE)/2-Common/Source/board
AWK_SRC             = $(APP_BASE)/2-Common/Awk
```

Plus, a similar change lower down in the makefile:

```
APP_OUT_DIR = $(APP_BASE)/2-$(TARGET)/Build/$(TARGET_FULL)
APP_OBJ_DIR = $(APP_BASE)/2-$(TARGET)/Build/$(TARGET_FULL)/obj
```

In **mcux/Build/Makefile** duplicate the lines from step 1 and adjust them to use the new step 2 folders:

```
$(MAKE) -C ../../1-WindowCovering/Build/mcux
    APP_NTAG_NWK=1 APP_NTAG_OTA=1 OTA=1 OTA_ENCRYPTED=0 all -r
$(MAKE) -C ../../2-WindowCovering/Build/mcux
    APP_NTAG_NWK=0 OTA=0 OTA_ENCRYPTED=0 clean -r
$(MAKE) -C ../../2-WindowCovering/Build/mcux
    APP_NTAG_NWK=0 OTA=0 OTA_ENCRYPTED=0 all -r
$(MAKE) -C ../../2-WindowCovering/Build/mcux
    APP_NTAG_NWK=1 APP_NTAG_OTA=1 OTA=1 OTA_ENCRYPTED=0 clean -r
$(MAKE) -C ../../2-WindowCovering/Build/mcux
    APP_NTAG_NWK=1 APP_NTAG_OTA=1 OTA=1 OTA_ENCRYPTED=0 all -r
```

In MCUXpresso select the project in the Project Explorer pane. The on the menu bar select Project > Build Configurations > Manage. In the window that opens click the New… button enter the name for the first Step 2 Window Covering build and copy the settings from the existing configuration for the Step 1 device, then click OK:

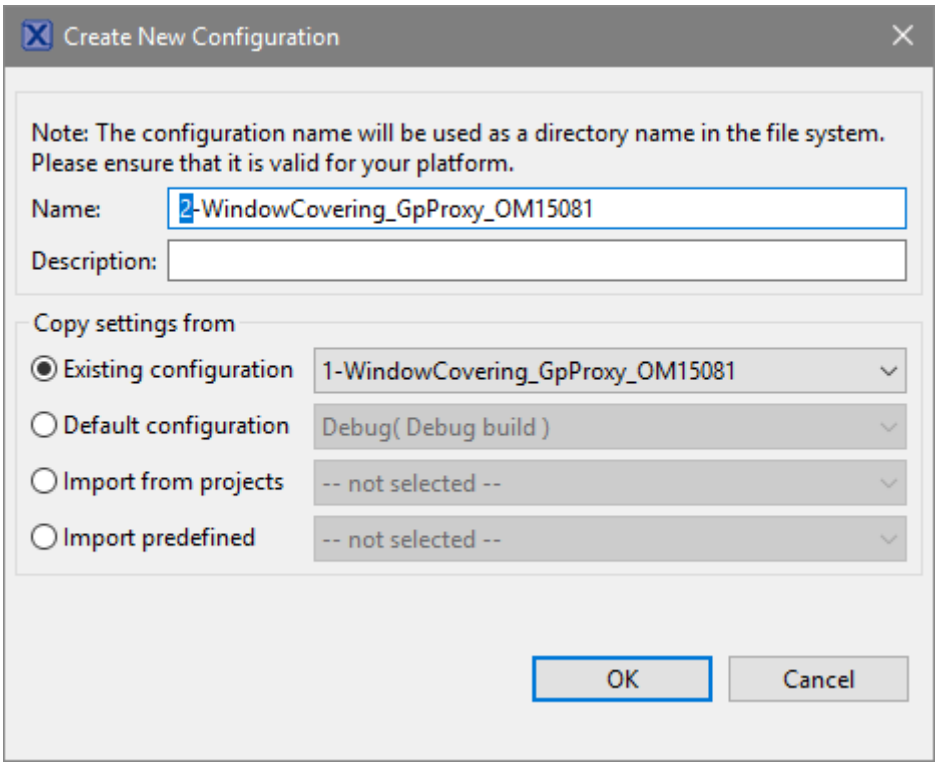

Figure 20. Create New Configuration

Repeat for the other Window Covering Configuration, then click OK on the Manage Configurations window.

With the project still selected in the Project Explorer pane select Project > Properties on the menu bar. In the window that opens select “C/C++ Build” in the tree on the left. Select the first step 2 Window Covering configuration from the top drop-down, update the path in the “Build directory” edit-line so it builds from the new folder for step 2. Then click the Apply button:

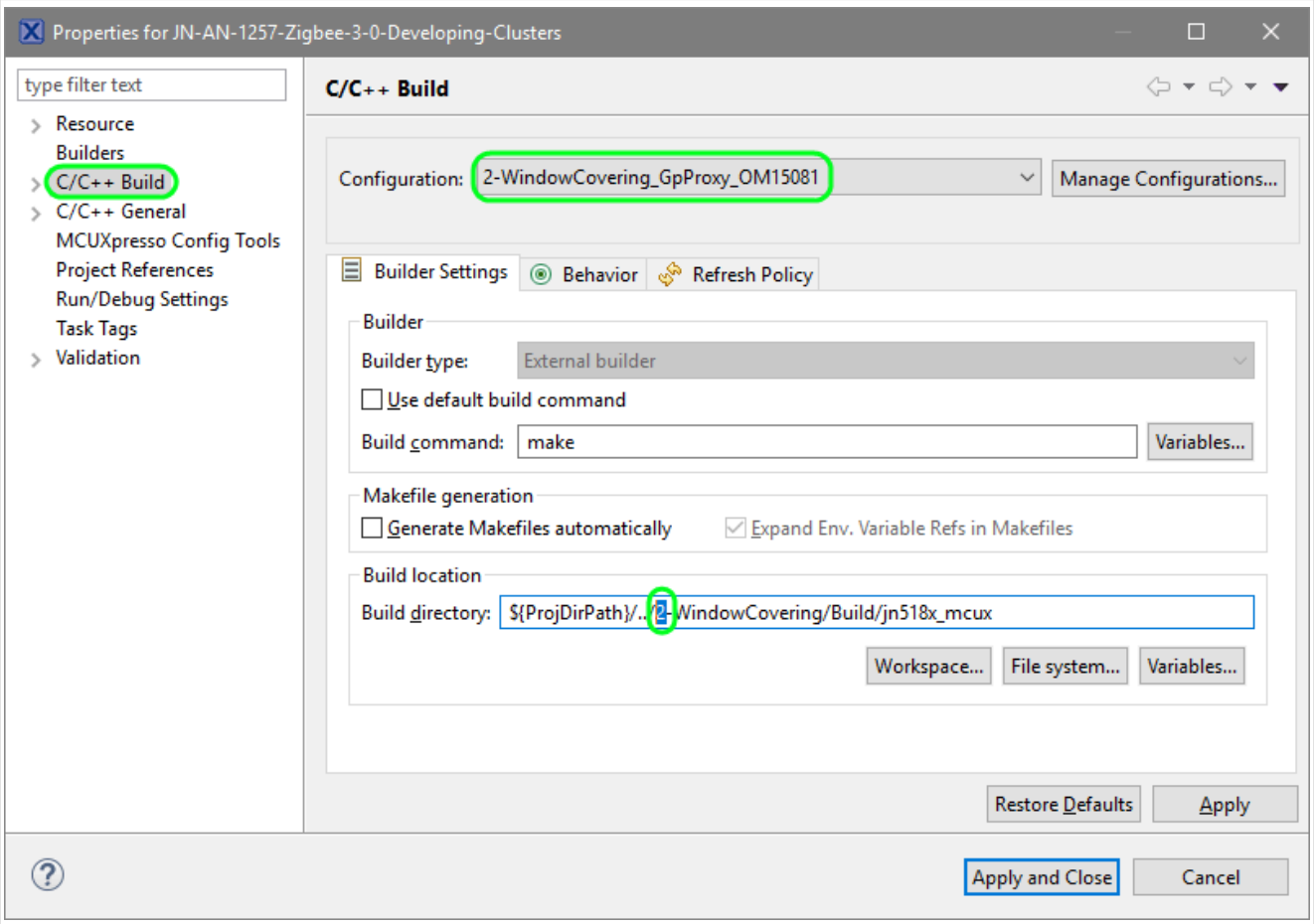

Figure 21. Edit Configurations

Repeat the process for the second step 2 Window Covering, then click Apply and Close.

Click the File > New > Other menu entry, select Folder in the tree then click the Next button:

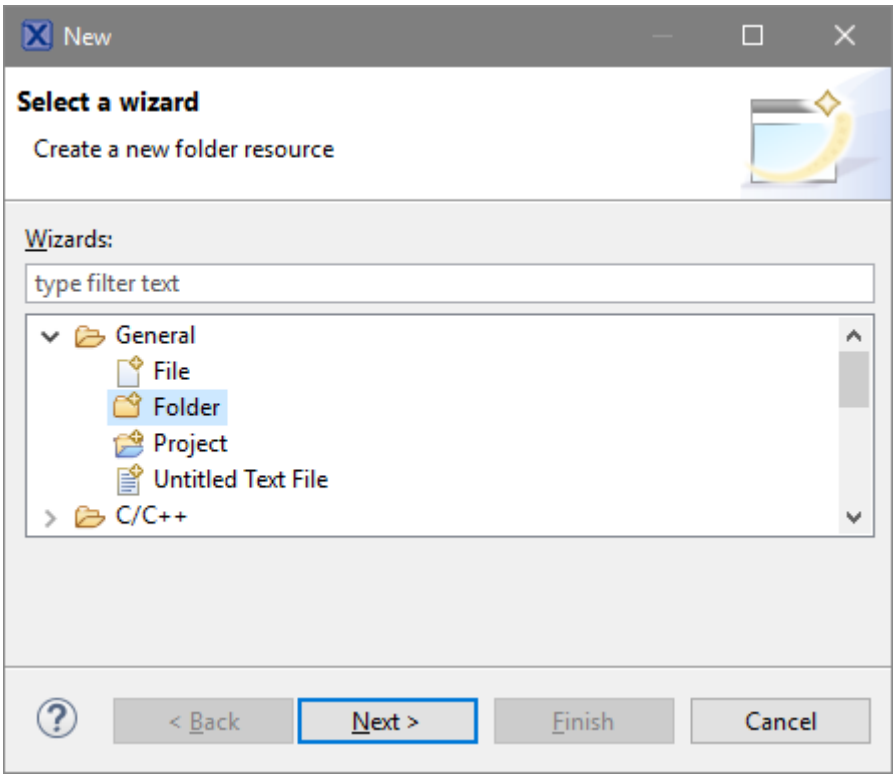

Figure 22. Create New Folder

In the next window ensure the JN-AN-1257-Zigbee-3-0-Developing-Clusters project is selected in the upper pane. Click the Advanced >> button to reveal the additional controls. Select the Link to alternate location (Linked Folder) radio button and enter “PROJECT_LOC\..\2-Common” in the edit-box. The Folder name edit-box should update to contain “2-Common” automatically:

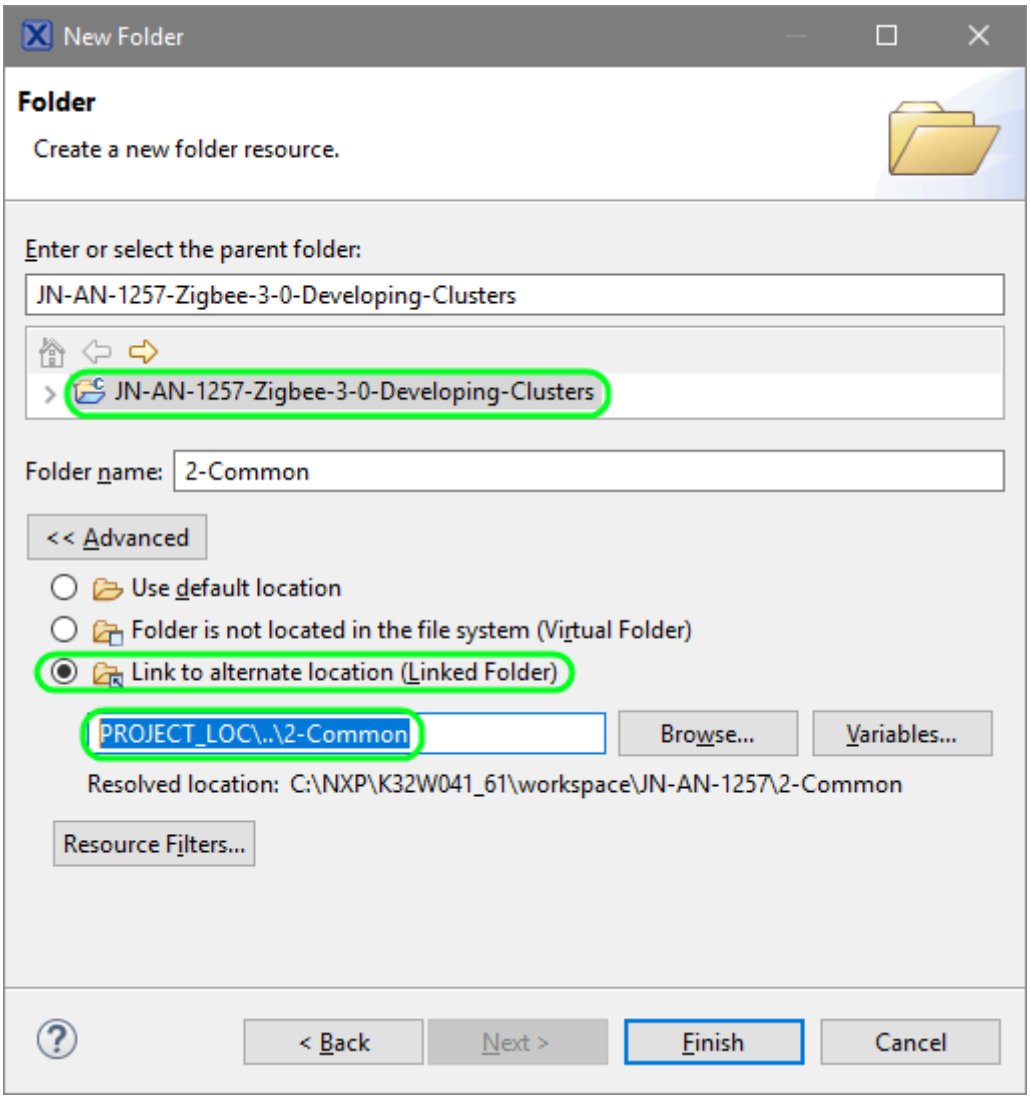

Figure 23. Link To Folders

Click Finish to add the folder, then repeat to add the 2-WindowCovering folder to the project:

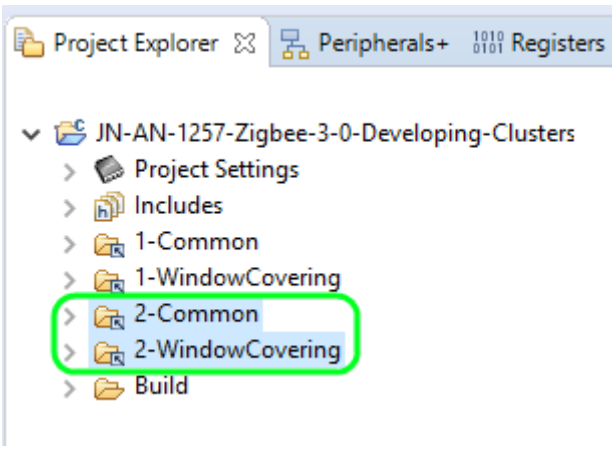

Figure 24. Add Folder

To check the Step 2 setup test that the new Build Configurations can be compiled.

6.3.2 Step 2.2 – Window Covering Device Type

The Router template taken from JN-AN-1243 is based upon a pseudo-device type, the Base Device. This is not a device type specified by Zigbee.org but one that has been implemented by NXP for the purposes of providing a minimal, but functional, template to build upon. To properly move to a Zigbee.org specified Window Covering device type the structure, clusters and options need to be moved from the Base Device type to the Window Covering type.

The ZCL code provided by NXP contains implementations for many device types and clusters specified by Zigbee.org however the Window Covering device and cluster are not provided in the NXP ZCL implementation. This sub-step works through the process of creating these resources to add them to the application.

Each ZCL device type implementation has a header and source file within the **JN-SW-4470\middleware \wireless\zigbee3.0\ZCL\Devices** folder. The code for the starting Base Device type is in the **ZHA\Generic** sub-folder (**base_device.h/c**), new files will be created from these for the Window Covering device within the application folder named **device_window_covering.h/c**.

Each device type holds its data in a structure, the Base Device type structure is **tsZHA_BaseDevice** declared in **base_device.h**, a new structure called **tsZHA_WindowCovering** will be declared in **device_window_covering.h**. The places in the application code that use **tsZHA_BaseDevice** need to be changed to use the **tsZHA_WindowCovering** structure.

6.3.2.1 2-WindowCovering\Source\app_zcl_task.h

The inclusion of the **base.device.h** header file needs to change to **device_window_covering.h** to access the structures and functions of the Window Covering device.

This file contains an external declaration of the Base Device structure for other application modules to use:

```
extern tsZHA_BaseDevice sBaseDevice;
```

This needs to be changed to the Window Covering structure and also renamed appropriately:

```
extern tsZHA_WindowCovering sWindowCovering;
```

6.3.2.2 2-WindowCovering\Source\app_zcl_task.c

The inclusion of the **base.device.h** header file needs to change to **device_window_covering.h** to access the structures and functions of the Window Covering.

The actual declaration of the Window Covering structure is in this file and needs to be changed from [needs updating to min retention changes]:

```
tsZHA_BaseDevice sBaseDevice;
```

to:

```
tsZHA_WindowCovering sWindowCovering;
```

This file re-initialises the data to 0 if a factory reset command is received which also requires an update (including the structure variable name):

```
memset(&sWindowCovering, 0, sizeof(tsZHA_WindowCovering));
```

As we have renamed a variable used in **app_zcl_task.h** from **sBaseDevice** to **sWindowCovering** we need to make this replacement throughout **app_zcl_task.c**.

This module also makes use of the Base Device function **eZHA_RegisterBaseDeviceEndPoint()** which needs to be changed to the equivalent function for a Window Covering **eZHA_RegisterWindowCoveringEndPoint()**.

The function APP_vZCL_DeviceSpecific_Init() contains a model identifier and other identifying data which can be updated as follows:

```
PRIVATE void APP_vZCL_DeviceSpecific_Init(void)
{
    sWindowCovering.sOnOffServerCluster.bOnOff = FALSE;
    memcpy(sWindowCovering.sBasicServerCluster.au8ManufacturerName,
        "NXP", CLD_BAS_MANUF_NAME_SIZE);
    memcpy(sWindowCovering.sBasicServerCluster.au8ModelIdentifier,
        "WindowCove", CLD_BAS_MODEL_ID_SIZE);
    memcpy(sWindowCovering.sBasicServerCluster.au8DateCode,
        "20181218", CLD_BAS_DATE_SIZE);
    memcpy(sWindowCovering.sBasicServerCluster.au8SWBuildID,
        "1000-0002", CLD_BAS_SW_BUILD_SIZE);
}
```

6.3.2.3 2-WindowCovering\Source\app_window_covering.c

This source file also makes use of the **sBaseDevice** structure variable which needs to be changed to **sWindowCovering**.

6.3.2.4 2-Common\Source\app_ota_client.c

The inclusion of the **base_device.h** header file needs to change to **device_window_covering.h** to access the structures and functions of the Window Covering.

This source file uses the structure name and variable which both need to be changed.

6.3.2.5 2-Common\Source\app.zpscfg

The Zigbee specified Device ID needs to be updated in this file, so the correct Device ID is returned when queried. The Window Covering uses 0x0202 (0d514).

Open the **app.zpscfg** file in MCUXpresso to access the ZPS Configuration Editor, expand the tree so the application End Point can be selected. Ensure that the Properties tab is selected in the lower pane and enter 514 (decimal) for the Application Device ID (the hexadecimal value will be displayed to the right). Click out of the edit-box and save the file:

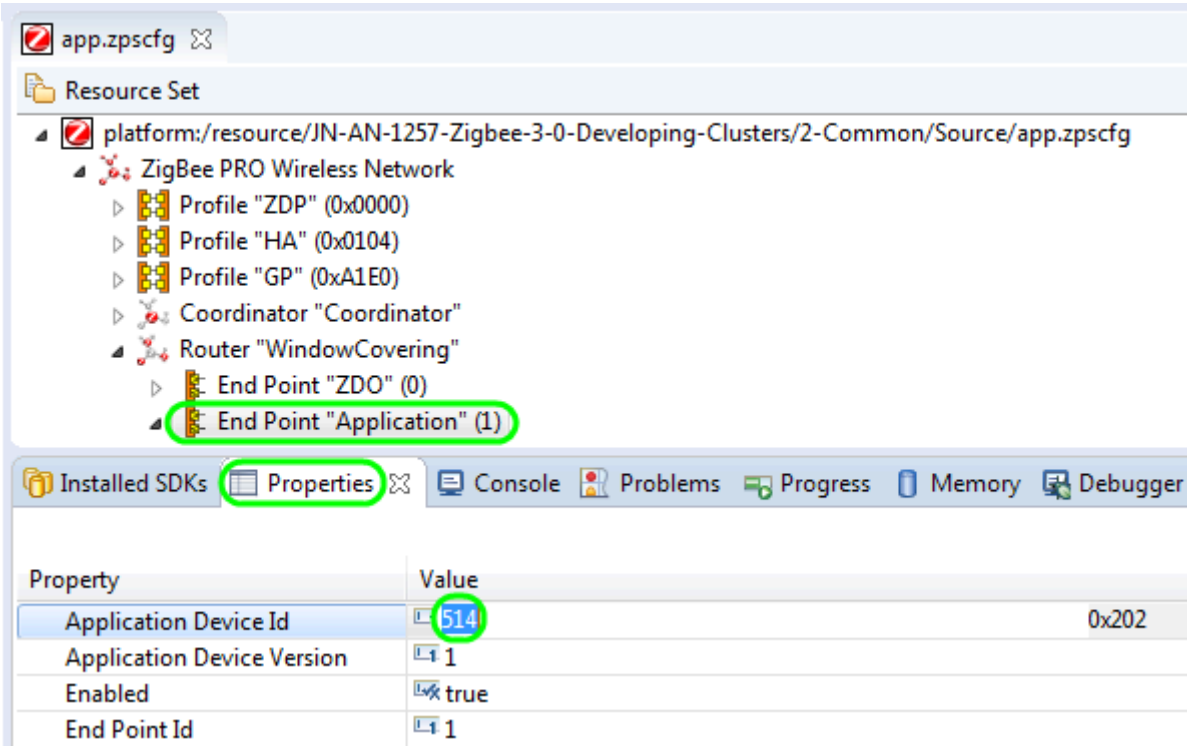

Figure 25. Edit ZPS Config file

6.3.2.6 2-WindowCovering\Build\mcux\Makefile

The Zigbee Device ID is also used as the ID for matching OTA images (with some manipulation of the most significant bit to flag encrypted images). In the NXP OTA implementation the OTA string is also used to determine whether an available OTA image is downloaded. These values should be changed in the makefile as follows:

```
MANUFACTURER_CODE = 0x1037
ifeq ($(OTA_ENCRYPTED),0)
    OTA_DEVICE_ID = 0x0202
    OTA_STRING = $(OTA_HW)-WCD-$(JENNIC_CHIP)00000000000000
else
    OTA_DEVICE_ID = 0x8202
    OTA_STRING = $(OTA_HW)-WCD-$(JENNIC_CHIP)--ENCRYPTED000
endif
```

Note that the string must retain its fixed length when altered.

6.3.3 Step 2.3 – Window Covering Device Code

With the application code updated to use a new Window Covering device implementation we next need to create the implementation using the Base Device implementation as a starting point. The Zigbee Home Automation Profile Specification details which clusters are mandatory or optional for each device type. For a Window Covering device the specification is as follows:

Table 13. Cluster

Server (Input) Side	Client (Output) Side
Mandatory	
Basic	*None*
Identify	
Groups	
Scenes	
Window Covering	
Optional	
None	OTA Upgrade

The Base Device implementation already includes the clusters coloured green. The Scenes cluster (coloured yellow) is present in the NXP ZCL but is not included in the Base Device implementation.

The Window Covering cluster is not included in the NXP ZCL and will be created during this step.

The OTA Upgrade cluster is included as an optional cluster it is a generally useful feature for any device type and is already inherited from the template Router.

NXP's Base Device also includes the On/Off cluster on the Server (Input) side this will be removed from the Window Covering application.

To create the Window Covering implementation copy the following files from the ZCL SDK folder (**JN-SW-4470\middleware\wireless\zigbee3.0\ZCL\Devices\ZHA\Generic**) into the application's **2-Common\Source** folder and rename as follows:

- **Include\base_device.h** to **device_window_covering.h**
- **Source\base_device.c** to **device_window_covering.c**

6.3.3.1 2-Common\Source\device_window_covering.h

The following updates are required in this file to implement the Window Covering device type:

The #define "BASE_DEVICE_H" is used to prevent multiple inclusions of the header file, this should be changed to "DEVICE_WINDOW_COVERING_H_":

```
#ifndef DEVICE_WINDOW_COVERING_H_
#define DEVICE_WINDOW_COVERING_H_
```

As we will be adding support for the Scenes cluster and creating a Window Covering cluster header files for these clusters will need to be included:

```
#include "Scenes.h"
#include "cluster_window_covering.h"
```

The #define GENERIC_BASE_DEVICE_ID should be changed to provide the Device ID for a Window Covering device:

```
#define WINDOW_COVERING_DEVICE_ID 0x0202
```

The ZCL cluster instance data storage for the Scenes and Window Covering clusters need to be added. This data is only allocated when the cluster is enabled, and an instance is allocated when used as a server and/or client. The options to use these clusters as clients is not required for the Window Covering device however by adding them here the same device implementation could be used to create a Window Covering Controller device:

```
#if (defined CLD_GROUPS) && (defined GROUPS_CLIENT)
    tsZCL_ClusterInstance sGroupsClient;
#endif
#if (defined CLD_SCENES) && (defined SCENES_SERVER)
    tsZCL_ClusterInstance sScenesServer;
#endif
#if (defined CLD_SCENES) && (defined SCENES_CLIENT)
    tsZCL_ClusterInstance sScenesClient;
#endif
#if (defined CLD_WINDOW_COVERING) && (defined WINDOW_COVERING_SERVER)
    tsZCL_ClusterInstance sWindowCoveringServer;
#endif
#if (defined CLD_WINDOW_COVERING) && (defined WINDOW_COVERING_CLIENT)
    tsZCL_ClusterInstance sWindowCoveringClient;
#endif
```

As the On/Off cluster is not part of a Window Covering device those cluster instances can be removed:

```
/* optional cluster */
#if (defined CLD_ONOFF) && (defined ONOFF_SERVER)
    tsZCL_ClusterInstance sOnOffServer;
#endif
#if (defined CLD_ONOFF) && (defined ONOFF_CLIENT)
    tsZCL_ClusterInstance sOnOffClient;
#endif
```

The name of the structure for these cluster instances is changed from Base Device to Window Covering:

```
} tsZHA_WindowCoveringClusterInstances __attribute__ ((aligned(4)));
```

The structure tsZHA_BaseDevice, which contains all the ZCL data for a device type, is defined here.

As the structure name for the cluster instances has been changed that update needs to be applied to this structure:

```
/* Holds everything required to create an instance of base device */
typedef struct
{
    tsZCL_EndPointDefinition sEndPoint;
    /* Cluster instances */
    tsZHA_WindowCoveringClusterInstances sClusterInstance;
```

The data members for the Scenes and Window covering clusters need to be added:

```
    tsCLD_GroupsCustomDataStructure sGroupsClientCustomDataStructure;
#endif
#if (defined CLD_SCENES) && (defined SCENES_SERVER)
```

```
    /* Scenes Cluster - Server */
    tsCLD_Scenes sScenesServerCluster;
    tsCLD_ScenesCustomDataStructure sScenesServerCustomDataStructure;
#endif
#if (defined CLD_SCENES) && (defined SCENES_CLIENT)
    /* Scenes Cluster - Client */
    tsCLD_Scenes sScenesClientCluster;
    tsCLD_ScenesCustomDataStructure sScenesClientCustomDataStructure;
#endif
#if (defined CLD_WINDOW_COVERING) && (defined WINDOW_COVERING_SERVER)
    /* Window Covering Cluster - Server */
    tsCLD_WindowCovering sWindowCoveringServerCluster;
    tsCLD_WindowCoveringCustomDataStructure
        sWindowCoveringServerCustomDataStructure;
#endif
#if (defined CLD_WINDOW_COVERING) && (defined WINDOW_COVERING_CLIENT)
    /* Window Covering Cluster - Client */
    tsCLD_WindowCoveringClient sWindowCoveringClientCluster;
    tsCLD_WindowCoveringCustomDataStructure
        sWindowCoveringClientCustomDataStructure;
#endif
```

The On/Off cluster members can be removed:

```
#if (defined CLD_ONOFF) && (defined ONOFF_SERVER)
    /* On/Off Cluster - Server */
    tsCLD_OnOff sOnOffServerCluster;
    tsCLD_OnOffCustomDataStructure     sOnOffServerCustomDataStructure;
#endif
#if (defined CLD_ONOFF) && (defined ONOFF_CLIENT)
    tsCLD_OnOffClient sOnOffClientCluster;
#endif
```

Finally this structure is renamed from Base Device to Window Covering:

```
} tsZHA_WindowCovering;
```

The eZHA_RegisterBaseDeviceEndPoint() function prototype should be renamed for a window covering and a Window Covering data structure passed in as a parameter:

```
PUBLIC teZCL_Status eZHA_RegisterWindowCoveringEndPoint(
    uint8 u8EndPointIdentifier,
    tfpZCL_ZCLCallBackFunction cbCallBack,
    tsZHA_WindowCovering *psDeviceInfo);
```

Comments in the file header have also been updated.

6.3.3.2 2-Common\Source\device_window_covering.c

The following updates are required in this file to implement to Window Covering device.

The inclusion of **base_device.h** should be changed to **device_window_covering.h**:

```
#include "device_window_covering.h"
```

The eZHA_RegisterBaseDeviceEndPoint() function should be changed to eZHA_RegisterWindowCoveringEnd Point() and the tsZHA_BaseDevice structure parameter to tsZHA_WindowCovering:

```
PUBLIC teZCL_Status eZHA_RegisterWindowCoveringEndPoint(
    uint8 u8EndPointIdentifier,
    tfpZCL_ZCLCallBackFunction cbCallBack,
    tsZHA_WindowCovering *psDeviceInfo)
```

Within this function the number of clusters member of the endpoint needs to be calculated from the tsZHA_ WindowCoveringClusterInstances structure instead of the tsZHA_BaseDeviceClusterInstances structure:

```
psDeviceInfo->sEndPoint.u16NumberOfClusters =
    sizeof(tsZHA_WindowCoveringClusterInstances) /
    sizeof(tsZCL_ClusterInstance);
```

The eZHA_RegisterWindowCoveringEndPoint() then creates each client or server cluster in turn. The creation of the Scenes and Window Covering cluster instances are added here after the Groups cluster. The creation of the On/Off clusters instances is removed.

6.3.4 Step 2.4 – Window Covering Cluster Code

Now that the Window Covering device source files are in place we need to create the Window Covering cluster code. This will be based upon the existing On/Off Cluster. To create the Window Covering cluster implementation copy the following files from the ZCL SDK folder (**JN-SW-4470\middleware\wireless\zigbee3. 0\ZCL\Clusters\General**) into the application's **2-Common\Source** folder and rename as follows:

- **Include\OnOff.h** to **cluster_window_covering.h**
- **Source\OnOff.c** to **cluster_window_covering.c**
- **Source\OnOff_internal.h** to **cluster_window_covering_internal.h**
- **Source\OnOffCommandHandler.c** to **cluster_window_covering_command_handler.c**
- **Source\OnOffCommands.c** to **cluster_window_covering_commands.c**

6.3.4.1 2-Common\Source\cluster_window_covering.h

Comments throughout the file referencing the On/Off cluster are updated to refer to the Window Covering cluster.

The #define ONOFF_H is used to prevent multiple inclusions of the header file, this should be changed to CLUSTER_WINDOW_COVERING_H_:

```
#ifndef CLUSTER_WINDOW_COVERING_H_
#define CLUSTER_WINDOW_COVERING_H_
```

The On/Off cluster ID #define is changed and renamed:

```
/* Cluster ID's */
#define CLUSTER_ID_WINDOW_COVERING 0x0102
```

The next set of commented out #defines are used to specify which optional attributes are to be compiled into the cluster. The #defines are listed here as a reference but should normally be defined (when the attribute is to be included) in the application's **zcl_options.h** file. The optional attributes for the On/Off cluster are removed and a #define added for each optional Window Covering cluster attribute:

```
/******************************************************************************/
/*             Window Covering Cluster - Attributes                           */
/*                                                                            */
/* Add the following #define's to your zcl_options.h file to add optional*/
/* attributes to the window covering cluster.                                 */
/******************************************************************************/
//#define CLD_WC_ATTR_PHYSICAL_CLOSED_LIMIT_LIFT
//#define CLD_WC_ATTR_PHYSICAL_CLOSED_LIMIT_TILT
//#define CLD_WC_ATTR_CURRENT_POSITION_LIFT
//#define CLD_WC_ATTR_CURRENT_POSITION_TILT
//#define CLD_WC_ATTR_NUMBER_OF_ACTUATIONS_LIFT
//#define CLD_WC_ATTR_NUMBER_OF_ACTUATIONS_TILT
//#define CLD_WC_ATTR_CURRENT_POSITION_LIFT_PERCENTAGE
//#define CLD_WC_ATTR_CURRENT_POSITION_TILT_PERCENTAGE
//#define CLD_WC_ATTR_INSTALLED_OPEN_LIMIT_LIFT
//#define CLD_WC_ATTR_INSTALLED_CLOSED_LIMIT_LIFT
//#define CLD_WC_ATTR_INSTALLED_OPEN_LIMIT_TILT
//#define CLD_WC_ATTR_INSTALLED_CLOSED_LIMIT_TILT
//#define CLD_WC_ATTR_VELOCITY_LIFT
//#define CLD_WC_ATTR_ACCELERATION_TIME_LIFT
//#define CLD_WC_ATTR_DECELERATION_TIME_LIFT
//#define CLD_WC_ATTR_INTERMEDIATE_SETPOINTS_LIFT
//#define CLD_WC_ATTR_INTERMEDIATE_SETPOINTS_TILT
```

A set of commented out #defines are used to specify which optional commands are to be compiled into the cluster. To enable the optional command the required #defines should be present in the application's **zcl_options.h** file:

```
/******************************************************************************/
/*             Window Covering Cluster - Commands                             */
/*                                                                            */
/* Add the following #define's to your zcl_options.h file to add optional*/
/* commands to the window covering cluster.                                   */
/******************************************************************************/
//#define CLD_WC_CMD_GO_TO_LIFT_VALUE
//#define CLD_WC_CMD_GO_TO_LIFT_PERCENTAGE
//#define CLD_WC_CMD_GO_TO_TILT_VALUE
//#define CLD_WC_CMD_GO_TO_TILT_PERCENTAGE
```

The bit definitions for the On/Off cluster On with Timed Off command are removed and the bit definitions for the Window Covering cluster Config/Status attribute are added:

```
/* Bit definitions in Config/Status attribute */
#define CLD_WC_CONFIG_BIT_OPERATIONAL             (1 << 0)
#define CLD_WC_CONFIG_BIT_ONLINE                  (1 << 1)
#define CLD_WC_CONFIG_BIT_REVERSED                (1 << 2)
#define CLD_WC_CONFIG_BIT_LIFT_CLOSED_LOOP        (1 << 3)
#define CLD_WC_CONFIG_BIT_TILT_CLOSED_LOOP        (1 << 4)
#define CLD_WC_CONFIG_BIT_LIFT_ENCODER_CONTROLLED (1 << 5)
#define CLD_WC_CONFIG_BIT_TILT_ENCODER_CONTROLLED (1 << 6)
```

The bit definitions for the Mode attribute are also added here:

```
/* Bit definitions for Mode attribute */
#define CLD_WC_MODE_BIT_REVERSED                    (1 << 0)
#define CLD_WC_MODE_BIT_CALIBRATION                 (1 << 1)
#define CLD_WC_MODE_BIT_MAINTENANCE                 (1 << 2)
#define CLD_WC_MODE_BIT_LEDS_ON                     (1 << 3)
```

The On/Off cluster revision #define is changed to the Window Covering:

```
#ifndef CLD_WC_CLUSTER_REVISION
    #define CLD_WC_CLUSTER_REVISION 1
#endif
```

The next changes from On/Off to Window Covering are the enumerations for the cluster command IDs:

```
/* Window Covering Command IDs */
typedef enum
{
    E_CLD_WC_CMD_UP_OPEN                = 0x00, /* Mandatory */
    E_CLD_WC_CMD_DOWN_CLOSE,                    /* Mandatory */
    E_CLD_WC_CMD_STOP,                          /* Mandatory */
    E_CLD_WC_CMD_GO_TO_LIFT_VALUE       = 0x04,
    E_CLD_WC_CMD_GO_TO_LIFT_PERCENTAGE,
    E_CLD_WC_CMD_GO_TO_TILT_VALUE       = 0x07,
    E_CLD_WC_CMD_GO_TO_TILT_PERCENTAGE
} teCLD_WindowCovering_Command;
```

Followed by the enumerations for the attribute IDs:

```
/* Window Covering Attribute IDs */
typedef enum
{
    /* Window Covering Information Attribute Set */
    E_CLD_WC_ATTR_ID_WINDOW_COVERING_TYPE      = 0x0000, /* Mandatory */
    E_CLD_WC_ATTR_ID_PHYSICAL_CLOSED_LIMIT_LIFT,         /* Optional */
    E_CLD_WC_ATTR_ID_PHYSICAL_CLOSED_LIMIT_TILT,         /* Optional */
    E_CLD_WC_ATTR_ID_CURRENT_POSITION_LIFT,              /* Optional */
    E_CLD_WC_ATTR_ID_CURRENT_POSITION_TILT,              /* Optional */
    E_CLD_WC_ATTR_ID_NUMBER_OF_ACTUATIONS_LIFT,          /* Optional */
    E_CLD_WC_ATTR_ID_NUMBER_OF_ACTUATIONS_TILT,          /* Optional */
    E_CLD_WC_ATTR_ID_CONFIG_STATUS,                      /* Mandatory */
    E_CLD_WC_ATTR_ID_CURRENT_POSITION_LIFT_PERCENTAGE,   /* Mandatory for
                                                           closed loop */
    E_CLD_WC_ATTR_ID_CURRENT_POSITION_TILT_PERCENTAGE,   /* Mandatory for
                                                           closed loop */
    /* Window Covering Settings Attribute Set */
    E_CLD_WC_ATTR_ID_INSTALLED_OPEN_LIMIT_LIFT = 0x0010, /* Mandatory for
                                                           closed loop */
    E_CLD_WC_ATTR_ID_INSTALLED_CLOSED_LIMIT_LIFT,        /* Mandatory for
                                                           closed loop */
    E_CLD_WC_ATTR_ID_INSTALLED_OPEN_LIMIT_TILT,          /* Mandatory for
                                                           closed loop */
    E_CLD_WC_ATTR_ID_INSTALLED_CLOSED_LIMIT_TILT,        /* Mandatory for
                                                           closed loop */
```

```
    E_CLD_WC_ATTR_ID_VELOCITY_LIFT,                         /* Optional */
    E_CLD_WC_ATTR_ID_ACCELERATION_TIME_LIFT,                /* Optional */
    E_CLD_WC_ATTR_ID_DECELERATION_TIME_LIFT,                /* Optional */
    E_CLD_WC_ATTR_ID_MODE,                                  /* Mandatory */
    E_CLD_WC_ATTR_ID_INTERMEDIATE_SETPOINTS_LIFT,           /* Optional */
    E_CLD_WC_ATTR_ID_INTERMEDIATE_SETPOINTS_TILT            /* Optional */
} teCLD_WindowCovering_ClusterID;
```

The teCLD_OnOff_StartUpOnOff attribute values enumeration can be replaced with an enumeration for the Window Covering Type attribute values:

```
/* Window Covering Type Attribute Values */
typedef enum
{
    E_CLD_WC_TYPE_ROLLERSHADE = 0x00,
    E_CLD_WC_TYPE_ROLLERSHADE_2_MOTOR,
    E_CLD_WC_TYPE_ROLLERSHADE_EXTERIOR,
    E_CLD_WC_TYPE_ROLLERSHADE_EXTERIOR_2_MOTOR,
    E_CLD_WC_TYPE_DRAPERY,
    E_CLD_WC_TYPE_AWNING,
    E_CLD_WC_TYPE_SHUTTER,
    E_CLD_WC_TYPE_TILT_BLIND_TILT_ONLY,
    E_CLD_WC_TYPE_TILT_BLIND_LIFT_AND_TILT,
    E_CLD_WC_TYPE_PROJECTOR_SCREEN
} teCLD_WindowCovering_Type;
```

The structure to hold the attributes for the On/Off cluster server is updated for the Window Covering cluster server. Only members for attributes enabled by the #defines from the application's **zcl_options.h** file are included in the structure:

```
#ifdef WINDOW_COVERING_SERVER
/* Window Covering Cluster */
typedef struct
{
    zenum8            eWindowCoveringType;
#ifdef CLD_WC_ATTR_PHYSICAL_CLOSED_LIMIT_LIFT
    zuint16           u16PhysicalClosedLimitLift;
#endif
#ifdef CLD_WC_ATTR_PHYSICAL_CLOSED_LIMIT_TILT
    zuint16           u16PhysicalClosedLimitTilt;
#endif
#ifdef CLD_WC_ATTR_CURRENT_POSITION_LIFT
    zuint16           u16CurrentPositionLift;
#endif
#ifdef CLD_WC_ATTR_CURRENT_POSITION_TILT
    zuint16           u16CurrentPositionTilt;
#endif
#ifdef CLD_WC_ATTR_NUMBER_OF_ACTUATIONS_LIFT
    zuint16           u16NumberOfActuationsLift;
#endif
#ifdef CLD_WC_ATTR_NUMBER_OF_ACTUATIONS_TILT
    zuint16           u16NumberOfActuationsTilt;
#endif
    zbmap8            u8ConfigStatus;
#ifdef CLD_WC_ATTR_CURRENT_POSITION_LIFT_PERCENTAGE
    zuint8            u8CurrentPositionLiftPercentage;
#endif
```

```
#ifdef CLD_WC_ATTR_CURRENT_POSITION_TILT_PERCENTAGE
    zuint8            u8CurrentPositionTiltPercentage;
#endif
#ifdef CLD_WC_ATTR_INSTALLED_OPEN_LIMIT_LIFT
    zuint16           u16InstalledOpenLimitLift;
#endif
#ifdef CLD_WC_ATTR_INSTALLED_CLOSED_LIMIT_LIFT
    zuint16           u16InstalledClosedLimitLift;
#endif
#ifdef CLD_WC_ATTR_INSTALLED_OPEN_LIMIT_TILT
    zuint16           u16InstalledOpenLimitTilt;
#endif
#ifdef CLD_WC_ATTR_INSTALLED_CLOSED_LIMIT_TILT
    zuint16           u16InstalledClosedLimitTilt;
#endif
#ifdef CLD_WC_ATTR_VELOCITY_LIFT
    zuint16           u16VelocityLift;
#endif
#ifdef CLD_WC_ATTR_ACCELERATION_TIME_LIFT
    zuint16           u16AccelerationTimeLift;
#endif
#ifdef CLD_WC_ATTR_DECELERATION_TIME_LIFT
    zuint16           u16DecelerationTimeLift;
#endif
    zbmap8            u8Mode;
#ifdef CLD_WC_ATTR_INTERMEDIATE_SETPOINTS_LIFT
    tsZCL_OctetString sIntermediateSetpointsLift;
#endif
#ifdef CLD_WC_ATTR_INTERMEDIATE_SETPOINTS_TILT
    tsZCL_OctetString sIntermediateSetpointsTilt;
#endif
    zuint16           u16ClusterRevision;
} tsCLD_WindowCovering;
#endif
```

The On/Off client structure is also updated for a Window Covering client:

```
#ifdef WINDOW_COVERING_CLIENT
/* Window Covering Cluster */
typedef struct
{
    zuint16           u16ClusterRevision;
} tsCLD_WindowCoveringClient;
#endif
```

The On/Off command payloads are replaced with Window Covering command payloads:

```
#ifdef CLD_WC_CMD_GO_TO_LIFT_VALUE
/* Go to lift value command payload */
typedef struct
{
    zuint16 u16LiftValue;
} tsCLD_WindowCovering_GoToLiftValuePayload;
#endif
#ifdef CLD_WC_CMD_GO_TO_LIFT_PERCENTAGE
/* Go to lift percentage command payload */
typedef struct
```

```
{
    zuint8  u8LiftPercentage;
} tsCLD_WindowCovering_GoToLiftPercentagePayload;
#endif
#ifdef CLD_WC_CMD_GO_TO_TILT_VALUE
/* Go to tilt value command payload */
typedef struct
{
    zuint16 u16TiltValue;
} tsCLD_WindowCovering_GoToTiltValuePayload;
#endif
#ifdef CLD_WC_CMD_GO_TO_TILT_PERCENTAGE
/* Go to tilt percentage command payload */
typedef struct
{
    zuint8  u8TiltPercentage;
} tsCLD_WindowCovering_GoToTiltPercentagePayload;
#endif
```

The custom data structure is also updated:

```
/* Custom data structure */
typedef struct
{
    uint8                    u8Dummy;
} tsCLD_WindowCoveringCustomDataStructure;
```

The callback event message structure is also updated:

```
/* Window Covering Command Callback Event Structure */
typedef struct
{
    uint8                           u8CommandId;
    union
    {
    #ifdef CLD_WC_CMD_GO_TO_LIFT_VALUE
        tsCLD_WindowCovering_GoToLiftValuePayload
            *psGoToLiftValuePayload;
    #endif
    #ifdef CLD_WC_CMD_GO_TO_LIFT_PERCENTAGE
        tsCLD_WindowCovering_GoToLiftPercentagePayload
            *psGoToLiftPercentagePayload;
    #endif
    #ifdef CLD_WC_CMD_GO_TO_TILT_VALUE
        tsCLD_WindowCovering_GoToTiltValuePayload
            *psGoToTiltValuePayload;
    #endif
    #ifdef CLD_WC_CMD_GO_TO_TILT_PERCENTAGE
        tsCLD_WindowCovering_GoToTiltPercentagePayload
            *psGoToTiltPercentagePayload;
    #endif
    } uMessage;
} tsCLD_WindowCoveringCallBackMessage;
```

Next is the function to create the cluster:

```
PUBLIC teZCL_Status eCLD_WindowCoveringCreateWindowCovering(
    tsZCL_ClusterInstance                  *psClusterInstance,
    bool_t                                  bIsServer,
    tsZCL_ClusterDefinition                *psClusterDefinition,
    void                                   *pvEndPointSharedStructPtr,
    uint8                                  *pu8AttributeControlBits,
    tsCLD_WindowCoveringCustomDataStructure *psCustomDataStructure);
```

Then functions to send commands:

```
#ifdef WINDOW_COVERING_CLIENT
PUBLIC teZCL_Status eCLD_WindowCoveringCommandSend(
    uint8                          u8SourceEndPointId,
    uint8                          u8DestinationEndPointId,
    tsZCL_Address                 *psDestinationAddress,
    uint8                         *pu8TransactionSequenceNumber,
    teCLD_WindowCovering_Command   eCommand);
#ifdef CLD_WC_CMD_GO_TO_LIFT_VALUE
PUBLIC teZCL_Status eCLD_WindowCoveringCommandGoToLiftValueSend(
    uint8          u8SourceEndPointId,
    uint8          u8DestinationEndPointId,
    tsZCL_Address *psDestinationAddress,
    uint8         *pu8TransactionSequenceNumber,
    tsCLD_WindowCovering_GoToLiftValuePayload *psPayload);
#endif
#ifdef CLD_WC_CMD_GO_TO_LIFT_PERCENTAGE
PUBLIC teZCL_Status eCLD_WindowCoveringCommandGoToLiftPercentageSend(
    uint8          u8SourceEndPointId,
    uint8          u8DestinationEndPointId,
    tsZCL_Address *psDestinationAddress,
    uint8         *pu8TransactionSequenceNumber,
    tsCLD_WindowCovering_GoToLiftPercentagePayload *psPayload);
#endif
#ifdef CLD_WC_CMD_GO_TO_TILT_VALUE
PUBLIC teZCL_Status eCLD_WindowCoveringCommandGoToTiltValueSend(
    uint8          u8SourceEndPointId,
    uint8          u8DestinationEndPointId,
    tsZCL_Address *psDestinationAddress,
    uint8         *pu8TransactionSequenceNumber,
    tsCLD_WindowCovering_GoToTiltValuePayload *psPayload);
#endif
#ifdef CLD_WC_CMD_GO_TO_TILT_PERCENTAGE
PUBLIC teZCL_Status eCLD_WindowCoveringCommandGoToTiltPercentageSend(
    uint8          u8SourceEndPointId,
    uint8          u8DestinationEndPointId,
    tsZCL_Address *psDestinationAddress,
    uint8         *pu8TransactionSequenceNumber,
    tsCLD_WindowCovering_GoToTiltPercentagePayload *psPayload);
#endif
#endif /* WINDOW_COVERING_CLIENT */
```

The eCLD_OnOffUpdate() function will not have an equivalent function for the Window Covering cluster and its prototype can be removed.

Finally, some data variables are made externally available and again updated for a Window Covering cluster:

```
/****************************************************************************/
```

```
/***        External Variables                                            ***/
/*****************************************************************************/
#ifdef WINDOW_COVERING_SERVER
extern tsZCL_ClusterDefinition sCLD_WindowCovering;
extern const tsZCL_AttributeDefinition
                        asCLD_WindowCoveringClusterAttributeDefinitions[];
extern uint8 au8WindowCoveringAttributeControlBits[];
#endif
#ifdef WINDOW_COVERING_CLIENT
extern tsZCL_ClusterDefinition sCLD_WindowCoveringClient;
extern const tsZCL_AttributeDefinition
                  asCLD_WindowCoveringClientClusterAttributeDefinitions[];
extern uint8 au8WindowCoveringClientAttributeControlBits[];
#endif
```

6.3.4.2 2-Common\Source\cluster_window_covering.c

Comments throughout the file referencing the On/Off cluster are updated to refer to the Window Covering cluster.

Include files are updated from On/Off to Window Covering:

```
#include "cluster_window_covering.h"
#include "string.h"
#include "cluster_window_covering_internal.h"
```

UART debugging #defines are next with TRACE_ONOFF replaced by TRACE_WINDOW_COVERING throughout the file:

```
#include "dbg.h"
#ifdef DEBUG_CLD_WINDOW_COVERING
#define TRACE_WINDOW_COVERING TRUE
#else
#define TRACE_WINDOW_COVERING FALSE
#endif
```

The cluster inclusion test (CLD_ONOFF) is updated:

```
#ifdef CLD_WINDOW_COVERING
```

Throughout the file ONOFF_SERVER is replaced by WINDOW_COVERING_SERVER and ONOFF_CLIENT is replaced by WINDOW_COVERING_CLIENT.

The local function prototype for handling scene events is updated:

```
PRIVATE teZCL_Status eCLD_WindowCoveringSceneEventHandler(
                    teZCL_SceneEvent            eEvent,
                    tsZCL_EndPointDefinition   *psEndPointDefinition,
                    tsZCL_ClusterInstance      *psClusterInstance);
```

The command definitions for command discovery are updated:

```
#ifdef ZCL_COMMAND_DISCOVERY_SUPPORTED
    const tsZCL_CommandDefinition
        asCLD_WindowCoveringClusterCommandDefinitions[] = {
         {E_CLD_WC_CMD_UP_OPEN,               E_ZCL_CF_RX}  /* Mandatory */
        ,{E_CLD_WC_CMD_DOWN_CLOSE,            E_ZCL_CF_RX}  /* Mandatory */
        ,{E_CLD_WC_CMD_STOP,                  E_ZCL_CF_RX}  /* Mandatory */
#ifdef CLD_WC_CMD_GO_TO_LIFT_VALUE
        ,{E_CLD_WC_CMD_GO_TO_LIFT_VALUE,      E_ZCL_CF_RX}  /* Optional */
#endif
#ifdef CLD_WC_CMD_GO_TO_LIFT_PERCENTAGE
        ,{E_CLD_WC_CMD_GO_TO_LIFT_PERCENTAGE, E_ZCL_CF_RX}  /* Optional */
#endif
#ifdef CLD_WC_CMD_GO_TO_TILT_VALUE
        ,{E_CLD_WC_CMD_GO_TO_TILT_VALUE,      E_ZCL_CF_RX}  /* Optional */
#endif
#ifdef CLD_WC_CMD_GO_TO_TILT_PERCENTAGE
        ,{E_CLD_WC_CMD_GO_TO_TILT_PERCENTAGE, E_ZCL_CF_RX}  /* Optional */
#endif
    };
#endif
```

The attribute definition array containing the attribute IDs, access control settings, data types and data pointers for each attribute are updated (only the first few attributes are shown below):

```
const tsZCL_AttributeDefinition
 asCLD_WindowCoveringClusterAttributeDefinitions[] = {
    {E_CLD_WC_ATTR_ID_WINDOW_COVERING_TYPE,
    (E_ZCL_AF_RD),
     E_ZCL_ENUM8,
    (uint32)(&((tsCLD_WindowCovering*)(0))->eWindowCoveringType),
     0},
#ifdef CLD_WC_ATTR_PHYSICAL_CLOSED_LIMIT_LIFT
    {E_CLD_WC_ATTR_ID_PHYSICAL_CLOSED_LIMIT_LIFT,
    (E_ZCL_AF_RD),
     E_ZCL_UINT16,
    (uint32)(&((tsCLD_WindowCovering*)(0))->u16PhysicalClosedLimitLift),
     0},
#endif
#ifdef CLD_WC_ATTR_PHYSICAL_CLOSED_LIMIT_TILT
    {E_CLD_WC_ATTR_ID_PHYSICAL_CLOSED_LIMIT_TILT,
    (E_ZCL_AF_RD),
     E_ZCL_UINT16,
    (uint32)(&((tsCLD_WindowCovering*)(0))->u16PhysicalClosedLimitTilt),
     0},
#endif
```

The attributes in the scene extension table are updated:

```
/* List of attributes in the scene extension table */
tsZCL_SceneExtensionTable sCLD_WindowCoveringSceneExtensionTable =
{
    eCLD_WindowCoveringSceneEventHandler,
    2,
    {{E_CLD_WC_ATTR_ID_CURRENT_POSITION_LIFT_PERCENTAGE, E_ZCL_UINT8},
     {E_CLD_WC_ATTR_ID_CURRENT_POSITION_TILT_PERCENTAGE, E_ZCL_UINT8}}
};
```

The cluster definition structure is then declared:

```
tsZCL_ClusterDefinition sCLD_WindowCovering = {
    CLUSTER_ID_WINDOW_COVERING,
    FALSE,
    E_ZCL_SECURITY_NETWORK,
    (sizeof(asCLD_WindowCoveringClusterAttributeDefinitions) /
        sizeof(tsZCL_AttributeDefinition)),
    (tsZCL_AttributeDefinition*)
        asCLD_WindowCoveringClusterAttributeDefinitions,
    &sCLD_WindowCoveringSceneExtensionTable
#ifdef ZCL_COMMAND_DISCOVERY_SUPPORTED
    ,
    (sizeof(asCLD_WindowCoveringClusterCommandDefinitions) /
        sizeof(tsZCL_CommandDefinition)),
    (tsZCL_CommandDefinition*)
        asCLD_WindowCoveringClusterCommandDefinitions
#endif
};
```

The final data initialisation for the Window Covering server cluster are the attribute control bits:

```
uint8 au8WindowCoveringAttributeControlBits[
(sizeof(asCLD_WindowCoveringClusterAttributeDefinitions) /
 sizeof(tsZCL_AttributeDefinition))];
```

A similar set of updates are applied to the data for the Window Covering cluster operating in client mode:

```
#ifdef WINDOW_COVERING_CLIENT
const tsZCL_AttributeDefinition
    asCLD_WindowCoveringClientClusterAttributeDefinitions[] = {
    {E_CLD_GLOBAL_ATTR_ID_CLUSTER_REVISION,
    (E_ZCL_AF_RD|E_ZCL_AF_GA),
     E_ZCL_UINT16,
    (uint32)(&((tsCLD_WindowCoveringClient*)(0))->u16ClusterRevision),
     0}
};
tsZCL_ClusterDefinition sCLD_WindowCoveringClient = {
    CLUSTER_ID_WINDOW_COVERING,
    FALSE,
    E_ZCL_SECURITY_NETWORK,
    (sizeof(asCLD_WindowCoveringClientClusterAttributeDefinitions) /
     sizeof(tsZCL_AttributeDefinition)),
    (tsZCL_AttributeDefinition*)
        asCLD_WindowCoveringClientClusterAttributeDefinitions,
     NULL
#ifdef ZCL_COMMAND_DISCOVERY_SUPPORTED
     ,
     0,
     NULL
#endif
};
uint8 au8WindowCoveringClientAttributeControlBits[
    (sizeof(asCLD_WindowCoveringClientClusterAttributeDefinitions) /
     sizeof(tsZCL_AttributeDefinition))];
```

```
#endif /* WINDOW_COVERING_CLIENT */
```

The create cluster function eCLD_OnOffCreateOnOff() and its contents are updated for the Window covering cluster as eCLD_WindowCoveringCreateWindowCovering (not shown).

The following functions for the On/Off cluster have no equivalents in the Window Covering cluster and are removed:

- eCLD_OnOffSetState()
- eCLD_OnOffGetState()
- eCLD_OnOffUpdate()
- eCLD_OnOffSetGlobalSceneControl()

The scene event handler function is updated, including the removal of the On/Off cluster Global Scene Control attribute:

```
/****************************************************************************
 **
 ** NAME:       eCLD_WindowCoveringSceneEventHandler
 **
 ** DESCRIPTION:
 ** Handles events generated by a scenes cluster (if present)
 **
 ** PARAMETERS:                 Name                        Usage
 ** teZCL_SceneEvent            eEvent                      Scene Event
 ** tsZCL_EndPointDefinition    *psEndPointDefinition       End Point
 **                                                         Definition
 ** tsZCL_ClusterInstance       *psClusterInstance          Cluster
 **                                                         structure
 **
 ** RETURN:
 ** teZCL_Status
 **
 ****************************************************************************/
PRIVATE  teZCL_Status eCLD_WindowCoveringSceneEventHandler(
    teZCL_SceneEvent            eEvent,
    tsZCL_EndPointDefinition   *psEndPointDefinition,
    tsZCL_ClusterInstance      *psClusterInstance)
{
    tsZCL_CallBackEvent sZCL_CallBackEvent;
#ifdef CLD_ONOFF_ATTR_GLOBAL_SCENE_CONTROL
    tsCLD_OnOff *psSharedStruct = (tsCLD_OnOff *)psClusterInstance->
                                                  pvEndPointSharedStructPtr;
#endif
    switch(eEvent)
    {
    case E_ZCL_SCENE_EVENT_SAVE:
        DBG_vPrintf(TRACE_WINDOW_COVERING, "\r\nWC: Scene Event: Save");
        break;
    case E_ZCL_SCENE_EVENT_RECALL:
        DBG_vPrintf(TRACE_WINDOW_COVERING, "\r\nWC: Scene Event: Recall");
        /* See ZCL specification 3.8.2.2.2 */
#ifdef CLD_ONOFF_ATTR_GLOBAL_SCENE_CONTROL
        psSharedStruct->bGlobalSceneControl = 1;
#endif
        /* Inform the application that the cluster has just been updated */
        sZCL_CallBackEvent.u8EndPoint = psEndPointDefinition->
                                                        u8EndPointNumber;
```

```
        sZCL_CallBackEvent.psClusterInstance = psClusterInstance;
        sZCL_CallBackEvent.pZPSevent = NULL;
        sZCL_CallBackEvent.eEventType = E_ZCL_CBET_CLUSTER_UPDATE;
        // call user
        psEndPointDefinition->pCallBackFunctions(&sZCL_CallBackEvent);
#if (defined CLD_SCENES) && (defined SCENES_SERVER)
        vCLD_ScenesUpdateSceneValid(psEndPointDefinition);
#endif
        break;
    }
    return E_ZCL_SUCCESS;
}
```

6.3.4.3 2-Common\Source\cluster_window_covering_command_handler.c

This module handles incoming Window Covering commands for the cluster operating in server mode. The original On/Off cluster code is updated to handle Window Covering commands.

6.3.4.4 2-Common\Source\cluster_window_covering_commands.c

This module is used to transmit commands for the cluster operating in client mode. The On/Off cluster code is updated to send Window Covering commands.

6.3.4.5 2-Common\Source\cluster_window_covering_internal.h

This header contains function prototypes for the function calls between the Window Covering cluster source files.

6.3.4.6 2-WindowCovering\Source\app_zcl_task.c

As the On/Off cluster has been removed references to this cluster need to be removed when the cluster is not compiled into the application (one example shown below):

```
#ifdef CLD_ONOFF
    APP_vSetLed(LED1, sWindowCovering.sOnOffServerCluster.bOnOff);
#endif
```

Other changes are made to correctly control LED1 when in identify mode without reliance on the On/Off clusters On/Off state attribute.

This module also handles the reception of custom commands for each cluster so code needs to be added for the Window Covering cluster. The following case statement is added to the APP_vHandleClusterCustomCommands() function to call a handler for Window Covering cluster commands:

```
#if (defined CLD_WINDOW_COVERING) && (defined WINDOW_COVERING_SERVER)
    case CLUSTER_ID_WINDOW_COVERING:
    {
        teZCL_CommandStatus eStatus;
        tsCLD_WindowCoveringCallBackMessage *psCallBackMessage =
            (tsCLD_WindowCoveringCallBackMessage *)psEvent->
             uMessage.sClusterCustomMessage.pvCustomData;
        eStatus = APP_eHandleWindowCoveringCommand(psCallBackMessage);
        if (eStatus != E_ZCL_CMDS_SUCCESS)
        {
```

```
            /* Send default response */
            eZCL_SendDefaultResponse(psEvent->pZPSevent, eStatus);
        }
    }
    break;
#endif
```

The APP_eHandleWindowCoveringCommand() function is added to **app_zcl_task.c** that simply debugs out the received commands (not shown).

6.3.4.7 2-WindowCovering\Source\app_reporting.c

This module is updated to support attribute reporting from the Window Covering cluster instead of the On/Off cluster.

6.3.4.8 2-WindowCovering\Source\zcl_options.h

The enumeration for attribute reports is updated from the On/Off cluster to the Window Covering cluster:

```
/* Reporting related configuration */
enum
{
 REPORT_WINDOWCOVERING_SLOT = 0,
 NUMBER_OF_REPORTS
};
```

The **device_window_covering.h** header will allow the full set of possible clusters for the Window Covering device type to be enabled using a set of #defines. In the application layer the required #defines are made in the **zcl_options.h** header file (in the **2-WindowCovering\Source** folder). Examining the **zcl_options.h** file that originally came from the template Router shows that the clusters coloured green and yellow in the above table are already in place. The new Scenes and Window Covering server clusters are added and the unused On/Off cluster is removed:

```
/*****************************************************************/
/*                          Enable Cluster                       */
/*                                                               */
/* Add the following #define's to your zcl_options.h file to enable      */
/* cluster and their client or server instances                  */
/*****************************************************************/
#define CLD_BASIC
#define BASIC_SERVER
#define CLD_IDENTIFY
#define IDENTIFY_SERVER
#define CLD_GROUPS
#define GROUPS_SERVER
#define CLD_ONOFF
#define ONOFF_SERVER
#define CLD_SCENES
#define SCENES_SERVER
#define CLD_WINDOW_COVERING
#define WINDOW_COVERING_SERVER
#ifdef BUILD_OTA
#define CLD_OTA
#define OTA_CLIENT
#endif
```

The next set of #defines enable optional attributes in each cluster. The options for the unused On/Off cluster can be removed. The options for the Scenes cluster are added:

```
/******************************************************************************/
/*             Scenes Cluster - Optional Attributes                           */
/*                                                                            */
/* Add the following #define's to your zcl_options.h file to add optional*/
/* attributes to the Scenes cluster.                                          */
/******************************************************************************/
#define CLD_SCENES_MAX_NUMBER_OF_SCENES                    16
#define CLD_SCENES_DISABLE_NAME_SUPPORT
#define CLD_SCENES_MAX_SCENE_NAME_LENGTH                   0
#define CLD_SCENES_MAX_SCENE_STORAGE_BYTES                 10
#define CLD_SCENES_ATTR_LAST_CONFIGURED_BY
```

The options for the Window Covering cluster's optional attributes are added (and all are enabled at this stage):

```
/******************************************************************************/
/*             Window Covering Cluster - Optional Attributes                  */
/*                                                                            */
/* Add the following #define's to your zcl_options.h file to add optional*/
/* attributes to the window covering cluster.                                 */
/******************************************************************************/
#define CLD_WC_ATTR_PHYSICAL_CLOSED_LIMIT_LIFT
#define CLD_WC_ATTR_PHYSICAL_CLOSED_LIMIT_TILT
#define CLD_WC_ATTR_CURRENT_POSITION_LIFT
#define CLD_WC_ATTR_CURRENT_POSITION_TILT
#define CLD_WC_ATTR_NUMBER_OF_ACTUATIONS_LIFT
#define CLD_WC_ATTR_NUMBER_OF_ACTUATIONS_TILT
#define CLD_WC_ATTR_CURRENT_POSITION_LIFT_PERCENTAGE
#define CLD_WC_ATTR_CURRENT_POSITION_TILT_PERCENTAGE
#define CLD_WC_ATTR_INSTALLED_OPEN_LIMIT_LIFT
#define CLD_WC_ATTR_INSTALLED_CLOSED_LIMIT_LIFT
#define CLD_WC_ATTR_INSTALLED_OPEN_LIMIT_TILT
#define CLD_WC_ATTR_INSTALLED_CLOSED_LIMIT_TILT
#define CLD_WC_ATTR_VELOCITY_LIFT
#define CLD_WC_ATTR_ACCELERATION_TIME_LIFT
#define CLD_WC_ATTR_DECELERATION_TIME_LIFT
#define CLD_WC_ATTR_INTERMEDIATE_SETPOINTS_LIFT
#define CLD_WC_ATTR_INTERMEDIATE_SETPOINTS_TILT
```

The next set of #defines are used to enable optional commands. The options for the Window Covering cluster optional commands are added:

```
/******************************************************************************/
/*             Window Covering Cluster - Optional Commands                    */
/*                                                                            */
/* Add the following #define's to your zcl_options.h file to add optional*/
/* commands to the window covering cluster.                                   */
/******************************************************************************/
#define CLD_WC_CMD_GO_TO_LIFT_VALUE
#define CLD_WC_CMD_GO_TO_LIFT_PERCENTAGE
#define CLD_WC_CMD_GO_TO_TILT_VALUE
#define CLD_WC_CMD_GO_TO_TILT_PERCENTAGE
```

6.3.4.9 2-Common\Source\PDM_IDs.h

A new ID is added to store Scene data in PDM memory:

```
#define PDM_ID_APP_SCENES_DATA                    (0xA101)
```

6.3.4.10 2-Common\Source\app_scenes.h

This file has been copied from the JN-AN-1244 Zigbee 3.0 Light Bulb Application Note. No changes have been made to this file.

6.3.4.11 2-Common\Source\app_scenes.c

This file has been copied and adapted from the JN-AN-1244 Zigbee 3.0 Light Bulb Application Note.

It works with data from the sWindowCovering structure instead of the sLight structure (and includes app_zcl_task.h to access that structure). UART debug messages have also been added.

6.3.4.12 2-Common\Source\app.zpscfg

The Window Covering Input Cluster needs to be added to the Window Covering device's Application Endpoint in the Zigbee configuration file.

First the Window Covering Cluster needs to be made available for devices to use. Open the Profile "HA" (0x0104) entry in the tree:

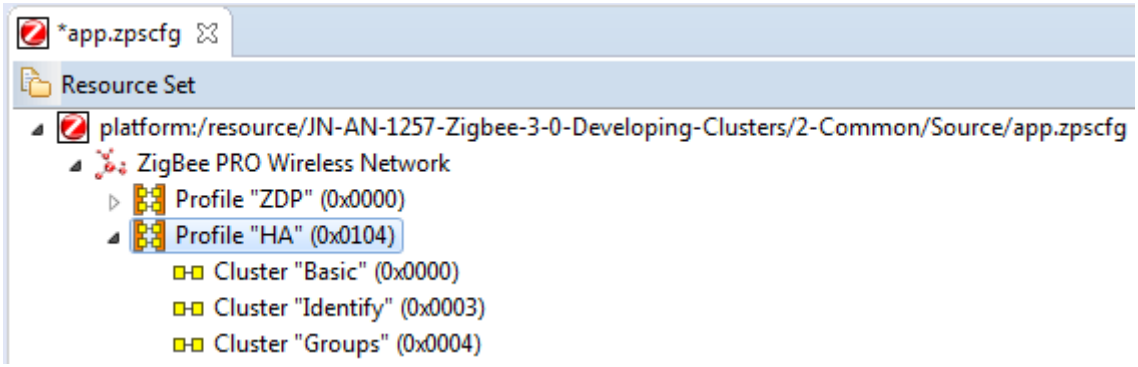

Figure 26. Open ZPS Config file

Right click the Profile "HA" (0x0104) leaf and click New Child > Cluster. Select the new cluster named <undefined> and ensure the Properties tab is selected in the lower pane. Enter 0x0102 for the Cluster ID and WindowCovering for the name:

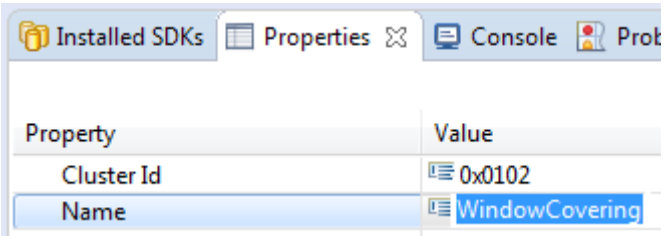

Figure 27. Edit ZPS Config File

The Window Covering Cluster should now be available under the HA Profile:

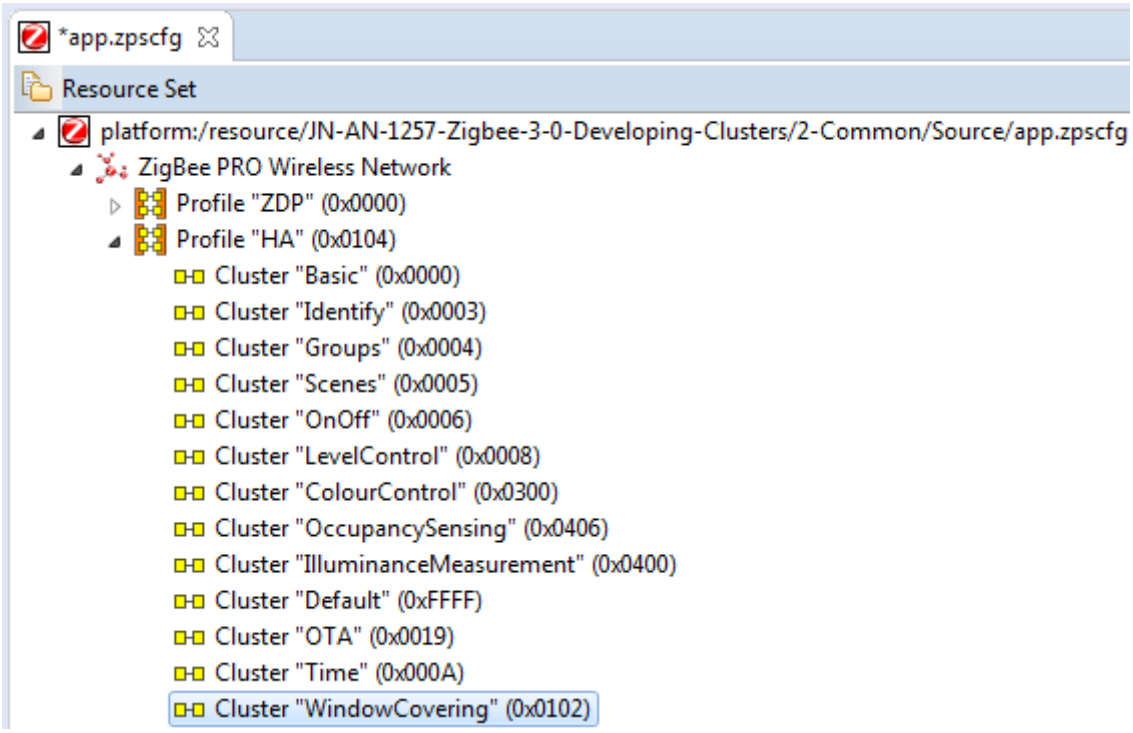

Figure 28. Edit ZPS Config file

Next the Window Covering Cluster needs to be added as an Input Cluster under the Router “Window Covering” device’s End Point “Application”:

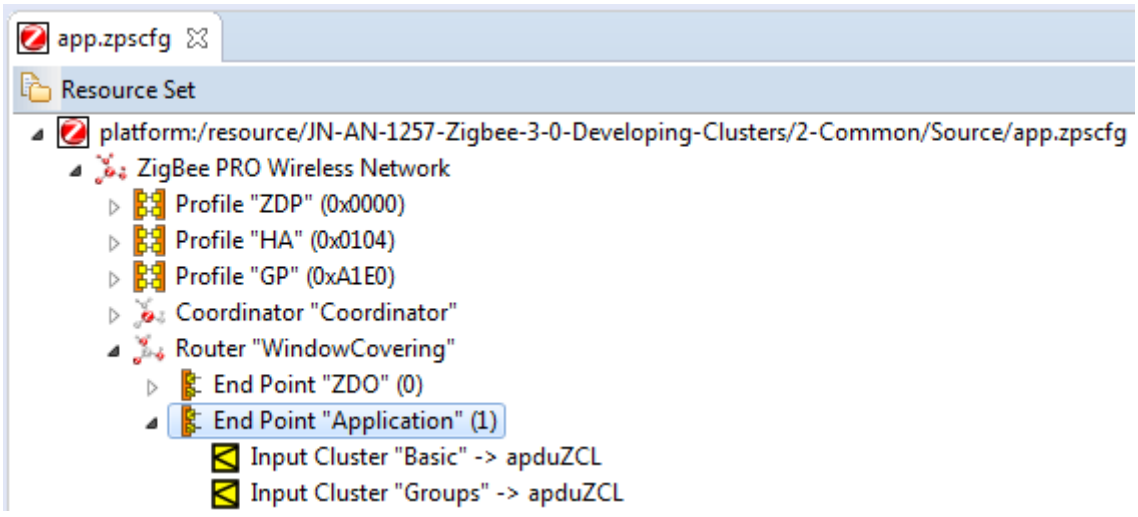

Figure 29. Edit ZPS Config File

Right click the Endpoint and select New Child > Input Cluster. Select the newly created Input Cluster named <undefined> and select the Properties tab in the lower panel. Select the Window Covering Cluster from the drop-down for the Cluster property, set the Discoverable property to true and the Receive APDU to APDU “apduZCL”:

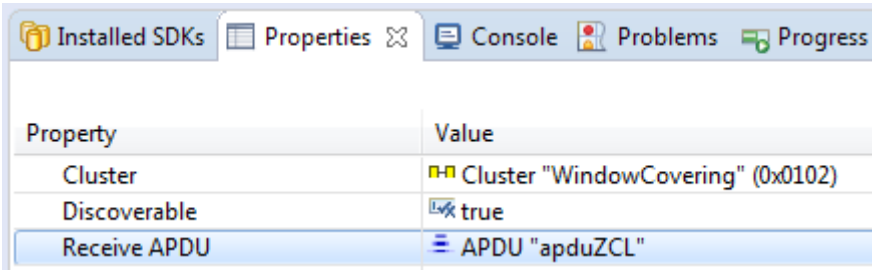

Figure 30. APDU

The Input Cluster “Window Covering” should now be displayed under End Point “Application”:

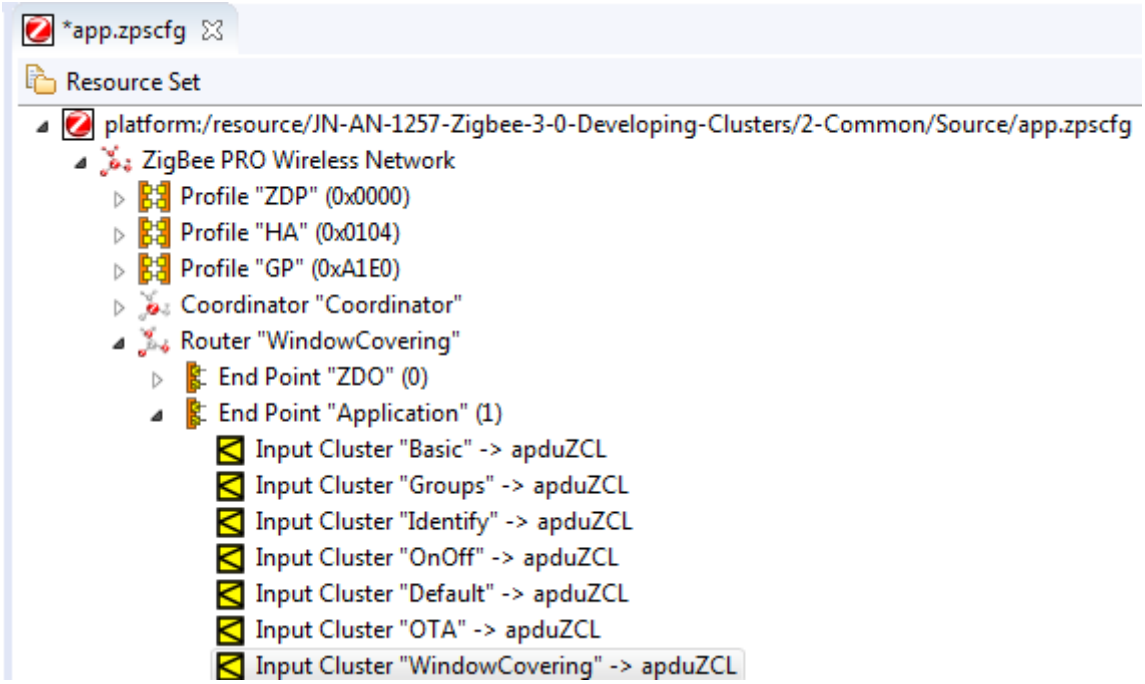

Figure 31. Add Cluster

Next select the Input Cluster “OnOff” and change the Discoverable property from true to false. (This Input Cluster could be removed but it will be left in place in case a manufacturer wishes to re-enable it.)

Finally ensure the file is saved.

6.3.4.13 2-WindowCovering\Build\ mcux\Makefile

The **2-WindowCovering\Build\ mcux\Makefile** is updated to enable debugging for the Window Covering cluster:

```
CFLAGS += -DDEBUG_CLD_WINDOW_COVERING
```

The new device, cluster and scenes files are added for compilation:

```
APPSRC += device_window_covering.c
APPSRC += cluster_window_covering.c
APPSRC += cluster_window_covering_command_handler.c
```

```
APPSRC += cluster_window_covering_commands.c
APPSRC += app_scenes.c
```

6.3.5 Step 2.5 – Testing

The step 2 Window Covering device is now complete and can be compiled and tested.

The Window Covering device can be joined to a Control Bridge using Classic Join or NFC Join (as we have that code in place from the Router).

The correct Device ID of 0x0202 (though mis-identified as a HVAC – Fan Control device by the ZGWUI) and Cluster ID of 0x0102 (unknown to the ZGWUI here) can be checked using a Simple Descriptor request to the application endpoint:

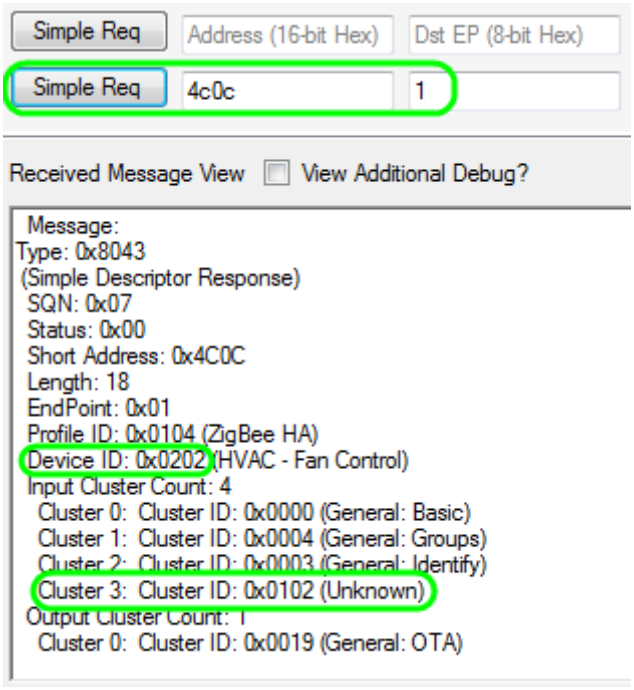

Figure 32. Simple Descriptor request

The ZGWUI can also be used to send Window Covering cluster commands to the Window Covering device using the Raw Data command on the General tab. The screen shot below shows how to issue a Down/Close command. The three bytes in the Raw Data edit-box are:

Frame Control: The commonly used value of 0x11 is used here.

Transaction ID: 0x55 is used here though any value can be used.

Cluster Command ID: 0x01 for the Down/Close command. Other values may be used for other commands, additional data for the “Go to” commands may be added after the command ID.

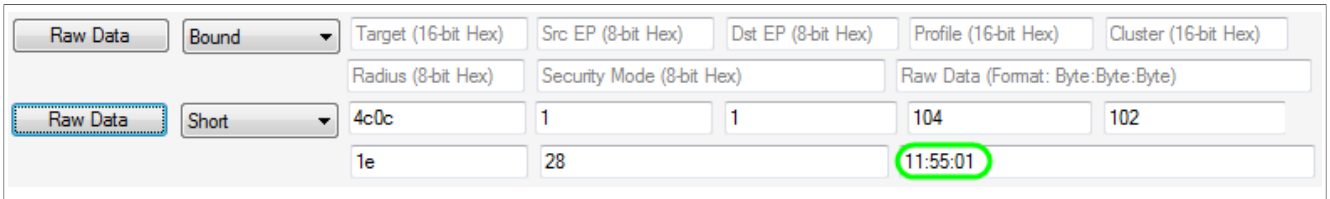

Figure 33. Raw Data Command

Sending the command shown above results in the following debug when received on the Window Covering device, the final line of the output is the application layer command handler function APP_eHandleWindowCoveringCommand() in **app_zcl_task.c** debugging the event:

```
WC: DownClose:
EP EVT: Custom Cl 0102
Window Covering: Command Id = 0x1 Down/Close
```

At this point we have a Window Covering device with the correct clusters that can receive cluster commands.

The next step is to react to the commands by driving some hardware and updating the cluster attributes as the Window Covering is opened and closed.

6.4 Step 3 – Hardware Control

With the cluster attributes and commands in place we will add code to react to commands and update attributes appropriately. For motor control 4 digital output lines will be used to drive a 4-phase stepper motor. Switches on the OM15082-2 Generic Expansion Board will be used for local control and LEDs used to indicate the operation of the motor and Window Covering modes.

The table below summarises the copied and renamed source code folders within the JN-AN-1257 Application Note to aid comparisons with difference tools:

Table 14. Source Folder

Source Folder	Destination Folder
JN-AN-1257\2-Common	JN-AN-1257\3-Common
JN-AN-1257\2-WindowCovering	JN-AN-1257\3-WindowCovering

The next table summarises new files added to the application and the original files in other Application Notes and the ZCL folder from which they are adapted to aid comparisons with difference tools:

Table 15. Source File

Source File	Destination File
None	JN-AN-1257\3-WindowCovering\Source\DriverMotor.c
None	JN-AN-1257\3-WindowCovering\Source\DriverMotor.h

The following sections go through these changes including the changes made within individual files step-by-step.

6.4.1 Step 3.1 – Folders, Makefiles and Build Configurations

- This first sub-step creates copies of the Step 2 folders for further development into Step 3. It can be skipped if you are developing in a “real-world” scenario with just a single set of source file folders.

Repeat the instructions in Step 2.1 – Folders, Makefiles and Build Configurations but this time duplicate the Step 2 setup to form a Step 3 setup.

6.4.2 Step 3.1 – Swap to OM15082-2 Generic Expansion Board

So far we have been using the OM15081-1 Lighting Expansion Board inherited from the JN-AN-1243 Router device. Only the white LED on the lighting board is used to indicate when the device is in identify mode.

Swapping to the OM15082-2 Generic Expansion Board will provide the additional LEDs we need to signal motor use and switches for input to the Window Covering device.

6.4.2.1 3-WindowCovering\Build\mcux\Makefile

The DR variable value is changed from OM15081 to OM15082 and the OTA_HW variable is also updated (this forms part of the OTA ID string:

```
DR ?= OM15082
ifeq ($(DR),OM15082)
OTA_HW = OM15082
endif
```

The contents of the DR variable are used to create a #define. The LED and button code inherited from the JN-AN-1243 Router device already contains code for the OM15082 board that will be enabled by the changed #define.

6.4.2.2 Build Configurations

The makefile will also include the value of the DR variable in output filenames. This change should be reflected in the MCUXpresso Project Build Configuration names. With the JN-AN-1257-Zigbee-3-0-Developing-Clusters project selected in the Project Explorer pane on the menu bar select Project > Build Configurations > Manage… In the window that opens select the first **3-WindowCovering** configuration then click the Rename… button. In the Rename Configuration window that opens change the OM15081 to OM15082 in the Name edit-box and click the OK button:

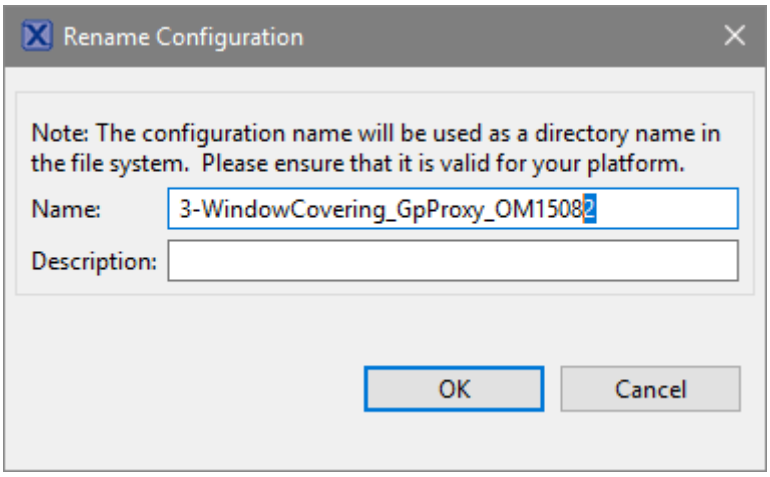

Figure 34. Rename Configuration

Repeat for the second 3-WindowCovering configuration. Finally close the Manage Configurations window by clicking the OK button.

The code can be compiled and programmed in to the device at this stage. Keep in mind that the output filename is now different. LED D1 on the OM15082 Generic Expansion Board will flash when placed into identify mode (instead of the white LED on the OM15081 Lighting Expansion Board).

6.4.3 Step 3.2 – Motor Control

This step drives hardware in response to the cluster commands, updates the read-only attributes and reacts to changes in writeable attributes.

6.4.3.1 3-Common\Source\PDM_IDs.h

As we will be storing the motor driver data across power cycles in the PDM a new PDM ID is added for this purpose:

```
#define PDM_ID_APP_MOTOR                                (0x20)
```

6.4.3.2 3-WindowCovering\Source\zcl_options.h

As we are going to implement a lift-only roller shade type device the optional attributes for tilt control and monitoring will be removed by commenting out the appropriate #defines in this module. The device will also not support the optional attributes for velocity, acceleration, deceleration and intermediate setpoints which are also commented out:

```
/******************************************************************************/
/*             Window Covering Cluster - Optional Attributes                  */
/*                                                                            */
/* Add the following #define's to your zcl_options.h file to add optional*/
/* attributes to the window covering cluster.                                 */
/******************************************************************************/
#define CLD_WC_ATTR_PHYSICAL_CLOSED_LIMIT_LIFT
//#define CLD_WC_ATTR_PHYSICAL_CLOSED_LIMIT_TILT
#define CLD_WC_ATTR_CURRENT_POSITION_LIFT
//#define CLD_WC_ATTR_CURRENT_POSITION_TILT
#define CLD_WC_ATTR_NUMBER_OF_ACTUATIONS_LIFT
//#define CLD_WC_ATTR_NUMBER_OF_ACTUATIONS_TILT
#define CLD_WC_ATTR_CURRENT_POSITION_LIFT_PERCENTAGE
//#define CLD_WC_ATTR_CURRENT_POSITION_TILT_PERCENTAGE
#define CLD_WC_ATTR_INSTALLED_OPEN_LIMIT_LIFT
#define CLD_WC_ATTR_INSTALLED_CLOSED_LIMIT_LIFT
//#define CLD_WC_ATTR_INSTALLED_OPEN_LIMIT_TILT
//#define CLD_WC_ATTR_INSTALLED_CLOSED_LIMIT_TILT
//#define CLD_WC_ATTR_VELOCITY_LIFT
//#define CLD_WC_ATTR_ACCELERATION_TIME_LIFT
//#define CLD_WC_ATTR_DECELERATION_TIME_LIFT
//#define CLD_WC_ATTR_INTERMEDIATE_SETPOINTS_LIFT
//#define CLD_WC_ATTR_INTERMEDIATE_SETPOINTS_TILT
```

The #defines for the optional tilt commands are also commented out to disable them:

```
/******************************************************************************/
/*             Window Covering Cluster - Optional Commands                    */
/*                                                                            */
/* Add the following #define's to your zcl_options.h file to add optional*/
/* commands to the window covering cluster.                                   */
/******************************************************************************/
#define CLD_WC_CMD_GO_TO_LIFT_VALUE
#define CLD_WC_CMD_GO_TO_LIFT_PERCENTAGE
//#define CLD_WC_CMD_GO_TO_TILT_VALUE
//#define CLD_WC_CMD_GO_TO_TILT_PERCENTAGE
```

6.4.3.3 3-WindowCovering\Source\DriverMotor.c/h

These two source files are added and provide an interface for driving a 4-phase stepper motor. Four output pins from the DK006 microcontroller are used to drive the motor activating a single pin at a time. The pins used are specified by #defines in **DriverMotor.h**:

```
#define DM_OUTPUT_PIN_A     18
#define DM_OUTPUT_PIN_B      2
#define DM_OUTPUT_PIN_C     11
#define DM_OUTPUT_PIN_D     10
```

The next #define sets the number of motor steps per cm, the value to be used here will be dependent upon the physical hardware used:

```
#define DM_STEPS_PER_CM    321                    /* Motor steps per cm */
```

The final two #defines are used to set the physical limits for the device and is set to 100cm here, again this is dependent upon the physical hardware used:

```
#define DM_STEP_HARD_MIN 0                       /* Hard limit for minimum step
 */
#define DM_STEP_HARD_MAX (DM_STEPS_PER_CM * 100) /* Hard limit for maximum step
 */
```

The data and functions in this module will be explained as their use is added to the existing source files.

6.4.3.4 3-WindowCovering\Source\app_main.c

As the step pins need to be pulsed regularly when driving the motor we are going to add a new ZTIMER for timing these pulses.

Towards the beginning of app_main.c the #define APP_NUM_STD_TMRS is increased from 3 to 4 to allow the addition of the new ZTIMER:

```
#define APP_NUM_STD_TMRS 4
```

Near the start of **app_main.c** a new variable should be added to hold the index of the motor control ZTIMER:

```
/****************************************************************************/
/***        Exported Variables                                            ***/
/****************************************************************************/
PUBLIC uint8 u8TimerButtonScan;
PUBLIC uint8 u8TimerZCL;
#if (defined APP_NTAG_NWK)
PUBLIC uint8 u8TimerNtag;
#endif
PUBLIC uint8 u8TimerMotor;
```

Then in the APP_vInitResources() function the new timer should be opened for use:

```
    /* Create Z timers */
```

```
    ZTIMER_eOpen(&u8TimerPowerOn,    APP_cbTimerPowerCount,
        NULL, ZTIMER_FLAG_PREVENT_SLEEP);
    ZTIMER_eOpen(&u8TimerButtonScan, APP_cbTimerButtonScan,
        NULL, ZTIMER_FLAG_PREVENT_SLEEP);
    ZTIMER_eOpen(&u8TimerZCL,        APP_cbTimerZclTick,
        NULL, ZTIMER_FLAG_PREVENT_SLEEP);
#if (defined APP_NTAG_NWK)
    ZTIMER_eOpen(&u8TimerNtag,       APP_cbNtagTimer,
        NULL, ZTIMER_FLAG_PREVENT_SLEEP);
#endif
#ifdef CLD_GREENPOWER
    ZTIMER_eOpen(&u8GPTimerTick,     APP_cbTimerGPZclTick,
        NULL, ZTIMER_FLAG_PREVENT_SLEEP);
#endif
    ZTIMER_eOpen(&u8TimerMotor,      APP_cbTimerMotor,
        NULL, ZTIMER_FLAG_PREVENT_SLEEP);
```

6.4.3.5 3-WindowCovering\Source\app_main.h

In app_main.h the u8TimerMotor variable is made available to other modules so they can make use of the ZTIMER:

```
/****************************************************************************/
/***        External Variables                                            ***/
/****************************************************************************/
extern PUBLIC uint8 u8TimerButtonScan;
extern PUBLIC uint8 u8TimerZCL;
#if (defined APP_NTAG_NWK)
extern PUBLIC uint8 u8TimerNtag;
#endif
extern PUBLIC uint8 u8TimerMotor;
```

6.4.3.6 3-WindowCovering\Source\app_window_covering.c

We will add new code to this source module to act as the middleware between the Zigbee layer and driving the motor to control the Window Covering device.

As **app_window_covering.c** will use functions in the driver module its header should be included here:

```
#include "DriverMotor.h"
```

A #define is added to set the ZTIMER period in ms:

```
#define MOTOR_TIMER_MS     3
```

A #define sets the period the motor should be stopped for before the current position is written to PDM. This avoids excessive wear on the flash memory (but may result in an unsynchronised position if the device is power cycled during this time period):

```
#define MOTOR_TIMER_PDM_MS  5000
```

Next we will add variables local to the module:

- sDM_State – this structure is passed to and used by the **DriverMotor.c** module to store the current mode and position of the motor and also to set software limits on the minimum and maximum step positions.
- eDM_Dir – is used to monitor and react to changes of the mode within the **DriverMotor.c** module.
- u32PdmMs – is used to time delay from being stopped to saving data to the PDM.
- u32Timer – is used to count timer callbacks and regularly flash LEDs if required.
- u32StepCalMin, u32StepCalMax – are used to monitor the minimum and maximum positions of the Window Covering when in calibration mode and are used to set the installation limits when leaving calibration mode.
- u8CurrentPositionLiftPercentage – is used to monitor the current position lift percentage as set by the module. If a scene is recalled the ZCL code will change the value stored in the attribute which can be detected by comparison to this local variable.

```
PRIVATE tsDM_State sDM_State; /* Motor driver state */
PRIVATE teDM_Dir   eDM_Dir;   /* Motor driver direction monitoring */
PRIVATE uint32 u32PdmMs = MOTOR_TIMER_PDM_MS;
                              /* PDM write ms delay timer */
PRIVATE uint32 u32Timer;      /* Timer counter */
PRIVATE uint32 u32StepCalMin; /* Calibration min step */
PRIVATE uint32 u32StepCalMax; /* Calibration max step */
PRIVATE uint8   u8CurrentPositionLiftPercentage;
                              /* For scene recall detection */
```

We will be adding a new function to handle the motor initialisation. In **app_window_covering.c** in the APP_vInitialiseRouter() function we will add this initialisation call after initialising LEDs and buttons:

```
/* Initialise LEDs and buttons */
APP_vLedInitialise();
APP_bButtonInitialise();
APP_vMotorInitialise();
```

In the APP_taskRouter() function case statements for the switch activations on the OM15082-2 Generic Expansion Board. These activations call additional new functions in **app_window_covering.c** to provide local control of the device as shown in the following table:

Table 16.

Button	Action	Function
SW1 Down	Up/Open	APP_vMotorBackward()
SW1 Up	Stop	APP_vMotorStop()
SW2 Down	Calibration Mode Toggle	APP_vMotorToggleModeBits()
SW3 Down	Maintenance Mode Toggle	APP_vMotorToggleModeBits
SW4 Down	Down/Close	APP_vMotorForward()
SW4 Up	Stop	APP_vMotorStop()

The new APP_vMotorInitialise() function is added. This function performs the following tasks:

- Initialises the local eDM_Dir variable used to monitor the motor driver's internal mode and react to changes.
- Attempts to read data from the PDM into the sDM_State structure, if unsuccessful the structure members are set to the default values.
- Initialises the motor driver using a call to vDM_Initialise().
- Initialises enabled Window Covering cluster attribute values converting from the step data held in the sDM_State structure.

- Starts the motor ZTIMER.

Two new functions, APP_vMotorForward() and APP_vMotorBackward(), are added to begin the process of starting to operate the motor. These functions in turn use the eDM_Forward() and eDM_Backward functions in **DriverMotor.c** to operate the motor.

The new function APP_vMotorStop() is also added to stop turning the motor, the Window Covering cluster attributes for the current position and the number of activations are updated here. This function uses the eDM_Stop() function in **DriverMotor.c**.

The new functions APP_bMotorGoToLiftValue() and APP_bMotorGoToLiftPercentage() are used to move the Window Covering to a specific position converting the passed in values to steps. If the position is outside the allowed range a failure code is returned. These functions use the eDM_GoToTarget() function in **DriverMotor.c**.

The new function APP_vMotorUpdateModeFromAttribute() reacts to changes in bits in the Mode attribute variable if they are different from that stored in the local sDM_State structure:

- When entering calibration mode the local min and max values are set to the current position.
- When exiting calibration mode if the recorded local min and max values are 10cm or more apart the measured values are applied as the soft limits and the Open Limit Lift, Closed Limit Lift and Current Position Lift Percentage attributes are updated.
- Changes to the Mode are immediately saved to PDM to ensure the Maintenance mode (when the motor is not allowed to operate) flag is retained for safety.

The new function APP_vMotorToggleModeBits() can be used to toggle a mask of bits in the Mode attribute and apply them. This function is used to react to the Mode button presses on SW2 and SW3.

The new function APP_vMotorClusterUpdate() checks whether the Current Position Lift Percentage attribute has been changed due to the device receiving a Scene Recall command. The APP_bMotorGoToLiftPercentage() function is used to move to the correct position (if required).

The new APP_cbTimerMotor() ZTIMER callback function is added. This function performs the following tasks:

- Calls the motor driver tick function eMD_Tick()
- When the motor driver has stopped moving (due to reaching a limit) the current position and actuation Window Covering cluster attributes are updated.
- When the motor driver is stopped and in calibration mode the limits are updated from the current position if required.
- When the motor is stopped for the PDM delay period the state structure is written to PDM.
- The LEDs are operated depending on the mode and status of the Window Covering.
- The timer is started again for the next iteration.

In the existing APP_vFactoryResetRecords() the motor control PDM record is deleted.

6.4.3.7 3-WindowCovering\Source\app_window_covering.h

The prototypes for the new public functions described above are added:

```
PUBLIC void   APP_vMotorInitialise(void);
PUBLIC void   APP_vMotorForward(void);
PUBLIC void   APP_vMotorBackward(void);
PUBLIC void   APP_vMotorStop(void);
PUBLIC bool_t APP_bMotorGoToLiftValue(uint16 u16LiftValue);
PUBLIC bool_t APP_bMotorGoToLiftPercentage(uint8 u8LiftPercentage);
PUBLIC void   APP_vMotorUpdateModeFromAttribute(void);
PUBLIC void   APP_vMotorToggleModeBits(uint8 u8ModeMask);
PUBLIC void   APP_vMotorClusterUpdate(void);
```

```
PUBLIC void   APP_cbTimerMotor(void *pvParams);
```

6.4.3.8 3-WindowCovering\Source\app_zcl_task.c

In the ZPP_ZCL_cbEndPointCallback() function code is added to the E_ZCL_WRITE_INDIVIDUAL_ATTRIBUTE case statement to react to a remote change to the Mode attribute by calling APP_vMotorUpdateModeFromAttribute() in **app_window_covering.c**:

```
case E_ZCL_CBET_WRITE_INDIVIDUAL_ATTRIBUTE:
    DBG_vPrintf(TRACE_ZCL,
        "\r\nEP EVT: Write Individual Attribute Status = %d",
        psEvent->eZCL_Status);
    /* Is this a successful write to the Window Covering cluster's
       Mode attribute? */
    if (psEvent->eZCL_Status == E_ZCL_SUCCESS
    &&  psEvent->uMessage.sIndividualAttributeResponse.u16AttributeEnum ==
        E_CLD_WC_ATTR_ID_MODE
    &&  psEvent->uMessage.sIndividualAttributeResponse.eAttributeStatus ==
        E_ZCL_CMDS_SUCCESS
    &&  psEvent->psClusterInstance->psClusterDefinition->u16ClusterEnum ==
        CLUSTER_ID_WINDOW_COVERING)
    {
        /* Update window covering motor control */
        APP_vMotorUpdateModeFromAttribute();
    }
break;
```

The APP_eHandleWindowCoveringCommand() function is updated to call the appropriate functions in **app_window_covering.c** for the Up/Open, Down/Close, Stop, Go To Lift Value and Go To Lift Percentage commands (not shown).

The function APP_vHandleClusterUpdate() is called whenever cluster contents are changed, including when a Scene Recall command is received. Code is added to call APP_vMotorClusterUpdate() when the Window Covering cluster is updated in this way which will result in the Window Covering moving to the correct position upon receiving a Scene Recall command:

```
/* Window covering cluster ? */
else if (psEvent->psClusterInstance->psClusterDefinition->u16ClusterEnum ==
    CLUSTER_ID_WINDOW_COVERING)
{
    /* Check for possible scene change and react */
    APP_vMotorClusterUpdate();
}
```

The LED that was being used for identification is now being used as one of the outputs to the stepper motor. To swap it to the unused LED DS2 on the OM15076 Carrier Board make the following replacement throughout the whole of **app_zcl_task.c**:

- “LED1” to “LED4”

6.4.3.9 3-WindowCovering\Build\mcux\Makefile

The motor driver source file is added to the compiled files:

```
APPSRC += DriverMotor.c
```

6.4.4 Step 3.3 – Testing

The step 3 Window Covering device is now complete and can be compiled and tested. It should operate as described in detail in Running the Demonstration Application.

7 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 Zigbee 3.0 applications for DK006 microcontrollers. They are available free-of-charge to authorised users via the MCUXpresso web site.

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.

This section provides an overview of the source code in the Application Note not specifically covered in the section Developing the Window Covering.

7.1 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
```

7.2 Common Code

This section lists and describes the source files that provide functionality common to all the devices in this Application Note and are held in the **3-Common/source** directory for the final version of the Window Covering.

app_zpscfg is a configuration file for the Zigbee stack. For each of the devices in the application, it defines all the required stack parameters, table sizes, servers etc. This file is processed as part of the build process and creates device-specific source files to be built into each device.

app_buttons.c provides an interface to read the switches/buttons on the expansion boards and to post button-press events to the application event queue.

app_events.h contains type definitions of the application events.

app_leds.c provides an interface to the LEDs on the expansion boards.

app_ntag.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_ota_client.c contains the application code that drives the OTA process.

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.

app_scenes.c provides functions used by the Scenes cluster to store and retrieve scene settings from PDM.

cluster_window_covering.c, cluster_window_covering_commands_handler.c, cluster_window_covering_commands.c provide the implementation of the Window Covering cluster.

device_window_covering.c provides the ZCL implementation of the Window Covering device.

PDM_IDs.h provides unique identifiers for all the data records in the PDM.

7.3 NFC Folder

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

7.4 Window Covering Application Code

This section lists and describes the source files for the Window Covering application code, which are provided in the **Source** directory for the application. You may wish to use this code as a basis for your own application development. You can rebuild your customised application as described in Rebuilding the Applications.

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

app_reporting.c configures the attribute reporting in the application.

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

app_window_covering.c hosts the event handlers for the application and the Base Device callback. The Base Device callback receives Base Device Events and AF Stack events after the Base Device has completed any processing that is required. These events can then be further processed by the application. The events include data indications that are passed to the ZCL for processing and network management events (such a ‘network formed’ and ‘new node has joined’) in order to keep the application informed of the network state. The application event queue is processed to receive button-press events.

This modules also acts as middleware between the Zigbee part of the application and the motor control part of the application translating Zigbee Window Covering events to motor control commands.

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

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

DriverMotor.c provides APIs to control the motor hardware, to adapt the application to drive different motor types the contents of these API functions can be updated.

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

7.4.1 Command Line Build Options

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

- `ICODES=0` to build so that install codes are not used
- `ICODES=1` to build so that install codes are used
- `APP_NTAG_NWK=1` to build with NTAG/NFC joining support. **NtagNwk** is included in output filenames when using this setting.
- `OTA=1` to build with OTA client. **Ota** is included in output filenames when using this setting.
- `OTA_ENCRYPTED=1` to build OTA images with encryption. **OtaEnc** is included in output filenames when using this setting.

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]
- Zigbee 3.0 Green Power User Guide [JN-UG-3134]
- Encryption Tool User Guide [JN-UG-3135]
- Zigbee Cluster Library Specification [07-5123-07] (from Zigbee.org)

9 Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2023 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

10 Revision History

Table 17. Revision History

Document ID	Release date	Description
AN1257 v.2.0	20200311	New version v2000 built with JN5189 2.6.0 SDK
AN1257 v.2.1	20200417	New release v2001 built with K32W061 RFP RC5
AN1257 v.2.2	20200821	New release v2002 built with K32W061 2.6.1 MR1
AN1257 v.2.3	20201223	New release v2003 built with K32W061 2.6.2 MR2
AN1257 v.2.4	20210420	New release v2004 Update to include K32W041AM for MR3 SDK
AN1257 v.2.5	20210906	New release v2005 built on MR3 QPATCH1 SDK
AN1257 v.2.6	20220320	New release v2006 built on MR4 SDK2.6.5
AN1257 v.2.7	20221111	New release v2007 built on SDK2.6.8 SDK
AN1257 v.2.8	20231107	New release v2008 built on SDK2.6.13 SDK

Legal information

Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

Disclaimers

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.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at https://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

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.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

Bluetooth — the Bluetooth wordmark and logos are registered trademarks owned by Bluetooth SIG, Inc. and any use of such marks by NXP Semiconductors is under license.

EDGEFAST — is a trademark of NXP B.V.

i.MX — is a trademark of NXP B.V.

Tables

Figures

Contents

For more information, please visit: https://www.nxp.com

Date of release: 12 December 2023
Document identifier: AN1257