**Application Note**

# ZigBee 3.0 IoT Control Bridge

The NXP ZigBee 3.0 IoT Control Bridge provides a means of controlling ZigBee devices via a serial link which is connected to a host controller. The IoT Control Bridge supports ZigBee Lighting & Occupancy (ZLO) devices, controlling the network by mostly client cluster commands, and runs on the NXP Kinetis MKW41Z wireless microcontrollers.

This guide provides information to allow users to connect to the Control Bridge using a Graphical User Interface (GUI), which simulates a host, to operate the ZigBee network. It also describes the serial protocol used to interface with the Control Bridge, as well as the payloads of all relevant commands and responses.

# 1 Application Note Overview

This Application Note is concerned with a ZigBee 3.0 Control Bridge device implemented as a serial device. This device would typically form the ZigBee side of an IoT Gateway. The Application Note shows how the ZigBee Control Bridge can be controlled by an application running on a PC. It also demonstrates the different commands that can be sent in the payload that the ZigBee Control Bridge requires. The demonstration described in this guide uses the hardware found in the FRDM-KW41Z: NXP® Freedom Development Kit for Kinetis.

This guide is intended to show how to set up and use the Control Bridge in a simple demonstration network of ZigBee Lighting & Occupancy (ZLO) devices, in order to familiarise users with the functions available to a Gateway host. This is done by using the ZigBee Gateway Graphical User Interface (ZGWUI) to interact with the Control Bridge to manage the network and the devices. The ZGWUI is a C# application that acts as a PC host that communicates serially with the Kinetis KW41Z-based Control Bridge. The firmware used in this Application Note is supplied as source code to allow customisation. Firmware for the ZigBee devices to be controlled can be built from the Application Note *ZigBee 3.0 Light Bulbs* and other NXP *ZigBee 3.0 Application Notes*.

# 2 Capabilities

This Application Note is designed for use with the following NXP hardware and software:

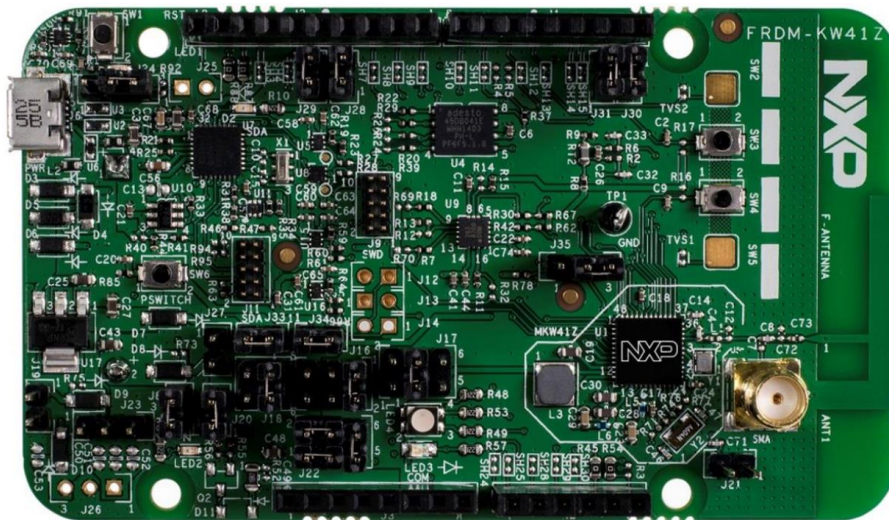| Product Type | Part Number |
|---|---|
| Evaluation/Development Kit | Kinetis  FRDM-KW41Z |
| IAR EWARM Toolchain | Kinetis  FRDM-KW41Z |
| MCUXpresso IDE | Kinetis FRDM-KW41Z |

**Figure 1 FRDM KW41Z Evaluation board**

The ZigBee Gateway can be used to control ZigBee 3.0 network nodes based on the ZigBee Lighting & Occupancy (ZLO) devices. However, for backwards compatibility, it can also be used to control devices from the former ZigBee Light Link and Home Automation profiles.

The main purpose of this Application Note is to provide a Kinetis FRDM-KW41Z slave application that receives various commands to control nodes within a ZigBee network. This allows a master (normally a host) to bridge into a ZigBee network while servicing IPv6 devices or other protocols.

The ZGWUI is provided in this Application Note as a way demonstrating all the different features that the Kinetis KW41Z Control Bridge supports.

# 3 What is Provided

The demonstration package comes with the following components, intended to be used with FRDM-KW41Z boards as wireless nodes:

- Documentation (this document)
- Application binary and source code for the following:
    - ZigBee Control Bridge
- Application binary for the following:
    - ZigBee Gateway Graphical User Interface (ZGWUI)

Although in most cases the ZigBee Control Bridge can be used "as is", developers may want to add extra functionality or even add application-specific behaviour.

To run the demonstration, application binaries are also required for the network nodes:

- Dimmable Light (**dimmable_light_frdmkw41z.bin**)
- Extended Colour Light (**extended_color_light_frdmkw41z.bin**)
- Colour Temperature Light (**color_temperature_light_frdmkw41z.bin**)

These binaries are provided in the Application Note *ZigBee 3.0 Light Bulbs* and must be loaded onto FRDM-KW41Z boards.

# 4 Running the Demonstration

## 4.1 Programming the Kinetis KW41Z Device

Each binary name provides details of the chip variant, supported device type for the Control Bridge – for example:

- **control_bridge_frdmkw41z.bin** is the Control Bridge binary which is built for the Kinetis FRDM-KW41Z chip (default configuration: Coordinator device type, supports a 115200 baud rate).

To run the demonstration, the ZigBee Control Bridge binary will need to be programmed into KW41Z Freedom board. For Kinetis KW41Z devices, this can be done from the IAR EW development platform or from MCUXpresso IDE

By default, the firmware uses the Kinetis KW41Z LPUART0 to communicate with the host.

## 4.2 Running the ZGWUI

The ZGWUI is a C# application integrated in the Test Tool, that was developed to allow a ZigBee network to be easily set up and run without needing any special knowledge. Below is a screenshot of the application. The sections that follow explain how to demonstrate the common functionality of the ZGWUI.
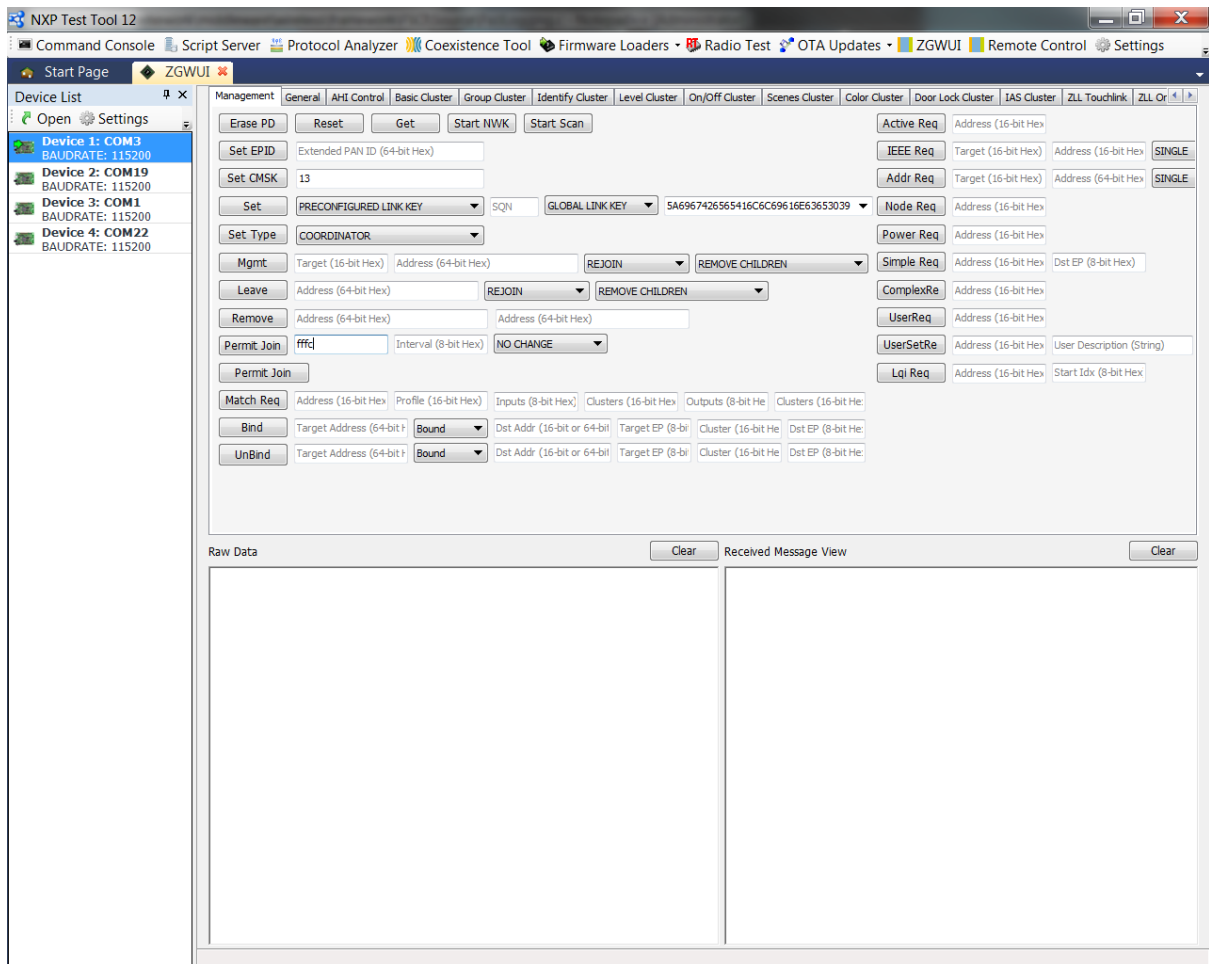
## 4.2.1 Connecting to the Control Bridge

In order to connect to the Control Bridge and issue commands to communicate with ZigBee devices, a serial connection must be set up and opened via the Test Tool. To do this, click on **Settings** towards the top-left of the interface, select the correct serial port, configure the baud rate to 115200, leave all the other settings as default and click **OK to apply serial port configuration**.
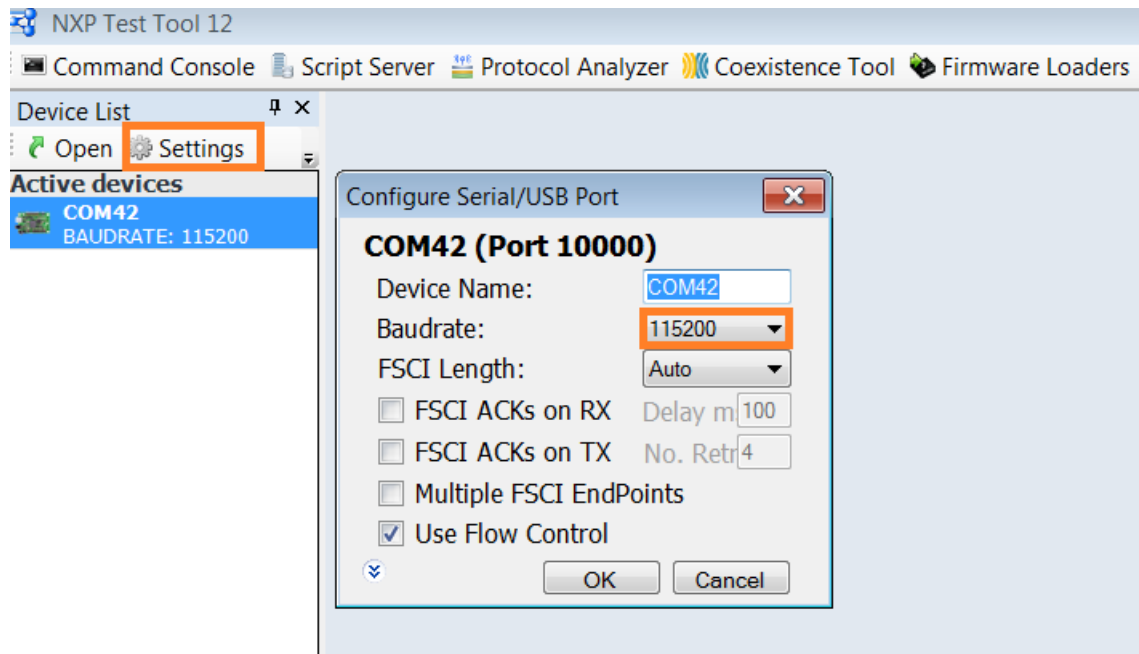


On top-left click **Open** or double-click the COM device to connect to the Control Bridge application

## 4.2.2 Configuring and Starting a Network

Before initiating a network, some network configuration needs to be done - certain commands need to be run before the network is started, as described below. The description assumes that classical joining will be used to form the network.

In this case, the Control Bridge starts as a Coordinator and allows devices into the network via MAC association. Before you start the network, there are basic commands that can be optionally issued to create a customised network.

The two commands that can be sent are "Set Channel Mask" and "Set Extended PAN ID". The "Set Channel Mask" command informs the Control Bridge which channels the network can start on. The Control Bridge will then chose the best channel available. The **Set CMSK** textbox can be used to specify either a hexadecimal value for a channel mask of possible channels or a decimal channel number if a fixed channel is to be used. The "Set Channel Mask" command can then be issued by clicking the **Set CMSK** button.
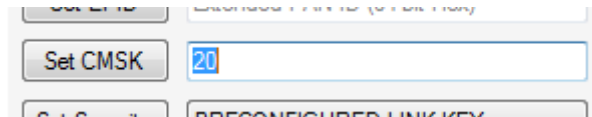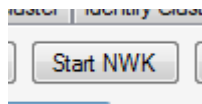
*Indicates the network is to start on channel 20*

The **Set EPID** textbox can be used to enter a pre-defined Extended PAN ID (EPID) as a 64-bit hexadecimal value. The "Set Extended PAN ID" command can then be issued by clicking the **Set EPID** button.
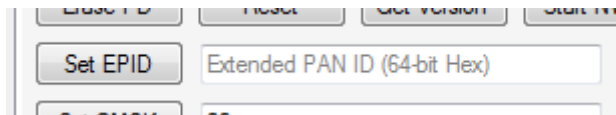


Once the network has been configured, it can be started. This is done by pressing the **Start NWK** button.



You will receive two messages back which will appear in the **Received Message View** pane in the bottom-right of the interface. The first will indicate a successful execution of the start network command and the second will indicate that the network has been formed, with information about the network parameters.
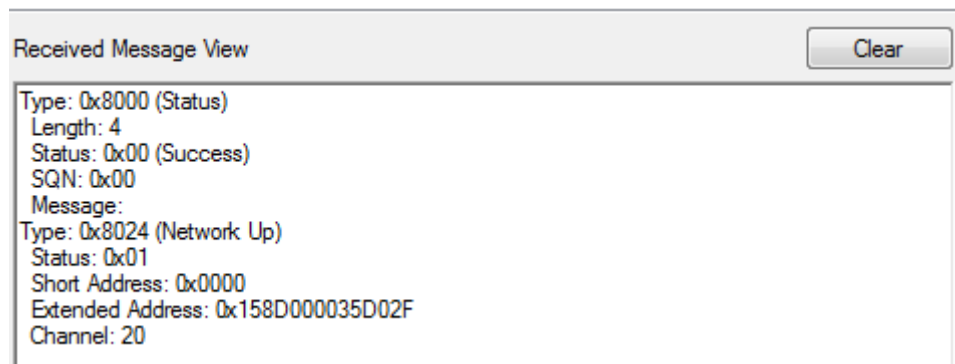


## 4.2.3 Setting up the Nodes

### 4.2.3.1 Programming the ZigBee Device Binaries

Depending on which type of device and ZigBee network configuration you are demonstrating, you will need to program each light board with the an application binary – one of:

- **dimmable_light_frdmkw41z.bin** (Dimmable Light)

- **extended_color_light_frdmkw41z.bin** (Extended Colour Light)

- **color_temperature_light_frdmkw41z.bin** (Colour Temperature Light)

These binaries are supplied in the *KW41Z_ZigBee_3.0_Software_v6.0.6*. They must be programmed into the devices through the USB connector using the on-board OpenSDA chip with JLink or CMSIS-DAP firmware or an external debug probe (for example Segger JLink) connected to the JTAG connector.

## 4.2.4 Joining Nodes to the Network

To successfully join a node to the network, a network must be started and 'permit join' must be enabled on the network node(s) that other devices will join. In the first (left) **Permit Join** textbox, enter the address of the node on which you wish to allow joining (normally 0x0000 for the Coordinator or 0xFFFC for all Router/Coordinator nodes). In the second (right) **Permit Join** textbox, enter the length of time in seconds for which you require 'permit join' to be active. Both values must be entered in hexadecimal. Click the **Permit Join** button to enable 'permit join' on the specified node(s).
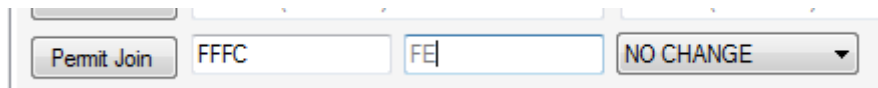


***Broadcast to all Router/Coordinator devices to allow joining for 254 seconds.***

Press the **SW3** switch on the FRDM-KW41Z board programmed with dimmable light app.
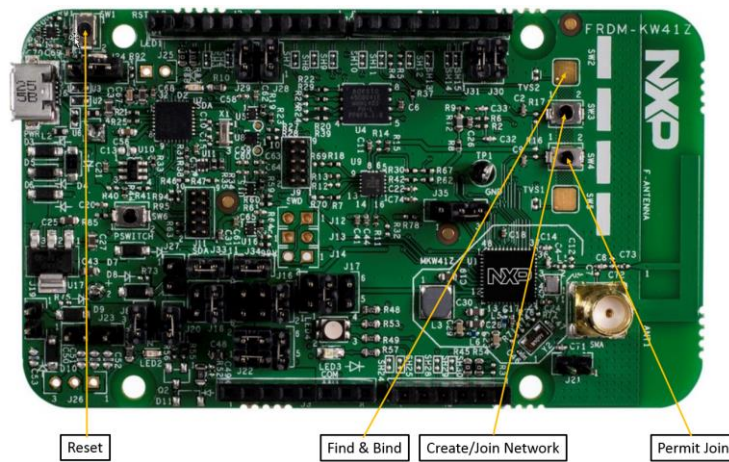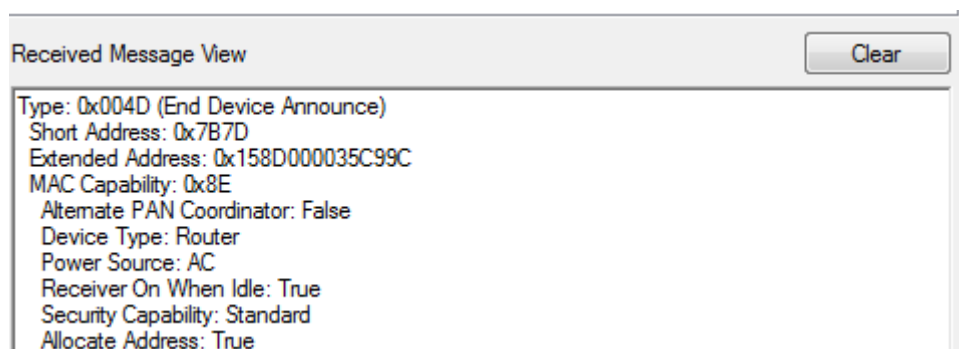


**Figure 2 FRDM-KW41Z button function mapping**

When a device joins the network, it will send out a Device Announce message which is captured in the **Received Message View** pane.

## 4.2.5 Controlling Devices

In this example, it is assumed that you have joined a Dimmable Light device to the network. A Dimmable Light device supports the On/Off and Level Control clusters that are used to modify the lighting characteristics of the bulb.

### 4.2.5.1 On/Off Cluster

Switching a light on or off is done using a command in the ZGWUI that has various attributes added.

Click on the **On/Off Cluster** tab along the top of the interface.

| Management | General | Basic Cluster | Group Cluster | Identify Cluster | Level Cluster | On/Off Cluster | Scenes Cluster | Color Cluster | Doc |
|---|---|---|---|---|---|---|---|---|---|
| OnOff | Bound Addr ▼ | Address (16-bit Hex) | Src EP (8-bit Hex) | | Dst EP (8-bit Hex) | Off ▼ | | | |

Select the address mode that you would like to use. Then in the three textboxes, enter the 16-bit network address of the node you want to control, the source endpoint number and the destination endpoint number (all in hexadecimal). Finally, select the type of "On/Off" command that you want to send.

| Management | General | Basic Cluster | Group Cluster | Identify Cluster | Level Cluster | On/Off Cluster | Scenes Cluster | Color Cluster | Dc |
|---|---|---|---|---|---|---|---|---|---|
| OnOff | Short Addr ▼ | 7B7D | 1 | | 1 | Toggle ▼ | | | |

The light will change its on/off state and a Default Response message will be received in the **Received Message View** pane. The Default Response confirms that a device received the "On/Off" command and processed the command. If the command was not sent via unicast, a Default Response will not be received.

```
Received Message View                    [ Clear ]

Type: 0x8000 (Status)
  Length: 4
  Status: 0x00 (Success)
  SQN: 0x03
  Message:
Type: 0x8101 (Default Response)
  SQN: 0x03
  EndPoint: 0x01
  Cluster ID: 0x0006 (General: On/Off)
  Command: 0x02
  Status: 0x00
```

### 4.2.5.2 Level Control Cluster

The Level Control cluster allows a bulb's dimmable light level to be set to a specific value. This value can be between 2 and 254 (inclusive), and can be set on the **Level Cluster** tab.



There are a number of attributes that can be passed to the Control Bridge as part of the Level Control cluster's "Move To Level" command:

- Addressing mode

- Hexadecimal destination address

- Source endpoint

- Destination endpoint

- With/without On/Off (indicates whether to modify On/Off state with Level Control)

- Hexadecimal level value

- Hexadecimal transition time (in tenths of a second)

These attributes appear (in the above order) on the **MoveToLevel** line in the interface:



The command is sent by clicking the **MoveToLevel** button. After sending this command with the above attribute values, the destination light will dim to the lowest level with a 1-second transition. A Default Response will be received in the **Received Message View** pane to indicate that the command was processed.

## 4.2.6 Managing Groups

In the ZGWUI, there are several commands available to manage groups and the devices that are members of these groups. All group commands are listed in the **Group Cluster** tab.



### 4.2.6.1 Add Group

You can add a device to a group by sending an "Add Group" command to the device, in order to add the relevant group ID into the device's Group Address table. This is done in the **Add Group** line of the interface by entering the network address of the device, source endpoint number, destination endpoint number and user-defined Group ID, and then clicking the **Add Group** button



An Add Group Response is then displayed in the **Received Message View** pane with the Group ID and the status of the command.



To verify that this group has been added, try sending an "On/Off" command with the group address you have just added. This will toggle the on/off state of the light. Note that since this is a groupcast, a Default Response will not be received.
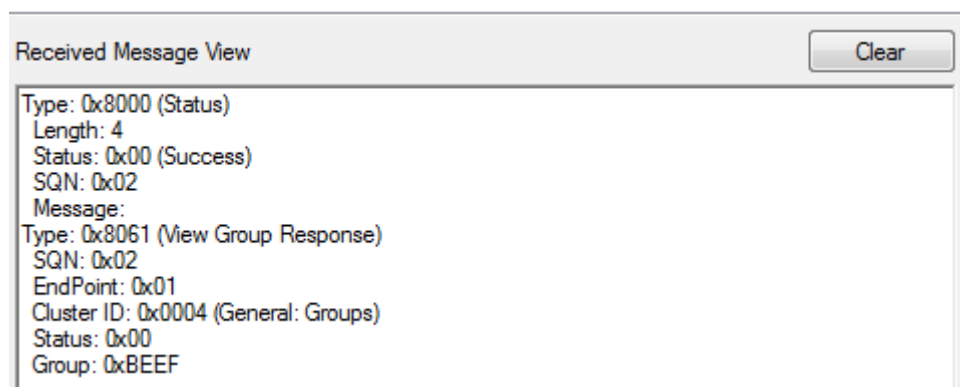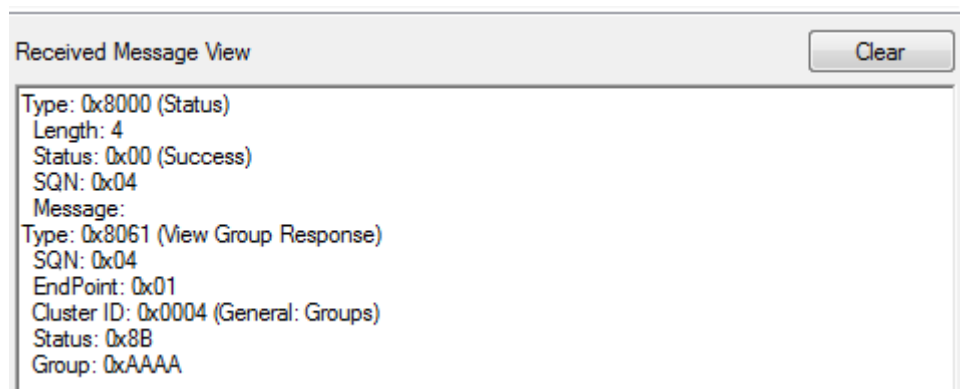
### 4.2.6.2 View Group

You can find out whether a device is a member of a specific group by sending a "View Group" command to the device. This is done in the **View Group** line of the interface by entering the network address of the device, source endpoint number, destination endpoint number and Group ID of the relevant group, and then clicking the **View Group** button.



If the device is a member of that group, you will receive a View Group Response with a status of "Success" (0x00).



If the device is not a member of that group, you will receive a View Group Response with a status of "Not Found" (0x8B).



### 4.2.6.3 Get Group Membership

You can find out which groups a specific device is a member of by sending a "Get Group Membership" command to the device. This is done in the **Get Group** line of the interface by entering the network address of the device, source endpoint number, destination endpoint number and group count (number of groups you want to look for), and then clicking the **Get Group** button.

If the device is a member of any groups, it will respond with the number of groups and the group addresses of the groups to which it belongs.



If the device is not a member of any groups, it will respond with an empty group list with a count of 0.



### 4.2.6.4 Remove Group

You can remove a group from a device's Group Address table by sending a "Remove Group" command to the device. This is done in the **Remove Grp** line of the interface by entering the network address of the device, source endpoint number, destination endpoint number and the relevant Group ID, and then clicking the **Remove Grp** button.

If the device is a member of the group that you are trying to remove then it will respond with a status of "Success" (0x00).



```
Received Message View                          Clear

Type: 0x8000 (Status)
  Length: 4
  Status: 0x00 (Success)
  SQN: 0x07
  Message:
Type: 0x8063 (Remove Group Response)
  SQN: 0x07
  EndPoint: 0x01
  Cluster ID: 0x0004 (General: Groups)
  Status: 0x00
  Group: 0xAAAA
```
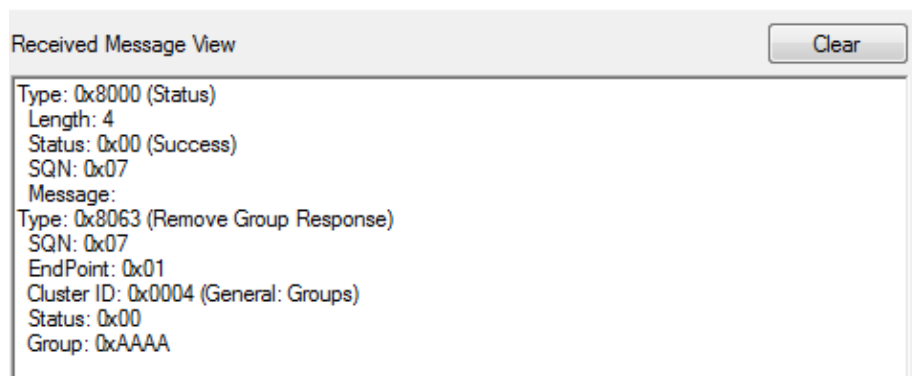
If the group does not exist on the device, it will respond with a status of "Not Found" (0x8B).

```
Received Message View                          Clear

Type: 0x8000 (Status)
  Length: 4
  Status: 0x00 (Success)
  SQN: 0x08
  Message:
Type: 0x8063 (Remove Group Response)
  SQN: 0x08
  EndPoint: 0x01
  Cluster ID: 0x0004 (General: Groups)
  Status: 0x8B
  Group: 0xBEEF
```
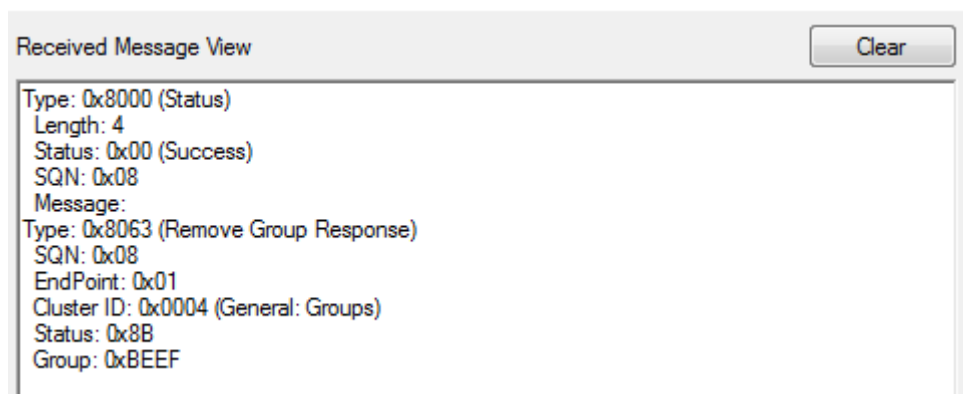
### 4.2.6.5 Remove All Groups

You can remove a device from all groups by sending the "Remove All Groups" command to the device. This is done in the **Remove All** line of the interface by entering the network address of the device, source endpoint number and destination endpoint number, and then clicking the **Remove All** button.

```
Remove All    42C3        1              1
```

Irrespective of whether the device is associated with any groups, it will always respond with a status of "Success" (0x00).
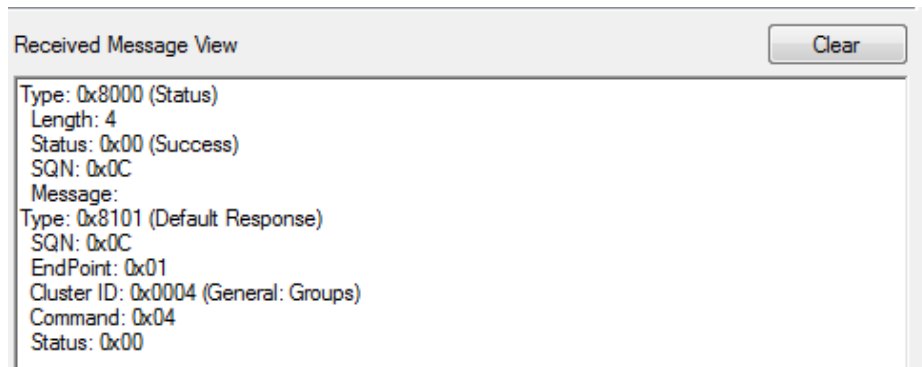
### 4.2.6.6 Add Group If Identifying

You can attempt to add a device to a group if the device has been put into Identify mode by sending the "Add Group If Identifying" command to the device. This is done in the **Add If Ident** line of the interface by entering the network address of the device, source endpoint number, destination endpoint number and the Group ID to be allocated, and then clicking the **Add If Ident** button.
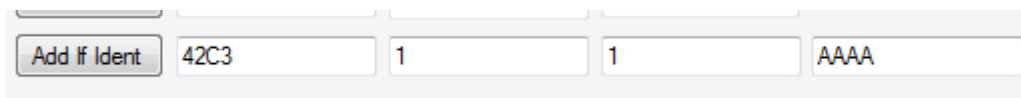


This command does not send a response back to the host, but you can perform a send "Get Group Membership" command to verify that device is a member of the group.

## 4.2.7 Managing Scenes

In the ZGWUI, there are several commands available to manage scenes and the devices that participate in these scenes. All scene commands are listed in the **Scenes Cluster** tab. To be able to use a scene command, the target device must be a member of a group with an associated scene.

### 4.2.7.1 Store Scene

The "Store Scene" command instructs a device to save its current state in a scene (new or existing). This is done in the **Store Scene** line of the interface by entering the addressing mode, address of the device, source endpoint number, destination endpoint number, Group ID and Scene ID, and then clicking the **Store Scene** button.
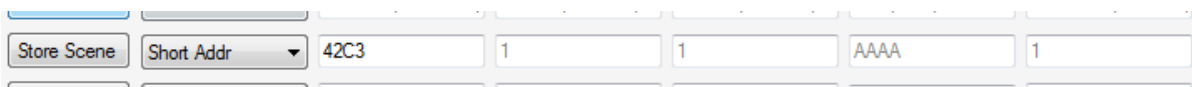


This results in the following "Store Scene Response" command which is displayed in the **Received Message View** pane.
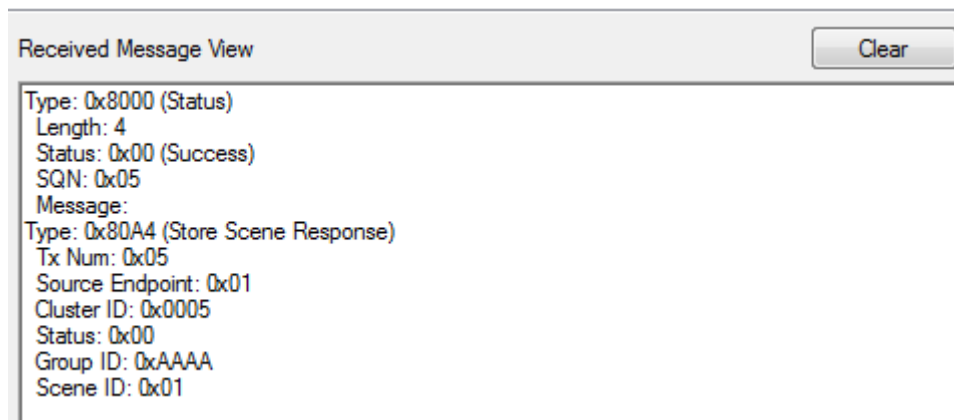
The above output indicates that the device state has been successfully stored in the scene with Scene ID 0x01 associated with the group with Group ID 0xAAAA

### 4.2.7.2 Recall Scene

The "Recall Scene" command instructs a device to restore a previously saved scene in the device - for a light bulb, this could be restoring an on/off or level state. This is done in the **Recall Scn** line of the interface by entering the addressing mode, address of the device, source endpoint number, destination endpoint number, Group ID and Scene ID, and then clicking the **Recall Scn** button.
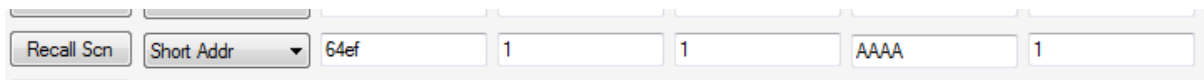


When the command is sent, a response will appear in the **Received Message View** pane indicating whether the command has been successful.



### 4.2.7.3 View Scene

You can view the details of a scene (e.g. on/off state, level) on a device by sending a "View Scene" command to the device. This is done in the **View Scene** line of the interface by entering the addressing mode, address of the device, source endpoint number, destination endpoint number, Group ID and Scene ID, and then clicking the **View Scene** button.



After sending a successful "View Scene" command, a response containing vital information like Transition time, Scene Name Length, Scene Name and Scene Data will be displayed in the **Received Message View** pane.
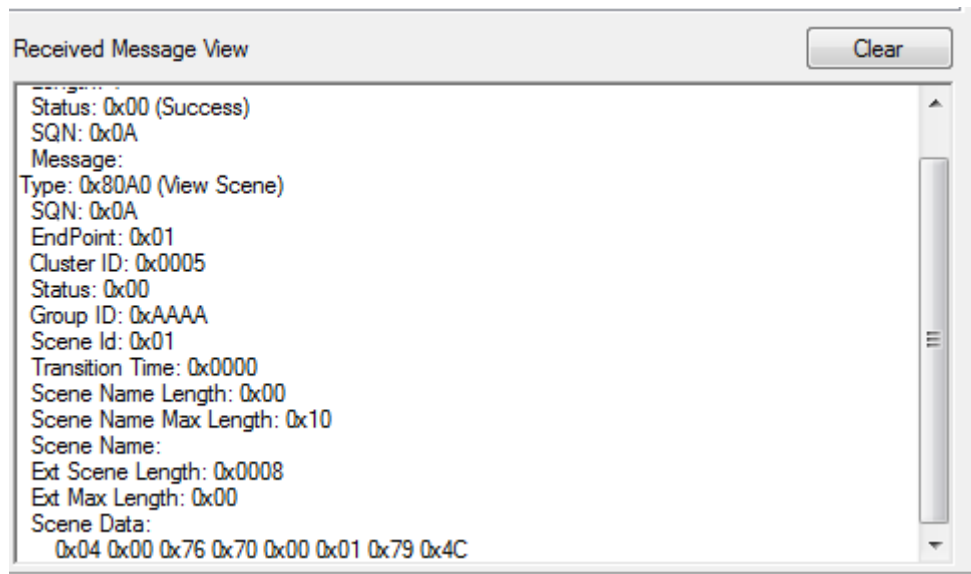
### 4.2.7.4 Get Scene Membership

You can find out which scenes associated with a particular group are available on a device by sending a "Get Scene Membership" command to the device. This is done in the **Get Memb** line of the interface by entering the addressing mode, address of the device, source endpoint number, destination endpoint number and Group ID, and then clicking the **Get Memb** button.



After sending a successful "Get Scene Membership" command, a response listing the number of scenes and the Scene IDs available will be displayed in the **Received Message View** pane.
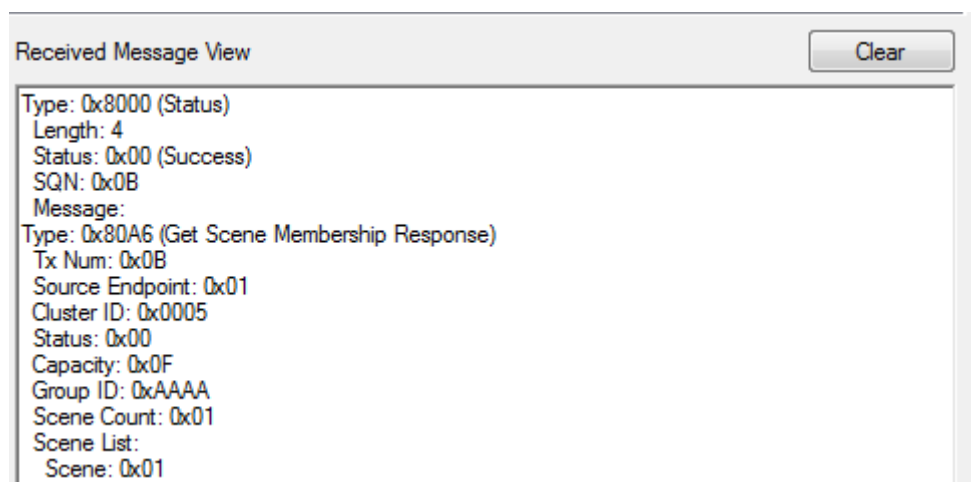


### 4.2.7.5 Remove All Scenes

You can remove all scenes associated with a particular group on a device by sending a "Remove all Scenes" command to the device. This is done in the **Remove All** line of the

interface by entering the addressing mode, address of the device, source endpoint number, destination endpoint number and Group ID, and then clicking the **Remove All** button.

| Remove All | Short Addr ▼ | 42C3 | 1 | 1 | AAAA |
|---|---|---|---|---|---|

After sending a successful "Remove All Scenes" command, a response indicating whether the removal was successful will be displayed in the **Received Message View** pane.
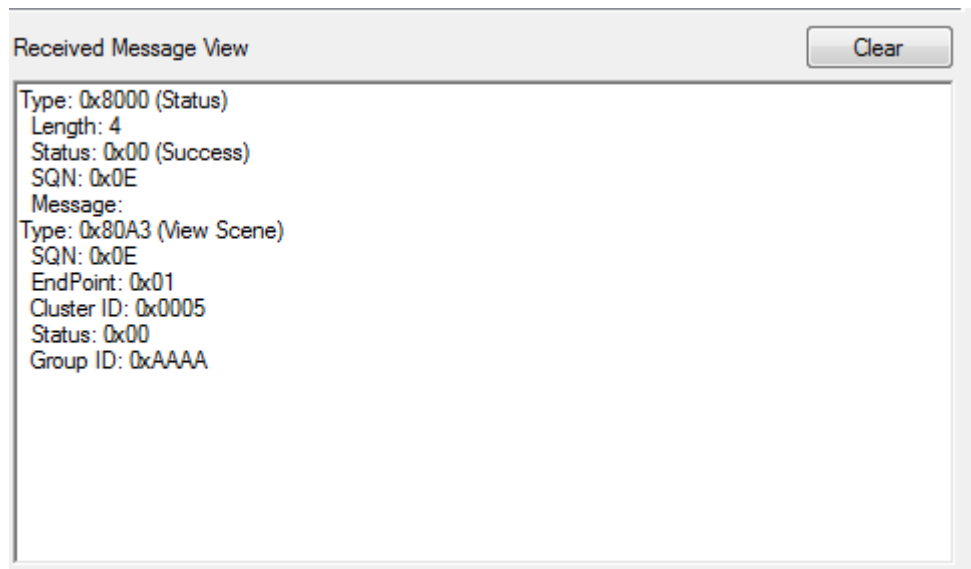


### 4.2.7.6 Remove Scene

You can remove a specific scene associated with a particular group on a device by sending a "Remove Scene" command to the device. This is done in the **Remove** line of the interface by entering the addressing mode, address of the device, source endpoint number, destination endpoint number, Group ID and Scene ID, and then clicking the **Remove** button.
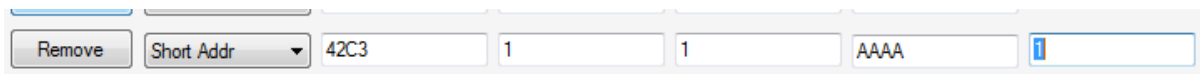


After sending a successful "Remove Scene" command, a response indicating whether the removal was successful will be displayed in the **Received Message View** pane.
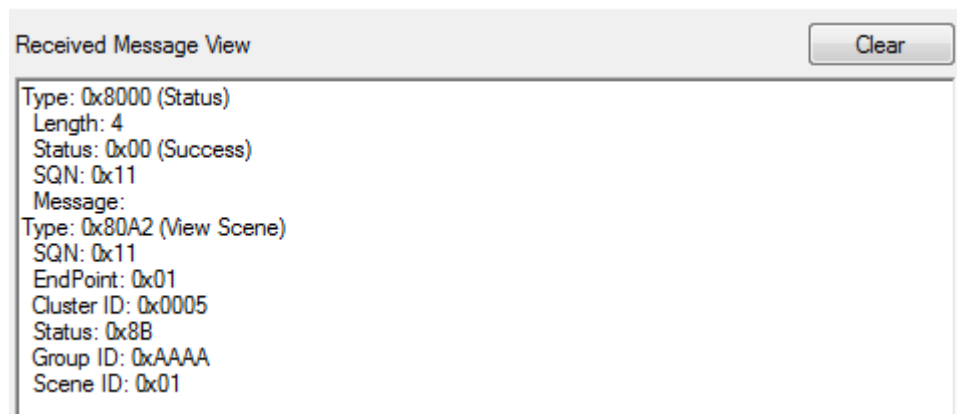
## 4.2.8 Running Over-The-Air (OTA) Upgrade

The ZGWUI provides an interface to perform an Over-The-Air (OTA) upgrade. This involves loading an application binary that will be served out 'over the air' to devices in the network. The following sections demonstrates how OTA upgrade is executed from the ZGWUI Tab in the Test Tool 12 application.

The OTA feature is available on both IDEs, IAR and MCUX, for the following projects:

**Lights**

- Dimmable_light

- Color_temperature_light

- Extended_color_light

**BDB**

- Router

- End_device

**Sensors**

- Light_sensor

- LTO_sensor

- Occupancy_sensor

The OTA upgrade process described below is the same for every project. This document will describe the process using a Dimmable Light device, for both IAR and MCUX. In this example, the following binary is used:

**dimmable_light_frdmkw41z.bin**

### 4.2.8.1 . Generating an OTA upgrade file using IAR EW

In order to upgrade the image on a client device using IAR, an \*.ota file with the necessary headers, bitmap and crc is required. The \*.ota file is generated from the binary above by taking the following steps:

The binary must be first compiled. First, some modifications are required for the OTA to be executed.

a) Right click on the project name in IAR EW -> Options -> Linker.
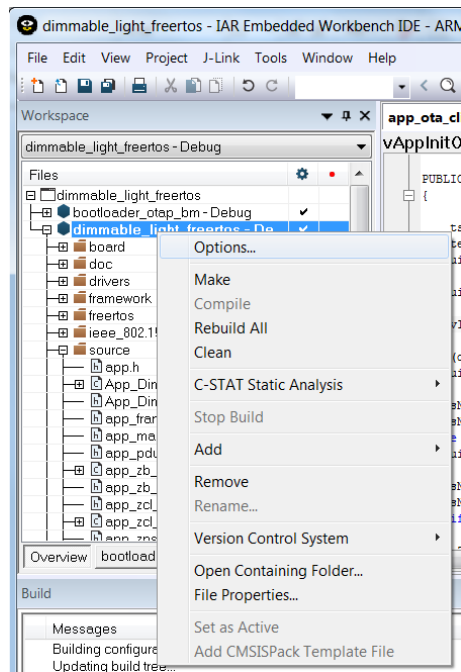
b) Set *gUseBootloaderLink_c* to 1 and *gEraseNVMLink_d* to 1



c) In the folder structure in IAR go to *source->config.h* and uncomment the following line:

> #define BUILD_OTA

> **Note:** Due to known hardware limitations, the TSI (SW2 and SW5) are disabled while OTA is enabled. SW3 and SW4 remain unchanged.

> **Note:** In order for OTA to work properly, "#define BUILD_OTA" must be uncommented in both Control Bridge and Dimmable Lights.

As can be seen, if the *BUILD_OTA* macro is added, the number of keys that can be used will be reduced. This is due to the fact that the external flash memory uses the same multiplexed pins as two of the switches.

d)  Open zcl_options.h, find and comment the following define:

   #define OTA_STRING_COMPARE



e)  Compile the project and download it on the board.

f)  Program the OTAP bootloader onto the board:

- Go to the following folder and open *bootloader_otap_bm.eww*:
  *boards\frdmkw41z\wireless_examples\framework\bootloader_otap\bm\iar*

- Compile the project and download it onto the board.

g)  In order to create an image suitable for OTA transfer:

- Open the OTA.h file, and search for the following define

   #define OTA_FILE_VERSION_DEFAULT          (uint32)0x31013102

   Change the value of this define to a different value, for example 0x31013103.
   OTA.h can be found here :
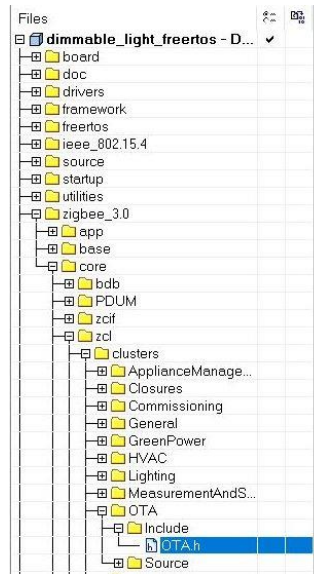
- Right click on the project name in IAR EW -> Options -> Linker (as in step 1) and set gEraseNVMLink_d to 0.
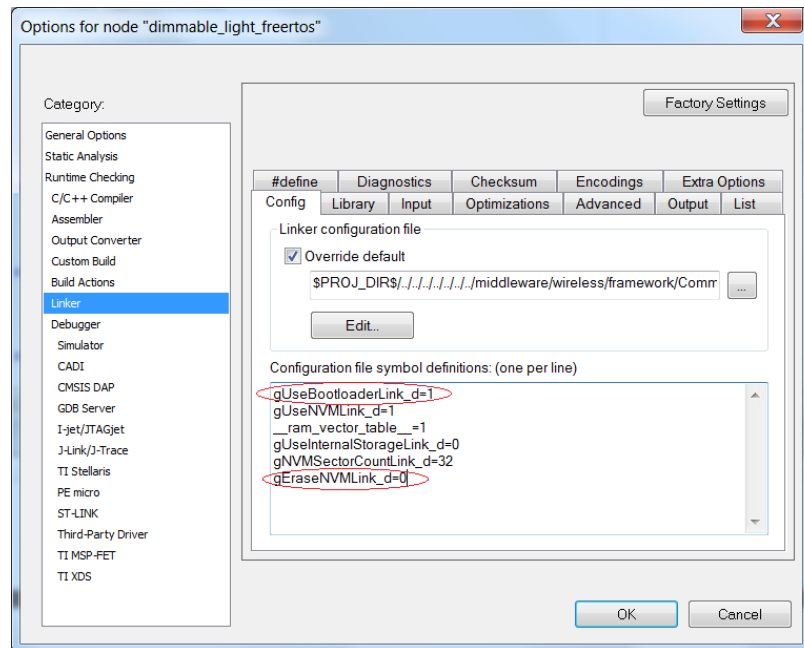


- Right click on the project name, select Make.
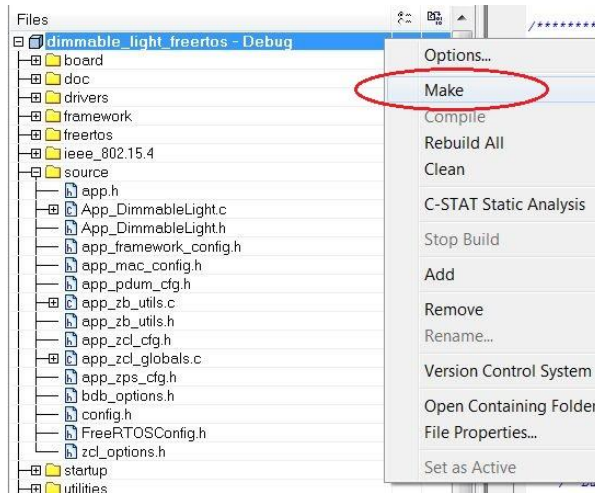
The board is now loaded with dimmable light project and with the bootloader and we have a *.bin file that is ready to be transferred. The steps necessary to obtain an OTA (*.ota) image, that can be transferred, are described in chapter **4.2.8.3. Using the ZGWUI OTA Cluster**

## 4.2.8.2 Generating an OTA upgrade file using MCUX

In order to upgrade the image on a client device using MCUX and obtain a proper *.bin file the following steps must be made :
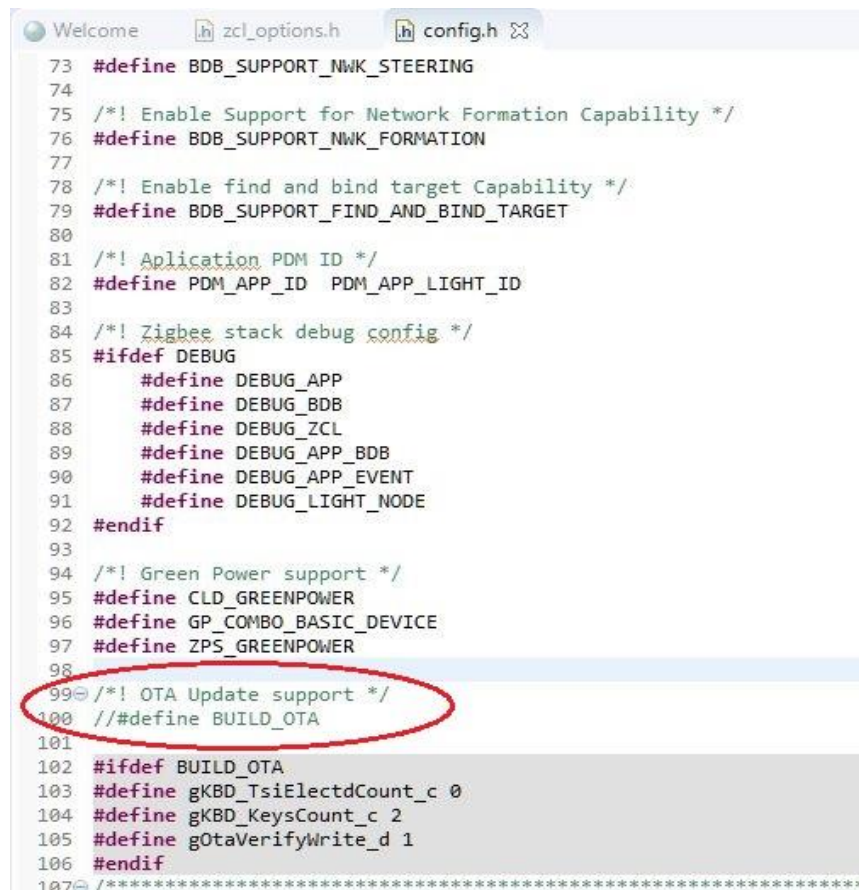
a) Open the Dimmable light project in MCUX:
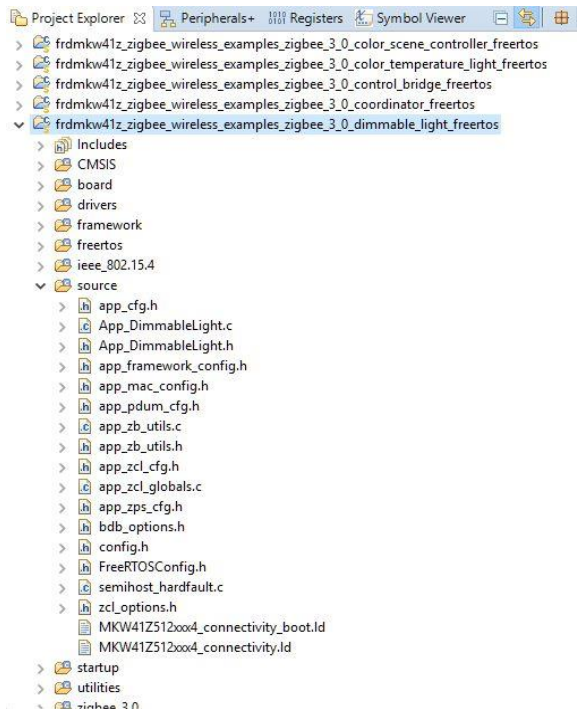


b) Open dimmable_light_freertos > source > **config.h** and uncomment the following line:

> #define BUILD_OTA

> **Note:** Due to known hardware limitations, the TSI (SW2 and SW5) are disabled while OTA is enabled. SW3 and SW4 remain unchanged.
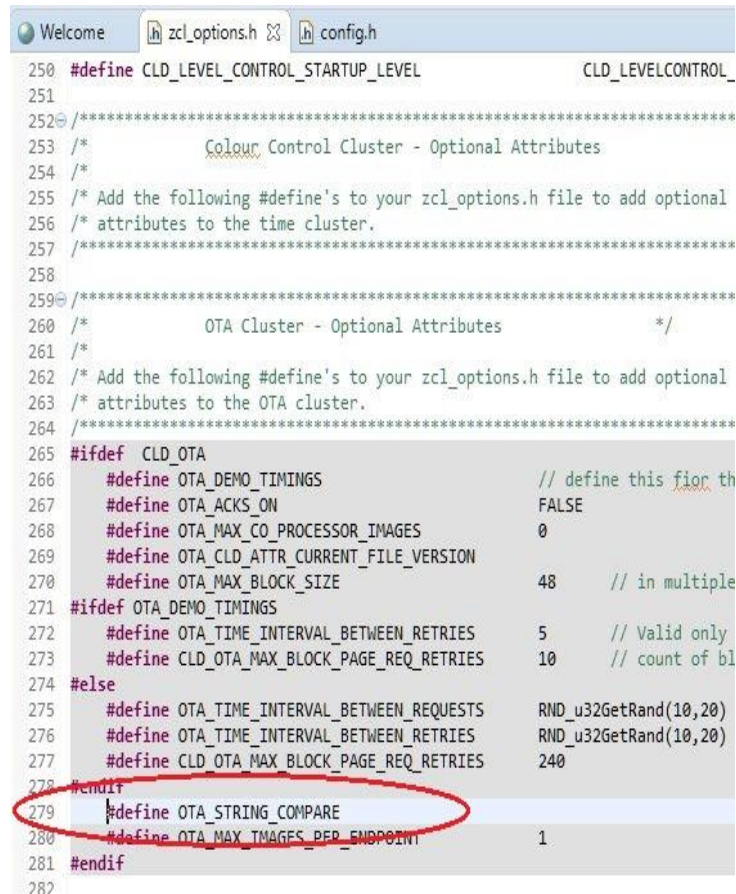
```
Welcome        h zcl_options.h      h config.h  ⋈
73  #define BDB_SUPPORT_NWK_STEERING
74
75  /*! Enable Support for Network Formation Capability */
76  #define BDB_SUPPORT_NWK_FORMATION
77
78  /*! Enable find and bind target Capability */
79  #define BDB_SUPPORT_FIND_AND_BIND_TARGET
80
81  /*! Aplication PDM ID */
82  #define PDM_APP_ID    PDM_APP_LIGHT_ID
83
84  /*! Zigbee stack debug config */
85  #ifdef DEBUG
86      #define DEBUG_APP
87      #define DEBUG_BDB
88      #define DEBUG_ZCL
89      #define DEBUG_APP_BDB
90      #define DEBUG_APP_EVENT
91      #define DEBUG_LIGHT_NODE
92  #endif
93
94  /*! Green Power support */
95  #define CLD_GREENPOWER
96  #define GP_COMBO_BASIC_DEVICE
97  #define ZPS_GREENPOWER
98
99⊖ /*! OTA Update support */
100 //#define BUILD_OTA
101
102 #ifdef BUILD_OTA
103 #define gKBD_TsiElectdCount_c 0
104 #define gKBD_KeysCount_c 2
105 #define gOtaVerifyWrite_d 1
106 #endif
107⊖ /***************************************************************
```

> **Note:** In order for OTA to work properly, "#define BUILD_OTA" must be uncommented in both Control Bridge and Dimmable Light.

   **c)** Open dimmable_light_freertos > source > **zcl_options.h** and comment the following line

> #define OTA_STRING_COMPARE

d) By default, MCUX won't produce a binary file after compiling a project. In order to obtain a *.bin file, follow the steps below:

1) Right click on Dimmable Light -> Properties -> C/C++ build -> Settings -> Build Steps.

2) Uncomment the last 2 lines of the post-build steps configuration, as shown in the picture below
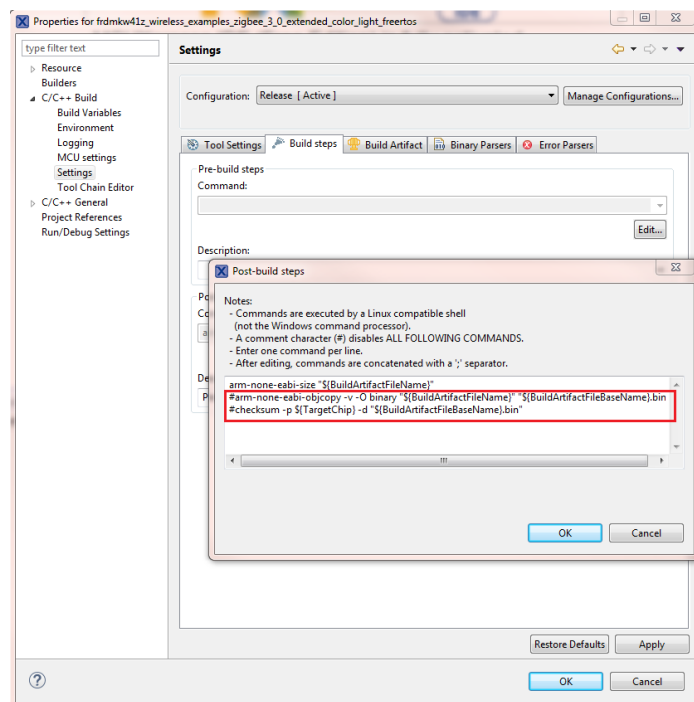
e) The next step is to change the default linker file of the project. Some changes had to be made to the default linker file in order to support the OTA feature. Follow the steps described below in order to change the linker file :

1) Right-click on the desired project and select properties (as seen in the picture below)
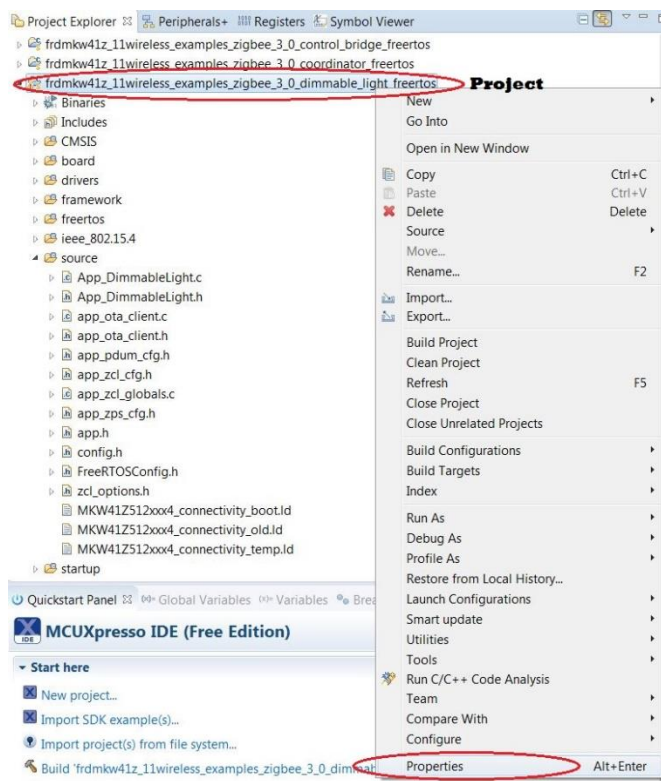


2) Select C/C++ Build -> Settings -> Tool Settings -> MCU Linker -> Managed Linker Script (as seen in the picture below)
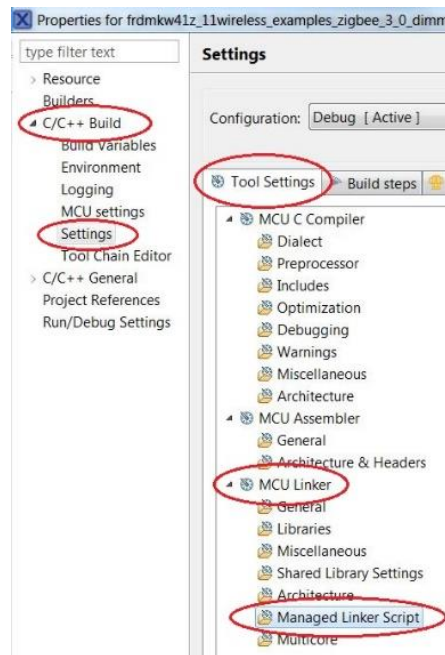
3) On the right side of the screen, rename the **Linker Script** field from **MKW41Z512xxx4_connectivity.ld** to **MKW41Z512xxx4_connectivity_boot.ld** (as seen in the picture below).
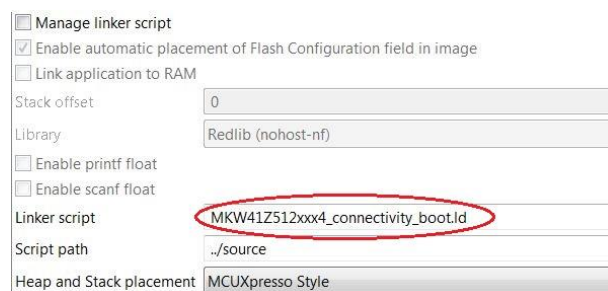


f) Open the OTA.h file and find the following define

#define OTA_FILE_VERSION_DEFAULT        (uint32)0x31013102

Change the value of this define to a different value, for example 0x31013103.
OTA.h can be found here :
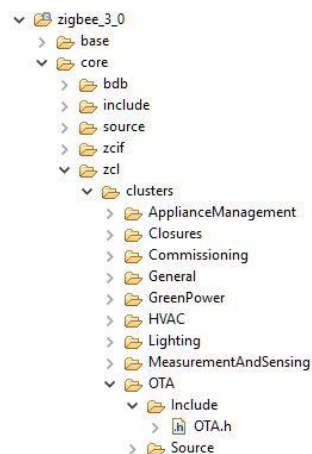


g) The last step is to program the Dimmable light project and the OTAP bootloader on to the board. (See Section 4.2.8.1 f))

The board is now loaded with the dimmable light project and with the bootloader and we have obtained a *.bin file that is ready to be transferred. The steps necessary to obtain an OTA (*.ota) image, that can be transferred, are described in chapter **4.2.8.3. Using the ZGWUI OTA Cluster**

### 4.2.8.3 Using the ZGWUI OTA Cluster

In the ZGWUI tab from TestTool 12, on the left side is a list of devices currently connected to your machine. Right click on the Control Bridge Device, select *configure device* and set the baud rate to 115200 bauds. Now the unit can function properly with OTA.
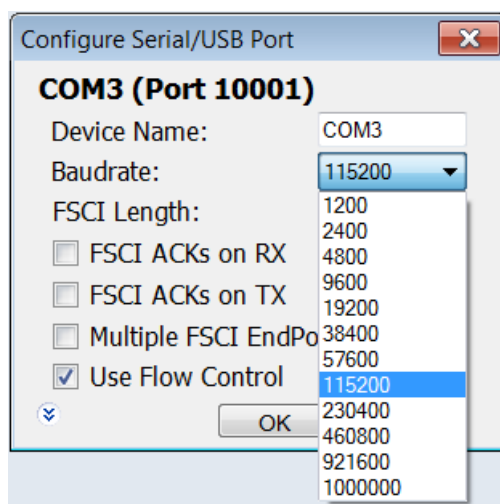


Go to the OTA cluster tab in the ZGWUI and click Load Image. When the explorer window appears, navigate to where the *.bin file obtained with IAR or MCUX is located, select it and click "Open".
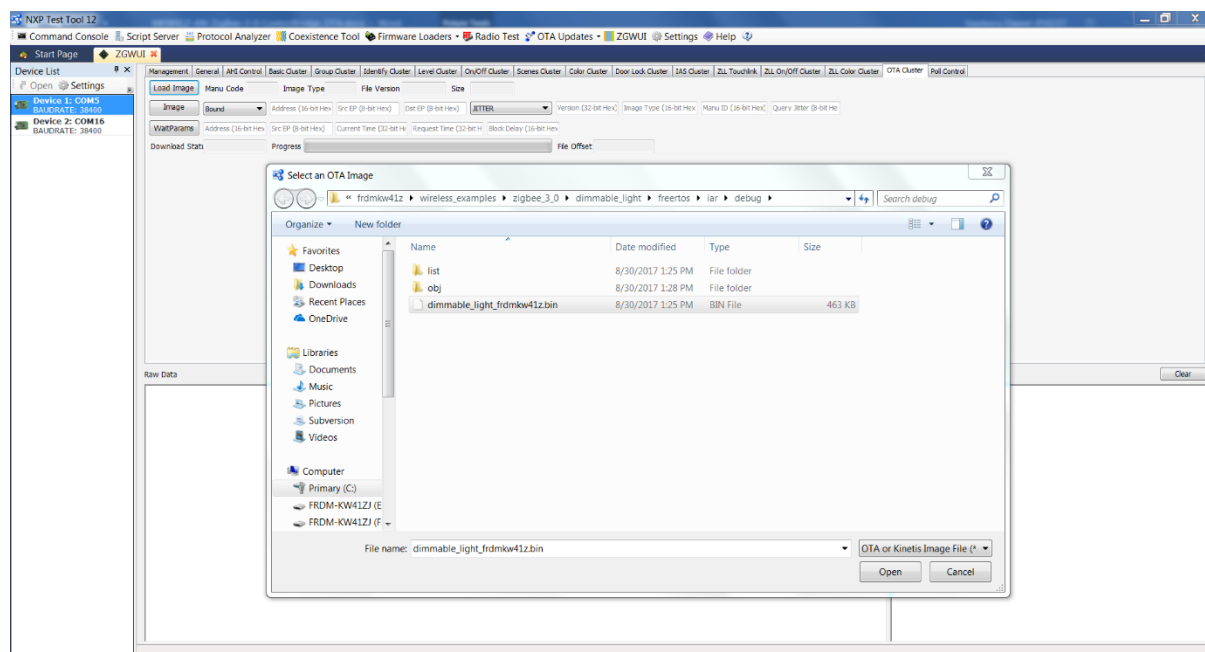
The window shown below will appear. Leave *Image contains bootloader* unselected and select *Do not include NVM firmware sectors in OTA file* in order to keep the NVM region from being overwritten. Click OK.
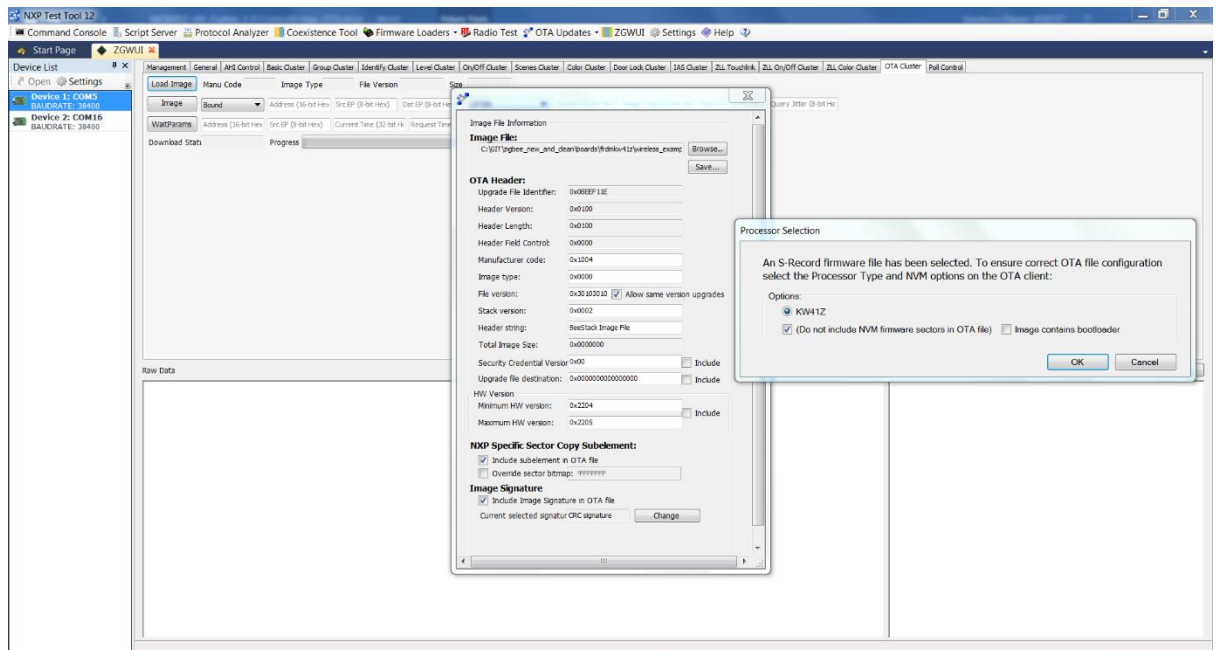


Click Save
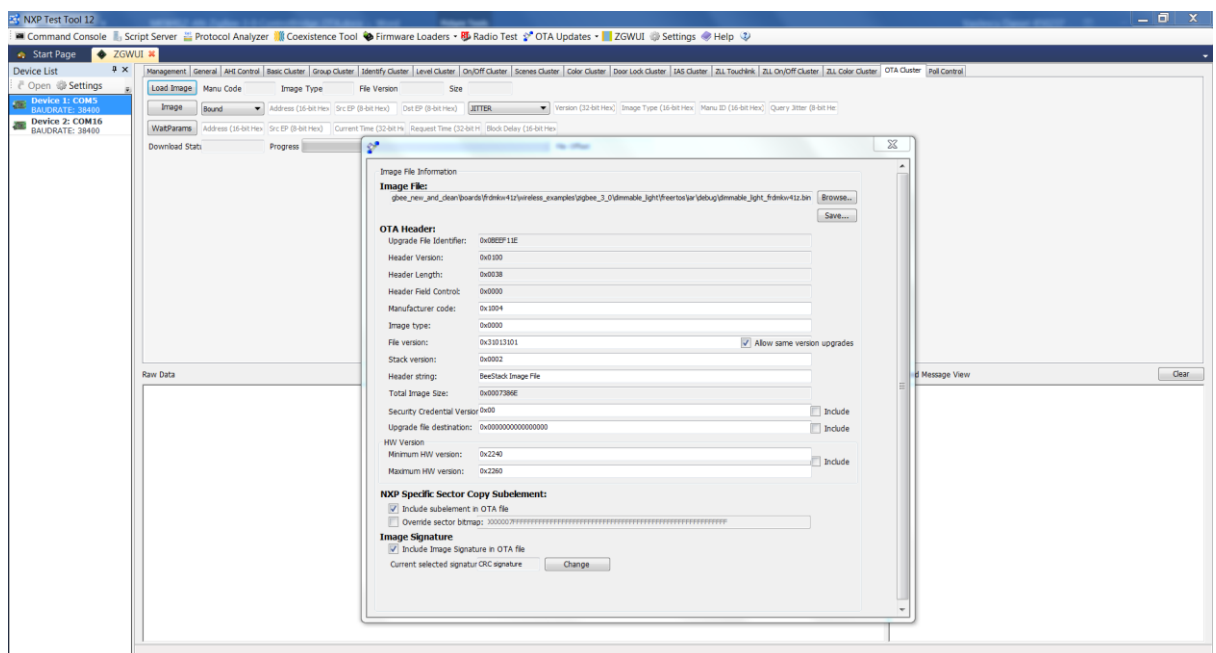


Choose the location and name of the *.ota file generated and click save
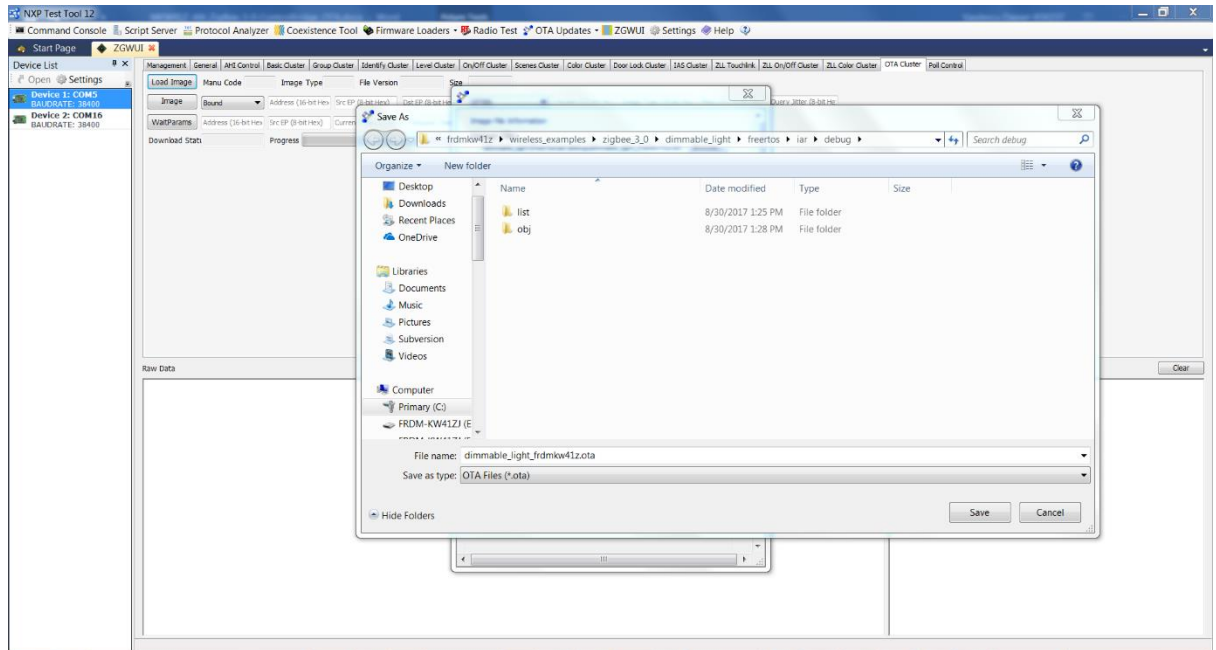
> **Note:** Any interruptions during the download process (resetting the board, power outage, user stop etc …) will cause the download to stop and the whole OTA process will start over from the beginning.

### 4.2.8.4 Modifying the file version number

The File Version number can be modified from the ota.h file, as follows:
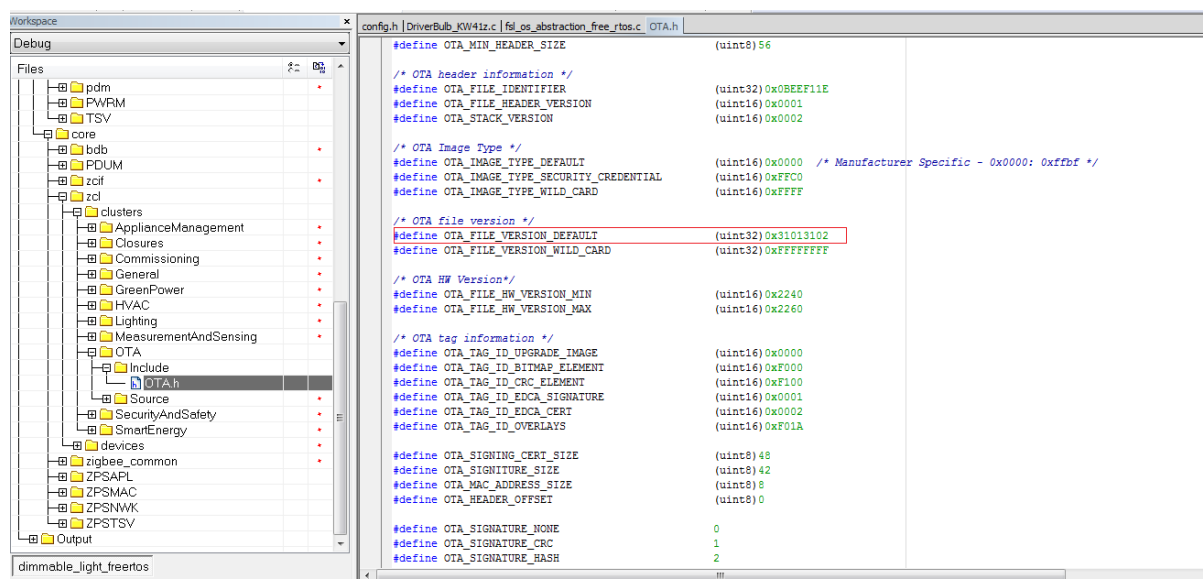
The ota.h header file is available in
<install_path>\middleware\wireless\zigbee_3_0\core\ZCL\Clusters\OTA\Include\OTA.h

**Note:** When preparing an image for an OTA upgrade, the **same** file version found in ota.h (pictured above in the red square), **must** be used in the File Version field from Test Tool (pictured below in the red square). If these file versions differ, then the OTA upgrade will enter an endless loop.

### 4.2.8.5 OTA file format

The OTA file format is composed of a header followed by a number of sub-elements. The header describes general information about the file such as version, the manufacturer that created it, and the device it is intended for. Sub-elements in the file may contain upgrade data for the embedded device, certificates, configuration data, log messages, or other manufacturer specific pieces. Below is an example file (this is the **.ota** file format used by our applications).

| Sub-element | Header | Image | Bitmap | CRC |
|---|---|---|---|---|
| Size | Variable | Variable | 38-bytes | 10-bytes |

**Table 1. OTA File Format**

### 4.2.8.5.1 OTA file format

| Octets | Data types | Field Names | Mandatory /Optional |
|---|---|---|---|
| 4 | Unsigned 32-bit integer | OTA Upgrade file identifier | M |
| 2 | Unsigned 16-bit integer | OTA Header version | M |
| 2 | Unsigned 16-bit integer | OTA Header length | M |
| 2 | Unsigned 16-bit integer | OTA Header Field control | M |
| 2 | Unsigned 16-bit integer | Manufacturer code | M |
| 2 | Unsigned 16-bit integer | Image type | M |
| 4 | Unsigned 32-bit integer | File version | M |
| 2 | Unsigned 16-bit integer | ZigBee Stack version | M |
| 32 | Character string | OTA Header string | M |
| 4 | Unsigned 32-bit integer | Total Image size (including header) | M |
| 0/1 | Unsigned 8-bit integer | Security credential version | O |
| 0/8 | IEEE Address | Upgrade file destination | O |
| 0/2 | Unsigned 16-bit integer | Minimum hardware version | O |
| 0/2 | Unsigned 16-bit integer | Maximum hardware version | O |

**Table 2. OTA Hardware File Format**

| Image | Tag ID | Length | Payload |
|---|---|---|---|
| Size | 2-bytes | 4-bytes | Variable |

**Table 3. Image Sub-element format**

| Bitmap | Tag ID | Length | Payload | Total |
|---|---|---|---|---|
| Size | 2-bytes | 4-bytes | 32-bytes | 38-bytes |

**Table 4. Bitmap sub-element format**

| CRC | Tag ID | Length | Payload | Total |
|------|---------|---------|---------|----------|
| Size | 2-bytes | 4-bytes | 4-bytes | 10-bytes |

**Table 5. CRC Sub-element format**

The ZGWUI in Test Tool always adds the Bitmap and CRC sub-elements in the order right after the image sub-element.

### 4.2.8.6 Loading the Upgrade Binary

To perform an OTA upgrade, the relevant upgrade binary file needs to be loaded into the ZGWUI application. Click on the **OTA Cluster** tab, which is displayed as follows:



Click the **Load Image** button to bring up the file explorer window. Navigate to the folder which contains the OTA upgrade binary file that will be used to upgrade the remote device and select the file – this is a **.ota** file, in this case:

**dimmable_light_frdmkw41z.ota**
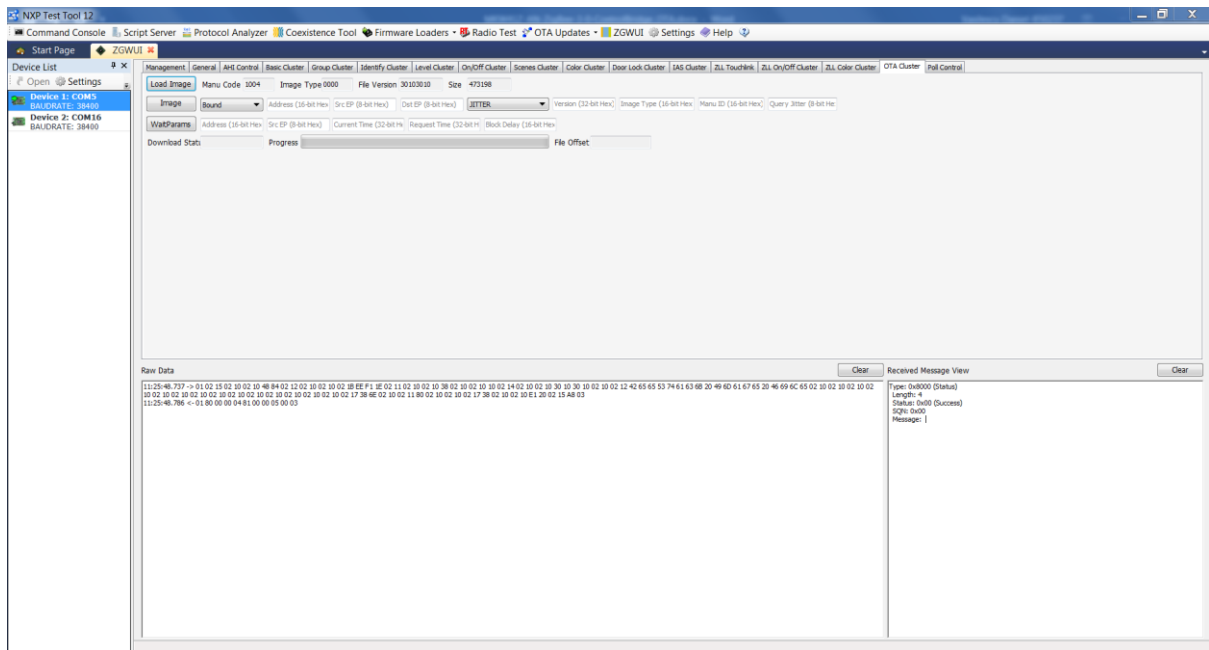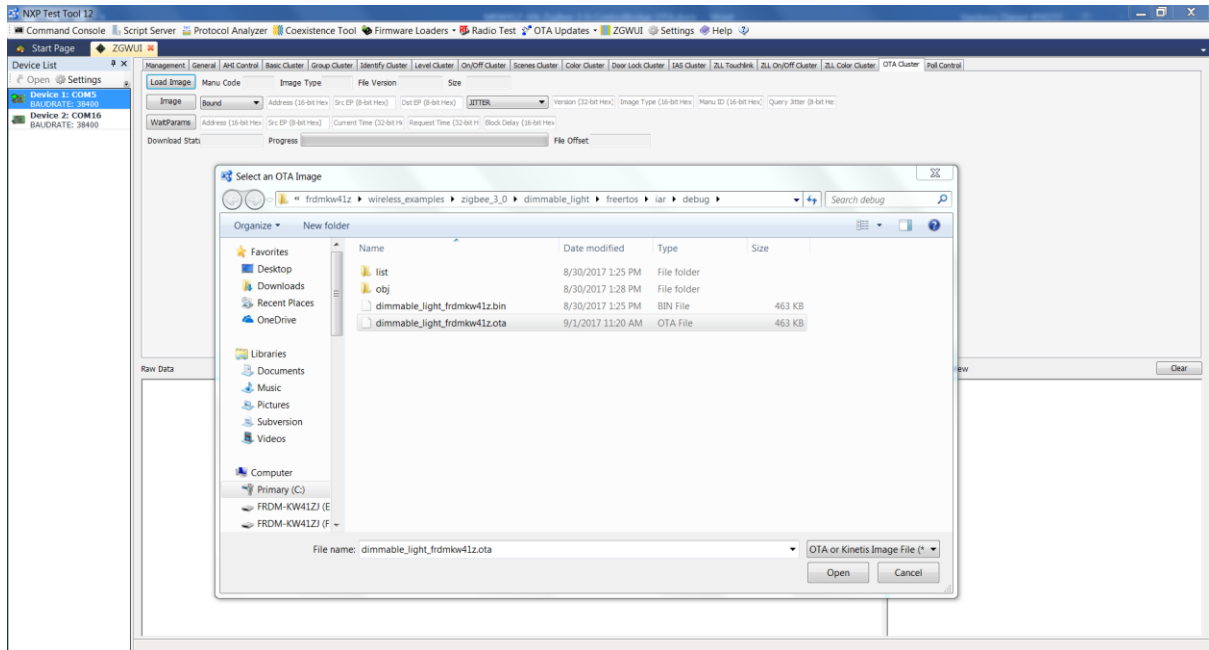
After loading the binary file, the ZGWUI will populate the Load Image textboxes with some useful data, including manufacturer code, image type, file version and binary size.
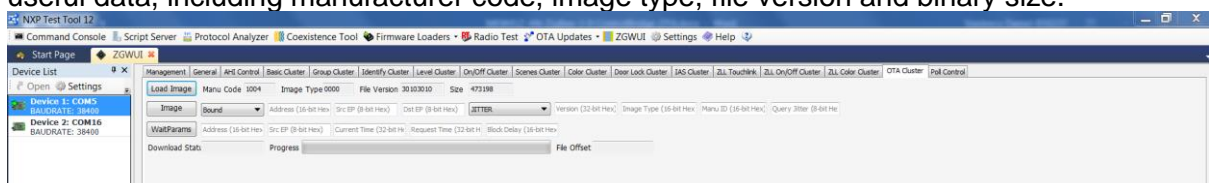


The ZGWUI also sends a serial command to the Control Bridge to inform the OTA Upgrade cluster of the loaded binary. The OTA header information is sent, which is loaded into the OTA Upgrade server. This means that when a remote device sends an image request to the server, the Control Bridge will be able to reply indicating that there is an image available.

### 4.2.8.7 Image Notify

The "Image Notify" command is used to inform all relevant devices in the network that an OTA upgrade image is available (only devices to which the image is applicable are notified). This command contains the following parameters:

- Addressing mode

- Destination address

- Source endpoint

- Destination endpoint

- Image notify payload type

- Version

- Image type

- Manufacturer ID

- Query jitter

For descriptions of the "image notify payload type" and "query jitter" parameters, please refer to the description of the `tsOTA_ImageNotifyCommand` structure in the *ZigBee Cluster Library User Guide*.

The version, image type and manufacturer ID are visible in the **Load Image** textboxes, which can be seen below along with the line for the **Image Notify** command.

| Load Image | Manu Code | 1037 | Image Type | 0101 | File Version | 00000002 | Size | 150158 | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Image Notify | Broadcast ▾ | FFFC | | 1 | | 1 | JITTER ▾ | 00000002 | 0101 | 1037 | 64 | |

The above command notifies all relevant devices in the network and instructs all of them to upgrade straight away.

### 4.2.8.8 Device Updating

When a device has determined that the OTA upgrade binary on the host is relevant to itself (regardless of whether it was informed via an Image Notify command or as the result of an update request), the device will start upgrading.

The progress bar in the ZGWUI, shown below, indicates the current status of the upgrade. The File Offset value is the number of bytes the server has sent to the device so far.

| Download Status | In Progress | Progress | | File Offset | 16560 |
|---|---|---|---|---|---|

Note that there is only one progress bar and if you have multiple devices upgrading, the bar will appear slightly random, as it will reflect whichever device is requesting a block of data. When a device has finished upgrading, the download status will change to "Complete" and the progress bar will be full.

| Download Status | Complete | Progress | | File Offset | 150144 |
|---|---|---|---|---|---|

Upon completing an OTA upgrade, an End Request is sent to the host (containing the OTA header information the device received from the OTA server) in order to indicate that the device is going to reset.

```
Received Message View                              [ Clear ]

SQN: 0x00
Message: h
Type: 0x8503 (OTA End Request)
SQN: 0x2A
Src Addr Mode: 0x02
Src Addr: 0x42C3
EndPoint: 0x01
Cluster ID: 0x0019 (General: OTA)
File Version: 0x00000002
Image Type: 0x0101
Manu Code: 0x1037
Status: 0x00
Type: 0x8000 (Status)
Length: 4
Status: 0x00 (Success)
SQN: 0x00
Message: μ
```
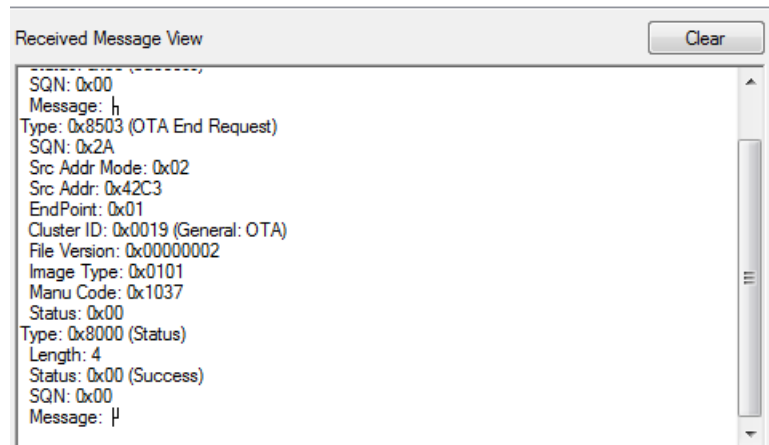
# 5 Release Details

## 5.1 Tested With

| Product Type | Part Number | Build |
|---|---|---|
| KW41Z ZigBee 3.0 SDK | FRDM-KW41Z | 6.0.6 |

## 5.2 Firmware Sizes

| Firmware | Sizes |
|---|---|
| App_Control Bridge | `267 134 bytes of readonly code memory`<br>`18 109 bytes of readonly data memory`<br>`37 880 bytes of readwrite data memory`<br>`Filename: control_bridge_frdmkw41z.bin` |

# Appendix A: Serial Protocol

## A.1.Physical Characteristics

The serial link between the ZGWUI (ZigBee Gateway User Interface integrated in the Test Tool application) and wireless microcontroller runs at 115200baud. The link settings are 8 data bits with no parity. No flow control (hardware or software) is used.

## A.2.Message Characteristics

The protocol reserves byte values less than 0x10 for use as special characters (Start and End characters, for example). So to allow data which contains these reserved values to be sent, a procedure known as "byte stuffing" is used. This consists of identifying a byte to be sent that falls into the reserved character range, sending an Escape character (0x02) first, followed by the data byte XOR'd with 0x10.

For example, if a non-special character with the value of 0x05 is to be sent:

- Send the Escape byte (0x02)
- XOR the byte to be sent with 0x10 (0x05 xor 0x10 = 0x15)
- Send the modified byte

The messages consist of the following:

- Start character (special character)
- Message type (byte stuffed)
- Message length (byte stuffed)
- Checksum (byte stuffed)
- Message data (byte stuffed)
- End character (special character)

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | | | n+6 | n+7 | n+8 |
|------|------|------|------|------|--------|------|------|--|--|------|------|------|
| 0x01 | | | n | | | | | | | | | 0x03 |
| Start | Msg Type | | Length | | Chksum | Data | | | | | | Stop |

**Figure 3: Layout of message before byte stuffing**

### A.2.1.Start Character

The Start character is a single-byte special character with the value 0x01 and is sent as the first byte of any message to allow the receiving end to synchronise. Since this is considered a special character, it will be sent without modification.

### A.2.2.Message Type

The message type is a 16-bit value identifying the nature of the data contained in the message payload. Values implemented are defined in the message table.

### A.2.3. Message Length

The message length is a 16-bit value equal to the number of bytes in the payload section of the message, sent most significant byte first.

### A.2.4. Checksum

The checksum is an 8 bit value calculated by XORing the following (starting with a checksum of 0x00):

- Message type most-significant-byte
- Message type least-significant-byte
- Message length most-significant-byte
- Message length least-significant-byte
- Data bytes

The checksum is calculated before byte stuffing the message.

### A.2.5. Message Data

The message data is a number of bytes equal to the value sent as the message length field. The number of bytes transmitted via the UART may be higher due to presence of escape bytes sent to identify values that fall in the reserved range. All multi-byte binary data is sent in network byte order (big-endian).

### A.2.6. End Character

The end character is a single byte special character with the value 0x03 and is sent as the last byte of any message to allow the receiving end to synchronise. Since this is considered a special character, it will be sent without modification.

### A.2.7. Sequence

All commands generate a synchronous response code followed by any asynchronous responses as they become available. There is no sequence number associated with each command/response – the user must ensure that commands are issued sequentially.

Expected command response sequence:

| Direction | Message |
|---|---|
| Host -> Node | Command e.g. Get Version |
| Node -> Host | Status e.g. OK or Error, Not implemented |
| Node -> Host | Optional data messages as requested by command, e.g. Version List |

## A.3.Data Types

The following data types are used in messages between the host and slave devices. All message definitions use 32-bit integer types, unless otherwise specified.

| Name | Type |
|---|---|
| uint8_t | Unsigned 8 bit integer (one byte) |
| uint16_t | Unsigned 16 bit integer (two bytes) |
| uint32_t | Unsigned 32 bit integer (four bytes) |
| uint64_t | Unsigned 64 bit integer (eight bytes) |
| uint128_t | Unsigned 128 bit integer (sixteen bytes) |
| string | Buffer of characters (Variable Length, NULL Terminated) |
| data | Buffer of bytes (Variable length, calculated using message length) |

## A.4.Response Codes

The node acknowledges each command with an "ACK" message. The message is defined in the message table.

# Appendix B: Serial Command Set

## B.1.Common Commands

In the following tables, the term Node refers to the Control Bridge

### B.1.1.ZigBee Stack and Node Management Commands

| Message Direction | Message Description | Message Format | Expected Response |
|---|---|---|---|
| Node->Host | Status<br>Msg Type = 0x8000 | \<status:uint8_t><br>\<sequence number: uint8_t><br>\<Packet Type: uint16_t><br>\<Optional additional error information: string><br><br>Status:<br>     0 = Success<br>     1 = Incorrect parameters<br>     2 = Unhandled command<br>     3 = Command failed<br>     4 = Busy (Node is carrying out a lengthy operation and is currently unable to handle the incoming command)<br>     5 = Stack already started (no new configuration accepted)<br>     128 – 244 = Failed (ZigBee event codes)<br><br>Packet Type: The value of the initiating command request. | All status messages will have a sequence number sent back. Default of 0 for messages which are not transmitted over the air. |
| Node->Host | Log message<br>Msg Type = 0x8001 | \<log level: uint8_t><br>\<log message : string><br>Log Level :<br>Use the Linux / Unix log levels<br>     0 = Emergency<br>     1 = Alert<br>     2 = Critical<br>     3 = Error<br>     4 = Warning<br>     5 = Notice<br>     6 = Information<br>     7 = Debug | |
| Node->Host | Data Indication<br>Msg Type = 0x8002 | \<status: uint8_t><br>\<Profile ID: uint16_t><br>\<cluster ID: uint16_t><br>\<source endpoint: uint8_t><br>\<destination endpoint: uint8_t><br>\<source address mode: uint8_t><br>\<source address: uint16_t or uint64_t><br>\<destination address mode: uint8_t><br>\<destination address: uint16_t or uint64_t><br>\<payload size : uint8_t><br>\<payload : data each element is uint8_t> | |
| Node->Host | Node Cluster List – Sent by gateway node after reset<br>Msg Type = 0x8003 | \<source endpoint: uint8_t t><br>\<profile ID: uint16_t><br>\<cluster list: data each entry is uint16_t> | |

| | | | |
|---|---|---|---|
| Node->Host | Node Cluster Attribute List – Sent by Gateway node after reset Msg Type = 0x8004 | <source endpoint: uint8_t> <profile ID: uint16_t> <cluster ID: uint16_t> <attribute list: data each entry is uint16_t> | |
| Node->Host | Node Command ID List – sent by Gateway node after reset Msg Type = 0x8005 | <source endpoint: uint8_t> <profile ID: uint16_t> <cluster ID: uint16_t> <command ID list:data each entry is uint8_t> | |
| Host->Node | Get Version Msg Type = 0x0010 | No payload | Status Version List |
| Node->Host | Version List Msg Type = 0x8010 | <Major version number: uint16_t> <Installer version number: uint16_t> | |
| Host->Node | Set Extended PANID Msg Type = 0x0020 | <64-bit Extended PAN ID:uint64_t> | Status |
| Host->Node | Set Channel Mask Msg Type = 0x0021 | <channel mask:uint32_t> | Status |
| Host->Node | Set Security State & Key Msg Type = 0x0022 | <key type: uint8_t> <key: data> | Status |
| Host->Node | Set Device Type Msg Type = 0x0023 | <device type: uint8_t> Device Types:      0 = Coordinator      1 = Router      2 = Legacy Router | Status |
| Host->Node | Start Network scan Msg Type = 0x0025 | No payload | Status Network Joined / Formed |
| Host->Node | Start Network Message Type = 0x0024 | No payload | Status Network Joined / Formed |
| Node->Host | Network Joined / Formed Msg Type = 0x8024 | <status: uint8_t> <short address: uint16_t> <extended address:uint64_t> <channel: uint8_t> Status:     0 = Joined existing network     1 = Formed new network     128 – 244 = Failed (ZigBee event codes) | |
| Host->Node | ZLO/ZLL "Factory New" Reset Msg Type=0x0013 | No payload  Resets ("Factory New") the Control Bridge but persists the frame counters. | Status, followed by chip reset |
| Host->Node | "Permit join" status on the target Msg Type = 0x0014 | No payload | Status, followed by "Permit join" status response |
| Node->Host | "Permit join" status response Msg Type=0x8014 | <Status: bool_t>     0 - Off     1 - On | |
| Host->Node | Reset Msg Type = 0x0011 | No payload | Status, followed by chip reset |
| Node->Host | Non "Factory new" Restart Msg Type=0x8006 | Status –     0 - STARTUP     2 - NFN_START     6 - RUNNING  The node is provisioned from previous restart. | |

| Node->Host | "Factory New" Restart Msg Type=0x8007 | Status –<br><br>    0 - STARTUP<br>    2 - NFN_START<br>    6 - RUNNING<br><br>The node is not yet provisioned. | |
|---|---|---|---|
| Host->Node | Erase Persistent Data Msg Type = 0x0012 | No payload | Status |
| Host->Node | Bind Msg Type = 0x0030 | <target extended address: uint64_t><br><target endpoint: uint8_t><br><cluster ID: uint16_t><br><destination address mode: uint8_t><br><destination address:uint16_t or uint64_t><br><destination endpoint (value ignored for group address): uint8_t> | Status Bind response |
| Node->Host | Bind response Msg Type = 0x8030 | <Sequence number: uint8_t><br><status: uint8_t> | |
| Host->Node | Unbind Msg Type = 0x0031 | <target extended address: uint64_t><br><target endpoint: uint8_t><br><cluster ID: uint16_t><br><destination address mode: uint8_t><br><destination address: uint16_t or uint64_t><br><destination endpoint(value ignored for group address): uint8_t> | Status Unbind response |
| Node->Host | Unbind  response Msg Type = 0x8031 | <Sequence number: uint8_t><br><status: uint8_t> | |
| Node->Host | Device Announce Msg Type = 0x004D | < short address: uint16_t><br>< IEEE address: uint64_t><br>< MAC capability: uint8_t><br>MAC capability<br>    Bit 0 - Alternate PAN Coordinator<br>    Bit 1 - Device Type<br>    Bit 2 - Power source<br>    Bit 3 - Receiver On when Idle<br>    Bit 4,5 - Reserved<br>    Bit 6 - Security capability<br>    Bit 7 - Allocate Address | |
| Host->Node | Network Address request Msg Type = 0x0040 | <target short address: uint16_t><br><extended address:uint64_t><br><request type: uint8_t><br><start index: uint8_t><br>Request Type:<br>    0 = Single Request<br>    1 = Extended Request | Status Network Address response |
| Node->Host | Network Address response Msg Type = 0x8040 | <Sequence number: uin8_t><br><status: uint8_t><br><IEEE address: uint64_t><br><short address: uint16_t><br><number of associated devices: uint8_t><br><start index: uint8_t><br><device list – data each entry is uint16_t> | |
| Host->Node | IEEE Address request Msg Type = 0x0041 | <target short address: uint16_t><br><short address: uint16_t><br><request type: uint8_t><br><start index: uint8_t><br>Request Type:<br>    0 = Single<br>    1 = Extended | Status IEEE Address response |

| | | | |
|---|---|---|---|
| Node->Host | IEEE Address response<br>Msg Type = 0x8041 | <Sequence number: uin8_t><br><status: uint8_t><br><IEEE address: uint64_t><br><short address: uint16_t><br><number of associated devices: uint8_t><br><start index: uint8_t><br><device list – data each entry is uint16_t> | |
| Host->Node | Node Descriptor request<br>Msg Type = 0x0042 | <target short address: uint16_t> | Status<br>Node Descriptor response |
| Node->Host | Node Descriptor response<br>Msg Type = 0x8042 | <Sequence number: uint8_t><br><Status uint8_t><br><network address: uint16_t><br><manufacturer code: uint16_t><br><max Rx Size: uint16_t><br><max Tx Size: uint16_t><br><server mask: uint16_t><br><descriptor capability: uint8_t><br><mac flags: uint8_t><br><max buffer size: uint8_t ><br><bit fields: uint16_t><br><br>Bitfields:<br>Logical type (bits 0-2<br>        0 - Coordinator<br>        1 - Router<br>        2 - End Device)<br>Complex descriptor available (bit 3)<br>User descriptor available (bit 4)<br>Reserved (bit 5-7)<br>APS flags (bit 8-10 – currently 0)<br>Frequency band(11-15 set to 3 (2.4Ghz))<br><br>Server mask bits:<br>    0  - Primary trust center<br>    1 - Back up trust center<br>    2 - Primary binding cache<br>    3 - Backup binding cache<br>    4 - Primary discovery cache<br>    5 - Backup discovery cache<br>    6  - Network manager<br>    7 to15 - Reserved<br><br>MAC capability<br>Bit 0 - Alternate PAN Coordinator<br>Bit 1 - Device Type<br>Bit 2 - Power source<br>Bit 3 - Receiver On when Idle<br>Bit 4-5 - Reserved<br>Bit 6 - Security capability<br>Bit 7 - Allocate Address<br><br>Descriptor capability:<br>    0 - extended Active endpoint  list available<br>    1 - Extended simple descriptor list available<br>    2 to 7 - Reserved | |
| Host->Node | Simple Descriptor request<br>Msg Type = 0x0043 | <target short address: uint16_t><br><endpoint: uint8_t> | Status<br>Simple Descriptor response |

| | | | |
|---|---|---|---|
| Node->Host | Simple Descriptor response Msg Type= 0x8043 | <Sequence number: uint8_t> <status: uint8_t> <nwkAddress: uint16_t> <length: uint8_t> <endpoint: uint8_t> <profile: uint16_t> <device id: uint16_t> <bit fields: uint8_t > <InClusterCount: uint8_t > <In cluster list: data each entry is uint16_t> <OutClusterCount: uint8_t> <Out cluster list: data each entry is uint16_t> Bit fields: Device version: 4 bits (bits 0-4) Reserved: 4 bits (bits4-7) | |
| Host->Node | Power Descriptor request Msg Type = 0x0044 | <target short address: uint16_t> | Status Power Descriptor response |
| Node->Host | Power Descriptor response Msg Type= 0x8044 | <Sequence number: uin8_t> <status : uint8_t> <bit field : uint16_t>  Bit fields      0 to 3: current power mode      4 to 7: available power source      8 to 11: current power source      12 to15: current power source level | |
| Host->Node | Active Endpoint request Msg Type = 0x0045 | <target short address: uint16_t> | Status Active Endpoint response |
| Node->Host | Active Endpoint response Msg Type = 0x8045 | <Sequence number: uint8_t> <status: uint8_t> <Address: uint16_t> <endpoint count: uint8_t> <active endpoint list: each data element of the type uint8_t > | |
| Host->Node | Match Descriptor request Msg Type = 0x0046 | <target short address: uint16_t> <profile id: uint16_t> <number of input clusters: uint8_t> <input cluster list:data: each entry is uint16_t > <number of output clusters:  uint8_t> <output cluster list:data: each entry is uint16_t> | Status Match Descriptor response |
| Node->Host | Match Descriptor response Msg Type = 0x8046 | <Sequence number: uint8_t> <status: uint8_t> <network address: uint16_t> <length of list: uint8_t> <match list: data each entry is uint8_t> | |
| Host->Node | Remove Device Msg Type = 0x0026 | <target short address: uint64_t> <extended address: uint64_t> | Status Leave indication |
| Host->Node | User Descriptor Set Msg Type = 0x002B | < target short address: uint16_t> < Address of interest:  uint16_t> < string length:          uint8_t> <data:                uint8_t stream > | Status User descriptor notify response |
| Host->Node | User Descriptor Request Msg Type = 0x002C | < target short address: uint16_t> < Address of interest:  uint16_t> | Status User Descriptor response |
| Node->Host | User Descriptor Response Msg Type = 0x802C | <Sequence number: uin8_t> <status: uint8_t> <network address of interest: uint16_6> <length:                      uint8_t> <data:                        uint8_t stream> | |

| | | | |
|---|---|---|---|
| Node->Host | User Descriptor Notify Msg Type = 0x802B | <Sequence number: uin8_t> <status: uint8_t> <Network address of interest: uint16_t> | |
| Host->Node | Complex Descriptor request Msg Type = 0x0034 | < target short address: uint16_t> < Address of interest:  uint16_t> | Status Complex Descriptor Response |
| Node->Host | Complex Descriptor response | <Sequence number: uin8_t> <status: uint8_t> <Network address of interest: uint16_t> <Length:        uint8_t> <xml tag:        uint8_t> <field count:    uint8_t> <field values:  uint8_t stream> | |
| Host->Node | Management Leave request Msg Type = 0x0047 | <target short address: uint16_t> <extended address: uint64_t> <Rejoin: uint8_t> <Remove Children: uint8_t>  Rejoin,              0 = Do not rejoin              1 = Rejoin  Remove Children              0 = Leave, removing children              1 = Leave, do not remove children | Status Management Leave response Leave indication |
| Node->Host | Management Leave response Msg Type = 0x8047 | <Sequence number: uin8_t> <status: uint8_t> | |
| Node->Host | Leave indication Msg Type = 0x8048 | <extended address: uint64_t> <rejoin status: uint8_t> | |
| Host->Node | Permit Joining request Msg Type = 0x0049 | <target short address: uint16_t> <interval: uint8_t> <TCsignificance: uint8_t>  Target address: May be address of gateway node or broadcast (0xfffc) Interval:              0 = Disable Joining              1 – 254 = Time in seconds to allow joins              255 = Allow all joins TCsignificance:              0 = No change in authentication              1 = Authentication policy as spec | Status |
| Host->Node | Management Network Update request Msg Type = 0x004A | <target short address: uint16_t> <channel mask: uint32_t> <scan duration: uint8_t> <scan count: uint8_t> <network update ID: uint8_t> <network manager short address: uint16_t>  Channel Mask:              Mask of channels to scan Scan Duration:              0 – 0xFF Multiple of superframe duration. Scan count:              Scan repeats 0 – 5 Network Update ID:              0 – 0xFF Transaction ID for scan | Status Management Network Update response |

| Node->Host | Management Network Update response<br>Msg Type = 0x804A | <Sequence number: uint8_t><br><status: uint8_t><br><total transmission: uint16_t><br><transmission failures: uint16_t><br><scanned channels: uint32_t ><br><scanned channel list count: uint8_t><br><channel list: list each element is uint8_t> | |
|---|---|---|---|
| Host->Node | System Server Discovery request<br>Msg Type = 0x004B | <target short address: uint16_t><br><Server mask: uint16_t><br>Bitmask according to spec. | Status<br>System Server Discovery response |
| Node->Host | System Server Discovery response<br>Msg Type = 0x804B | <Sequence number: uint8_t><br><status: uint8_t><br><Server mask: uint16_t><br>Bitmask according to spec. | |
| Host->Node | Management LQI request<br>Msg Type = 0x004E | <Target Address : uint16_t><br><Start Index : uint8_t> | Status<br>Management LQI response |

## B.1.2. Entire Profile

| Message Direction | Message Description | Message Format | Expected Response |
|---|---|---|---|
| Node->Host | Management LQI response Msg Type=0x804E | <Sequence number: uint8_t> <status: uint8_t> <Neighbour Table Entries : uint8_t> <Neighbour Table List Count : uint8_t> <Start Index : uint8_t> <List of Entries elements described below :> Note: If Neighbour Table list count is 0, there are no elements in the list. NWK Address : uint16_t Extended PAN ID : uint64_t IEEE Address : uint64_t Depth : uint_t Link Quality : uint8_t Bit map of attributes Described below: uint8_t<br><br>bit 0-1 Device Type (0-Coordinator 1-Router 2-End Device)<br><br>bit 2-3 Permit Join status (1- On 0-Off)<br><br>bit 4-5 Relationship (0-Parent 1-Child 2-Sibling)<br><br>bit 6-7 Rx On When Idle status (1-On 0-Off) | |
| Host->Node | Read Attribute request Msg Type = 0x0100 | <address mode: uint8_t> <target short address: uint16_t> <source endpoint: uint8_t> <destination endpoint: uint8_t> <Cluster id: uint16_t> <direction: uint8_t> <manufacturer specific: uint8_t> <manufacturer id: uint16_t> <number of attributes: uint8_t> <attributes list: data list of uint16_t each><br><br>Direction:    0 - from server to client    1 - from client to server Manufacturer specific :    0 – No    1 – Yes | Status Read Attribute response |
| Host->Node | Write Attribute request Msg Type = 0x0110 | <address mode: uint8_t> <target short address: uint16_t> <source endpoint: uint8_t> <destination endpoint: uint8_t> <Cluster id: uint16_t> <direction: uint8_t> <manufacturer specific: uint8_t> <manufacturer id: uint16_t> <number of attributes: uint8_t> <attributes list: data list of uint16_t each><br><br>Direction:    0 - from server to client    1 - from client to server Manufacturer specific :    1 – Yes    0 – No | Data Indication Msg Type = 0x8002 |

| | | | |
|---|---|---|---|
| Host->Node | Attribute Discovery request<br>Msg Type = 0x0140 | \<address mode: uint8_t><br>\<target short address: uint16_t><br>\<source endpoint: uint8_t><br>\<destination endpoint: uint8_t><br>\<Cluster id: uint16_t><br>\<Attribute id : uint16_t><br>\<direction: uint8_t><br>\<manufacturer specific: uint8_t><br>\<manufacturer id: uint16_t><br>\<Max number  of identifiers: uint8_t><br><br>Direction:<br>    0 - from server to client<br>    1 - from client to server<br>Manufacturer specific :<br>    1 – Yes<br>    0 – No | Status<br>Attribute<br>Discovery<br>response |
| Node->Host | Attribute Discovery response<br>Msg Type = 0x8140 | \<complete: uint8_t><br>\<attribute type: uint8_t><br>\<attribute id: uint16_t><br><br>Complete:<br>    0 – more attributes to follow<br>    1 – this was the last attribute | |
| Host->Node | Enable Permissions Controlled Joins<br>Msg Type = 0x0027 | \<Enable/Disable : uint8_t><br>1 – Enable<br>2 – Disable | Status |
| Host->Node | Authenticate Device<br>Msg Type = 0x0028 | \<IEEE address ; uint64_t><br>\<Key : 16 elements byte each> | Status<br>Authenticate<br>response |
| Node->Host | Authenticate response<br>Msg Type = 0x8028 | \<IEEE address of the Gateway: uint64_t><br>\<Encrypted Key : uint8_t  16 elements><br>\<MIC : uint8           4 elements><br>\<IEEE address of the initiating node : uint64_t><br>\<Active Key Sequence number : uint8_t><br>\<Channel : uint8_t><br>\<Short PAN Id : uint16_t><br>\<Extended PAN ID : uint64_t> | |
| Host->Node | Configure Reporting request<br>Msg Type = 0x0120 | \<address mode: uint8_t><br>\<target short address: uint16_t><br>\<source endpoint: uint8_t><br>\<destination endpoint: uint8_t><br>\<Cluster id: uint16_t><br>\<direction: uint8_t><br>\<manufacturer specific: uint8_t><br>\<manufacturer id: uint16_t><br>\<number of attributes: uint8_t><br>\<attributes list: data list of uint16_t  each><br>Attribute direction : uint8_t<br>Attribute type : uint8_t<br>Attribute id : uint16_t<br>Min interval : uint16_t<br>Max interval : uint16_t<br>Timeout : uint16_t<br>Change : uint8_t | Status<br>Configure<br>Reporting<br>response |
| Node->Host | Configure Reporting response<br>Msg Type = 0x8120 | \<Sequence number: uint8_t><br>\<Src address : uint16_t><br>\<Endpoint: uint8_t><br>\<Cluster id: uint16_t><br>\<Status: uint8_t> | |

| | | | |
|---|---|---|---|
| Node->Host | Read individual Attribute Response Msg Type = 0x8100 | <Sequence number: uint8_t> <Src address : uint16_t> <Endpoint: uint8_t> <Cluster id: uint16_t> <Attribute Enum: uint16_t> <Attribute status: uint8_t> <Attribute data type: uint8_t> <Size Of the attributes in bytes: uint16_t> <Data byte list : stream of uint8_t> | |
| Node->Host | Write Attribute Response Msg Type = 0x8110 | <Sequence number: uint8_t> <Src address : uint16_t> <Endpoint: uint8_t> <Cluster id: uint16_t> <Attribute Enum: uint16_t> <Attribute status: uint8_t> <Attribute data type: uint8_t> <Size Of the attributes in bytes: uint16_t> <Data byte list : stream of uint8_t> | |
| Node->Host | Report Individual Attribute response Msg Type = 0x8102 | <Sequence number: uint8_t> <Src address : uint16_t> <Endpoint: uint8_t> <Cluster id: uint16_t> <Attribute Enum: uint16_t> <Attribute status: uint8_t> <Attribute data type: uint8_t> <Size Of the attributes in bytes: uint16_t> <Data byte list : stream of uint8_t> | |
| Node->Host | Default response Msg Type = 0x8101 | <Sequence number: uint8_t> <endpoint: uint8_t> <Cluster id: uint16_t> <Command Id: uint8_t> <Status code: uint8_t> | |

## B.1.3. Group Cluster Commands

| Message Direction | Message Description | Message Format | Expected Response |
|---|---|---|---|
| Host->Node | Add Group Msg Type = 0x0060 Command ID = 0x00 | <address mode: uint8_t> <target short address: uint16_t> <source endpoint: uint8_t> <destination endpoint: uint8_t> <group address: uint16_t> | Status Add Group response |
| Node->Host | Add Group response Msg Type = 0x8060 Command ID = 0x00 | <Sequence number: uint8_t> <endpoint: uint8_t> <Cluster id: uint16_t> <status: uint8_t> <Group id: uint16_t> | Status |
| Host->Node | View Group Msg Type = 0x0061 Command ID = 0x01 | <address mode: uint8_t> <target short address: uint16_t> <source endpoint: uint8_t> <destination endpoint: uint8_t> <group address: uint16_t > | Status View Group response |
| Node->Host | View Group response Message Type = 0x8061 Command ID = 0x01 | <Sequence number: uint8_t> <endpoint: uint8_t> <Cluster id: uint16_t> <status: uint8_t> <Group id :uint16_t> | |
| Host->Node | Get Group Membership Msg Type = 0x0062 | <address mode: uint8_t> <target short address: uint16_t> <source endpoint: uint8_t> | Status |

| | | | |
|---|---|---|---|
| | Command ID = 0x02 | <destination endpoint: uint8_t><br><group count: uint8_t><br><group list:data> | Get Group Membership response |
| Node->Host | Get Group Membership response<br>Msg Type = 0x8062<br>Command ID = 0x02 | <Sequence number: uint8_t><br><endpoint: uint8_t><br><Cluster id: uint16_t><br><capacity: uint8_t><br><Group count: uint8_t><br><List of Group id: list each data item uint16_t> | |
| Host->Node | Remove Group<br>Msg Type = 0x0063<br>Command ID = 0x03 | <address mode: uint8_t><br><target short address: uint16_t><br><source endpoint: uint8_t><br><destination endpoint: uint8_t><br><group address: uint16_t > | Status<br>Remove Group response |
| Node->Host | Remove  Group response<br>Msg Type = 0x8063<br>Command ID = 0x03 | <Sequence number: uin8_t><br><endpoint: uint8_t><br><Cluster id: uint16_t><br><status: uint8_t><br><Group id: uint16_t> | Status |
| Host->Node | Remove All Groups<br>Msg Type = 0x0064<br>Command ID = 0x04 | <address mode: uint8_t><br><target short address: uint16_t><br><source endpoint: uint8_t><br><destination endpoint: uint8_t> | Status |
| Host->Node | Add Group if identify<br>Msg Type = 0x0065<br>Command ID = 0x05 | <address mode: uint8_t><br><target short address: uint16_t><br><source endpoint: uint8_t><br><destination endpoint: uint8_t><br><group address: uint16_t > | Status |

## B.1.4. Identify Cluster Commands

| Message Direction | Message Description | Message Format | Expected Response |
|---|---|---|---|
| Host->Node | Identify Send<br>Msg Type = 0x0070 | <address mode: uint8_t><br><target short address: uint16_t><br><source endpoint: uint8_t><br><destination endpoint: uint8_t><br><time: uint16_t><br>Time: Seconds | Status |
| Host->Node | Identify Query<br>Msg Type = 0x0071 | <address mode: uint8_t><br><target short address: uint16_t><br><source endpoint: uint8_t><br><destination endpoint: uint8_t> | Status |

## B.1.5. Level Cluster Commands

| Message Direction | Message Description | Message Format | Expected Response |
|---|---|---|---|
| Host->Node | Move to Level<br>Msg Type = 0x0080 | <address mode: uint8_t><br><target short address: uint16_t><br> <source endpoint: uint8_t><br><destination endpoint: uint8_t><br><onoff: uint8_t><br><mode: uint8_t><br><rate: uint8_t> | Status |

| Host->Node | Move to level with/without on/off Msg Type = 0x0081 | \<address mode: uint8_t\> \<target short address: uint16_t\> \<source endpoint: uint8_t\> \<destination endpoint: uint8_t\> \<onoff : uint8_t\> \<Level: uint8_t \> \<Transition Time: uint16_t\> | Status |
|---|---|---|---|
| Host->Node | Move Step Msg Type = 0x0082 | \<address mode: uint8_t\> \<target short address: uint16_t\> \<source endpoint: uint8_t\> \<destination endpoint: uint8_t\> \<onoff: uint8_t\> \<step mode: uint8_t \> \<step size: uint8_t\> \<Transition Time: uint16_t\> | Status |
| Host->Node | Move Stop Move Msg Type = 0x0083 | \<address mode: uint8_t\> \<target short address: uint16_t\> \<source endpoint: uint8_t\> \<destination endpoint: uint8_t\> | Status |
| Host->Node | Move Stop with On Off Msg Type = 0x0084 | \<address mode: uint8_t\> \<target short address: uint16_t\> \<source endpoint: uint8_t\> \<destination endpoint: uint8_t\> | Status |

## B.1.6. On/Off Cluster Commands

| Message Direction | Message Description | Message Format | Expected Response |
|---|---|---|---|
| Host->Node | On / Off with effects Send Msg Type = 0x0094 | \<address mode: uint8_t\> \<target short address: uint16_t\> \<source endpoint: uint8_t\> \<destination endpoint: uint8_t\> \<effect ID: uint8_t\> \<effect gradient: uint8_t\> | Status |
| Host->Node | On/Off with no effects Msg Type = 0x0092 | \<address mode: uint8_t\> \<target short address: uint16_t\> \<source endpoint: uint8_t\> \<destination endpoint: uint8_t\> \<command ID: uint8_t\> Command Id    0 - Off    1 - On    2 - Toggle | Status |
| Host->Node | On / Off Timed Send Msg Type = 0x0093 | \<address mode: uint8_t\> \<target short address: uint16_t\> \<source endpoint: uint8_t\> \<destination endpoint: uint8_t\> \<onoff: uint8_t\> \<on time: uint16_t\> \<off time: uint16_t\> On / Off:    0 = Off    1 = On Time: Seconds | Status |

## B.1.7. Scenes Cluster Commands

| Message Direction | Message Description | Message Format | Expected Response |
|---|---|---|---|
| Host->Node | View Scene<br>Msg Type = 0x00A0 | <address mode: uint8_t><br><target short address: uint16_t><br><source endpoint: uint8_t><br><destination endpoint: uint8_t><br><group ID: uint16_t><br><scene ID: uint8_t> | Status<br>View Scene response |
| Node->Host | View Scene response<br>Msg Type = 0x80A0 | <sequence number: uint8_t><br><endpoint : uint8_t><br><cluster id: uint16_t><br><status: uint8_t><br><group ID: uint16_t><br><scene ID: uint8_t><br><transition time: uint16_t><br><scene name length: uint8_t><br><scene name max length: uint8_t><br><scene name  data: data each element is uint8_t><br><extensions length: uint16_t><br><extensions max length: uint16_t><br><extensions data: data each element is uint8_t> | |
| Host->Node | Add Scene<br>Msg Type = 0x00A1 | <address mode: uint8_t><br><target short address: uint16_t><br><source endpoint: uint8_t><br><destination endpoint: uint8_t><br><group ID: uint16_t><br><scene ID: uint8_t><br><transition time: uint16_t><br><scene name length: uint8_t><br><scene name max length: uint8_t><br><scene name data: data each element is uint8_t> | Status<br>Add Scene response |
| Node->Host | Add Scene response<br>Msg Type = 0x80A1 | <sequence number: uint8_t><br><endpoint : uint8_t><br><cluster id: uint16_t><br><status: uint8_t><br><group ID: uint16_t><br><scene ID: uint8_t> | |
| Host->Node | Remove Scene<br>Msg Type = 0x00A2 | <address mode: uint8_t><br><target short address: uint16_t><br><source endpoint: uint8_t><br><destination endpoint: uint8_t><br><group ID: uint16_t><br><scene ID: uint8_t> | Status<br>Remove Scene response |
| Node->Host | Remove Scene response<br>Msg Type = 0x80A2 | <sequence number: uint8_t><br><endpoint : uint8_t><br><cluster id: uint16_t><br><status: uint8_t><br><group ID: uint16_t><br><scene ID: uint8_t> | |
| Host->Node | Remove all scenes<br>Msg Type = 0x00A3 | <address mode: uint8_t><br><target short address: uint16_t><br><source endpoint: uint8_t><br><destination endpoint: uint8_t><br><group ID: uint16_t> | Status<br>Data indication |
| Node->Host | Remove  All Scene response<br>Msg Type = 0x80A3 | <sequence number: uint8_t><br><endpoint : uint8_t><br><cluster id: uint16_t><br><status: uint8_t> | |

| | | | |
|---|---|---|---|
| | | <group ID: uint16_t> | |
| Host->Node | Store Scene<br>Msg Type = 0x00A4 | <address mode: uint8_t><br><target short address: uint16_t><br><source endpoint: uint8_t><br><destination endpoint: uint8_t><br><group ID: uint16_t><br><scene ID: uint8_t> | Status<br>Data indication |
| Node->Host | Store Scene response<br>Msg Type = 0x80A4 | <sequence number: uint8_t><br><endpoint : uint8_t><br><cluster id: uint16_t><br><status: uint8_t><br><group ID: uint16_t><br><scene ID: uint8_t> | |
| Host->Node | Recall Scene<br>Msg Type = 0x00A5 | <address mode: uint8_t><br><target short address: uint16_t><br><source endpoint: uint8_t><br><destination endpoint: uint8_t><br><group ID: uint16_t><br><scene ID: uint8_t> | Status<br>Data indication |
| Host->Node | Scene Membership request<br>Msg Type = 0x00A6 | <address mode: uint8_t><br><target short address: uint16_t><br><source endpoint: uint8_t><br><destination endpoint: uint8_t><br><group ID: uint16_t> | Status<br>Data indication |
| Node->Host | Scene Membership response<br>Msg Type = 0x80A6 | <sequence number: uint8_t><br><endpoint : uint8_t><br><cluster id: uint16_t><br><status: uint8_t><br><capacity: uint8_t><br><group ID: uint16_t><br><scene count: uint8_t><br><scene list: data each element uint8_t> | Status<br>Data indication |

## B.1.8. Colour Cluster Commands

| Message Direction | Message Description | Message Format | Expected Response |
|---|---|---|---|
| Host->Node | Move to Hue<br>Msg Type = 0x00B0 | <address mode: uint8_t><br><target short address: uint16_t><br><source endpoint: uint8_t><br><destination endpoint: uint8_t><br><hue: uint8_t><br><direction: uint8_t><br><transition time: uint16_t> | Status<br>Data indication |
| Host->Node | Move Hue<br>Msg Type = 0x00B1 | <address mode: uint8_t><br><target short address: uint16_t><br><source endpoint: uint8_t><br><destination endpoint: uint8_t><br><mode: uint8_t><br><rate: uint8_t> | Status<br>Data indication |
| Host->Node | Step Hue<br>Msg Type = 0x00B2 | <address mode: uint8_t><br><target short address: uint16_t><br><source endpoint: uint8_t><br><destination endpoint: uint8_t><br><mode: uint8_t><br><step size: uint8_t><br><transition time: uint8_t> | Status<br>Data indication |
| Host->Node | Move to saturation<br>Msg Type = 0x00B3 | <address mode: uint8_t><br><target short address: uint16_t><br><source endpoint: uint8_t><br><destination endpoint: uint8_t><br><saturation: uint8_t><br><transition time: uint16_t> | Status<br>Data indication |
| Host->Node | Move saturation<br>Msg Type = 0x00B4 | <address mode: uint8_t><br><target short address: uint16_t><br><source endpoint: uint8_t><br><destination endpoint: uint8_t><br><mode: uint8_t><br><rate: uint8_t> | Status<br>Data indication |
| Host->Node | Step saturation<br>Msg Type = 0x00B5 | <address mode: uint8_t><br><target short address: uint16_t><br><source endpoint: uint8_t><br><destination endpoint: uint8_t><br><mode: uint8_t><br><step size: uint8_t><br><transition time: uint8_t> | Status<br>Data indication |
| Host->Node | Move to hue and saturation<br>Msg Type = 0x00B6 | <address mode: uint8_t><br><target short address: uint16_t><br><source endpoint: uint8_t><br><destination endpoint: uint8_t><br><hue: uint8_t><br><saturation: uint8_t><br><transition time: uint16_t> | Status<br>Data indication |
| Host->Node | Move to colour<br>Msg Type = 0x00B7 | <address mode: uint8_t><br><target short address: uint16_t><br><source endpoint: uint8_t><br><destination endpoint: uint8_t><br><colour X: uint16_t><br><colour Y: uint16_t><br><transition time: uint16_t > | Status<br>Data indication |

| Host->Node | Move Colour<br>Msg Type = 0x00B8 | <address mode: uint8_t><br><target short address: uint16_t><br><source endpoint: uint8_t><br><destination endpoint: uint8_t><br><colour X: int16_t><br><colour Y: int16_t> | Status<br>Data indication |
| --- | --- | --- | --- |
| Host->Node | Step Colour<br>Msg Type = 0x00B9 | <address mode: uint8_t><br><target short address: uint16_t><br><source endpoint: uint8_t><br><destination endpoint: uint8_t><br><step X: int16_t><br><step Y: int16_t><br><transition time: uint16_t > | Status<br>Data indication |

# B.2. ZLO/ZLL-specific Commands

## B.2.1. Touchlink Commands

| Message Direction | Message Description | Message Format | Expected Response |
| --- | --- | --- | --- |
| Host->Node | Initiate Touchlink<br>Msg Type = 0x00D0 | No Payload | Status |
| Host->Node | Touch link factory reset target<br>Msg Type= 0x00D2 | No Payload | Status |
| Node->Host | Touchlink Status<br>Msg Type = 0x00D1 | <status: uint8_t><br><joined node short address: uint16_t><br>Status<br>       0 = Success<br>       1 = Failure | |

## B.2.2. Identify Cluster Commands

| Message Direction | Message Description | Message Format | Expected Response |
| --- | --- | --- | --- |
| Host->Node | Identify Trigger Effect<br>Msg Type = 0x00E0 | <address mode: uint8_t><br><target short address: uint16_t><br><source endpoint: uint8_t><br><destination endpoint: uint8_t><br><effect ID: uint8_t><br><effect gradient: uint8_t > | Status<br>Data indication |

## B.2.3. On/Off Cluster Commands

| Message Direction | Message Description | Message Format | Expected Response |
|---|---|---|---|
| Host->Node | On / Off with Effects Msg Type = 0x0092 | <address mode: uint8_t><br><target short address: uint16_t><br><source endpoint: uint8_t><br><destination endpoint: uint8_t><br><effect ID: uint8_t><br><effect gradient: uint8_t> | Status Data indication |
| Host->Node | On / Off Timed Msg Type = 0x0093 | <address mode: uint8_t><br><target short address: uint16_t><br><source endpoint: uint8_t><br><destination endpoint: uint8_t><br><onoff: uint8_t><br><on time: uint8_t><br><off time: uint8_t> | Status Data indication |

## B.2.4. Scenes Cluster Commands

| Message Direction | Message Description | Message Format | Expected Response |
|---|---|---|---|
| Host->Node | Add Enhanced Scene Msg Type = 0x00A7 | <address mode: uint8_t><br><target short address: uint16_t><br><source endpoint: uint8_t><br><destination endpoint: uint8_t><br><group ID: uint16_t><br><scene ID: uint8_t><br><transition time: uint16_t><br><scene name: string><br><length: uint16_t><br><max length: uint16_t><br><data: data> | Status Data indication |
| Host->Node | View Enhanced Host->Node Scene Msg Type = 0x00A8 | <address mode: uint8_t><br><target short address: uint16_t><br><source endpoint: uint8_t><br><destination endpoint: uint8_t><br><group ID: uint16_t><br><scene ID: uint8_t> | Status Data indication |
| Host->Node | Copy Scene Msg Type = 0x00A9 | <address mode: uint8_t><br><target short address: uint16_t><br><source endpoint: uint8_t><br><destination endpoint: uint8_t><br><mode: uint8_t><br><from group ID: uint16_t><br><from scene ID: uint8_t><br><to group ID: uint16_t><br><to scene ID: uint8_t> | Status Data indication |

## B.2.5. Colour Cluster Commands

| Message Direction | Message Description | Message Format | Expected Response |
|---|---|---|---|
| Host->Node | Enhanced Move to Hue<br>Msg Type = 0x00BA | &lt;address mode: uint8_t&gt;<br>&lt;target short address: uint16_t&gt;<br>&lt;source endpoint: uint8_t&gt;<br>&lt;destination endpoint: uint8_t&gt;<br>&lt;direction: uint8_t&gt;<br>&lt;Enhanced Hue: uint16_t&gt;<br>&lt;transition time: uint16_t&gt; | Status<br>Data indication |
| Host->Node | Enhanced Move Hue<br>Msg Type = 0x00BB | &lt;address mode: uint8_t&gt;<br>&lt;target short address: uint16_t&gt;<br>&lt;source endpoint: uint8_t&gt;<br>&lt;destination endpoint: uint8_t&gt;<br>&lt;mode: uint8_t&gt;<br>&lt;rate: uint8_t&gt; | Status<br>Data indication |
| Host->Node | Enhanced Step Hue<br>Msg Type = 0x00BC | &lt;address mode: uint8_t&gt;<br>&lt;target short address: uint16_t&gt;<br>&lt;source endpoint: uint8_t&gt;<br>&lt;destination endpoint: uint8_t&gt;<br>&lt;mode: uint8_t&gt;<br>&lt;step size: uint8_t&gt;<br>&lt;transition time: uint8_t&gt; | Status<br>Data indication |
| Host->Node | Enhanced Move to hue and saturation<br>Msg Type = 0x00BD | &lt;address mode: uint8_t&gt;<br>&lt;target short address: uint16_t&gt;<br>&lt;source endpoint: uint8_t&gt;<br>&lt;destination endpoint: uint8_t&gt;<br>&lt;enhanced hue: uint32_t&gt;<br>&lt;saturation: uint32_t&gt;<br>&lt;transition time: uint8_t&gt; | Status<br>Data indication |
| Host->Node | Colour Loop Set<br>Msg Type = 0x00BE | &lt;address mode: uint8_t&gt;<br>&lt;target short address: uint16_t&gt;<br>&lt;source endpoint: uint8_t&gt;<br>&lt;destination endpoint: uint8_t&gt;<br>&lt;update flags: uint8_t&gt;<br>&lt;action: uint8_t&gt;<br>&lt;direction: uint8_t&gt;<br>&lt;time: uint8_t&gt;<br>&lt;start hue: uint32_t&gt; | Status<br>Data indication |
| Host->Node | Stop Move Step<br>Msg Type = 0x00BF | &lt;address mode: uint8_t&gt;<br>&lt;target short address: uint16_t&gt;<br>&lt;source endpoint: uint8_t&gt;<br>&lt;destination endpoint: uint8_t&gt; | Status<br>Data indication |
| Host->Node | Move to colour temperature<br>Msg Type = 0x00C0 | &lt;address mode: uint8_t&gt;<br>&lt;target short address: uint16_t&gt;<br>&lt;source endpoint: uint8_t&gt;<br>&lt;destination endpoint: uint8_t&gt;<br>&lt;colour temperature: uint16_t&gt;<br>&lt;transition time: uint16_t&gt; | Status<br>Data indication |
| Host->Node | Move colour temperature<br>Msg Type = 0x00C1 | &lt;address mode: uint8_t&gt;<br>&lt;target short address: uint16_t&gt;<br>&lt;source endpoint: uint8_t&gt;<br>&lt;destination endpoint: uint8_t&gt;<br>&lt;mode: uint8_t&gt;<br>&lt;rate: uint16_t&gt;<br>&lt;minimum temperature: uint16_t&gt;<br>&lt;maximum temperature: uint16_t&gt;<br>&lt;options mask: uint8_t&gt;<br>&lt;options override: uint8_t&gt; | Status<br>Data indication |

| | | | |
|---|---|---|---|
| Host->Node | Step colour temperature<br>Msg Type = 0x00C2 | <address mode: uint8_t><br><target short address: uint16_t><br><source endpoint: uint8_t><br><destination endpoint: uint8_t><br><mode: uint8_t><br><step size: uint16_t><br><transition time: uint16_t><br><minimum temperature: uint16_t><br><maximum temperature: uint16_t> | Status<br>Data indication |

# B.3. ZHA-specific Commands

## B.3.1. Door Lock Cluster Commands

| Message Direction | Message Description | Message Format | Expected Response |
|---|---|---|---|
| Host->Node | Lock / Unlock Door<br>Msg Type = 0x00F0 | <address mode: uint8_t><br><target short address: uint16_t><br> <source endpoint: uint8_t><br><destination endpoint: uint8_t><br><lock/unlock: uint8_t><br>      0 = Lock<br>      1 = Unlock | Status<br>Data indication |

## B.3.2  IAS Cluster Commands

| Message Direction | Message Description | Message Format | Expected Response |
|---|---|---|---|
| Host->Node | IAS Zone enroll response<br>Msg Type = 0x0400 | <address mode: uint8_t><br><target short address: uint16_t><br> <source endpoint: uint8_t><br><destination endpoint: uint8_t><br><Enroll response code: uint8_t><br><Zone id : uint8_t> | Status |
| Node->Host | Zone status change notification<br>Msg Type = 0x8401 | <sequence number: uint8_t><br><endpoint : uint8_t><br><cluster id: uint16_t><br><src address mode: uint8_t><br><src address: uint64_t  or uint16_t based on address mode><br><zone status: uint16_t><br><extended status: uint8_t><br><zone id : uint8_t><br><delay: data each element uint16_t> | |

## B.4. Exporting Persistent Data to Host

The ZigBee Control Bridge node by default uses the internal EEPROM to hold persisted data. This is about 512Kbytes on a KW41Z device. Some applications may require more than this. To address this situation, the possibility to export the data persistence to the host device was added. This requires a binary with this feature turned "ON".

The host needs to provide message handshaking sequence to achieve this.  How the host actually stores the persisted data is beyond the scope of the document.

| Message Direction | Message Description | Message Format | Expected Response |
|---|---|---|---|
| Node->Host | Host Persistent Data manager available Request Msg Type = 0x0300 | Node enquires about the availability of the Host PDM. | Host persistent Data manager available response |
| Host->Node | Host persistent Data manager available response Msg Type = 0x8300 | The Host must send this as the first message to allow the Node to continue operation. | |
| Node->Host | Load Record Request Msg Type = 0x0201 | <Record Id : uint16_t> | Load Record response |
| Host->Node | Load Record response Msg Type = 0x8201 | <status: uint8_t><br><Record Id: uint16_t><br><total size: uint32_t><br><total number of blocks: uint32_t><br><current block: uint32_t><br><block size:  uint32_t><br><data: variable list each item is uint8_t><br>status:<br>      0- no record found<br>      1- Record recovered | Status |
| Node->Host | Save  Record request Msg Type = 0x0200 | <Record Id:  uint16_t><br><total size:  uint32_t><br><total number of blocks: uint32_t><br><current block: uint32_t><br><block size:  uint32_t><br><data: variable list, each item is uint8_t> | Save  Record response |
| Host->Node | Save  Record response Msg Type = 0x8200 | <Record Id: uint16_t><br><total size: uint32_t><br><total number of blocks: uint32_t><br><current block: uint32_t><br><block size: uint32_t> | |
| Node->Host | Delete all records Msg Type = 0x0202 | | |

## B.5. Extended Utilities

The ZigBee Control Bridge also has some extra commands that are sent or received which provide extra debug or features.

| Message Direction | Message Description | Message Format | Expected Response |
|---|---|---|---|
| Host->Node | Raw APS Data Request<br>Msg Type = 0x0530 | <address mode: uint8_t><br><target short address: uint16_t><br> <source endpoint: uint8_t><br><destination endpoint: uint8_t><br><profile ID: uint16_t><br><cluster ID: uint16_t><br><security mode: uint8_t><br><radius: uint8_t><br><data length: uint8_t><br><data: auint8_t> | Status |
| Node->Host | Router Discovery Confirm<br>Msg Type = 0x8701 | <status: uint8_t><br><nwk status: uint8_t> | |
| Node->Host | APS Data Confirm Fail<br>Msg Type = 0x8702 | <status: uint8_t><br><src endpoint: uint8_t><br><dst endpoint: uint8_t><br><dst address mode: uint8_t><br><destination address: uint64_t><br><seq number: uint8_t> | |

# Appendix C: Use Case Sequences

## C.1. Gateway Start-up

The following sequence of messages is exchanged at start-up. In the tables below, the Node refers to the Control Bridge.

| Direction | Message |
|---|---|
| Host->Node | Erase Persistent Data (Optional) |
| Node->Host | Status (If Erase command issued) |
| Host->Node | Reset |
| Node->Host | Status |
| Node->Host | Node Cluster List (multiple) |
| Node->Host | Node Attribute List (multiple) |
| Node->Host | Node Command ID List (multiple) |
| Host->Node | Get Version |
| Node->Host | Status |
| Node->Host | Version List |
| Host->Node | Set Extended PANID |
| Node->Host | Status |
| Host->Node | Set Channel Mask |
| Node->Host | Status |
| Host->Node | Set Security State & Key |
| Node->Host | Status |
| Host->Node | Set Device Type |
| Node->Host | Status |
| Host->Node | Start Network |
| Node->Host | Status |
| Node->Host | Network Formed / Joined |

## C.2. Touchlink Initiated by Another Control Node

| Direction | Message |
|---|---|
| Host->Node | Erase Persistent Data (Optional) |
| Node->Host | Status (If Erase command issued) |
| Host->Node | Reset |
| Node->Host | Status |
| Node->Host | Node Cluster List (multiple) |
| Node->Host | Node Attribute List (multiple) |
| Node->Host | Node Command ID List (multiple) |
| Host->Node | Get Version |
| Node->Host | Status |
| Node->Host | Version List |
| Host->Node | Set Extended PANID |
| Node->Host | Status |
| Host->Node | Set Channel Mask |
| Node->Host | Status |
| Host->Node | Set Security State & Key |
| Node->Host | Status |
| Host->Node | Set Device Type |
| Node->Host | Status |
| Host->Node | Start scan |
| Node->Host | Status |
| Node->Host | Network Joined/Failed |
| Node->Host | Touchlink status |
| Node->Host | Network formed |

## C.3. Network Formation and Join Under Control of Host

| Direction | Message |
|---|---|
| Host->Node | Erase Persistent Data (Optional) |
| Node->Host | Status (If Erase command issued) |
| Host->Node | Reset |
| Node->Host | Status |
| Node->Host | Node Cluster List (multiple) |
| Node->Host | Node Attribute List (multiple) |
| Node->Host | Node Command ID List (multiple) |
| Host->Node | Get Version |
| Node->Host | Status |
| Node->Host | Version List |
| Host->Node | Set Extended PANID |
| Node->Host | Status |
| Host->Node | Set Channel Mask |
| Node->Host | Status |
| Host->Node | Set Security State & Key |
| Node->Host | Status |
| Host->Node | Set Device Type |
| Node->Host | Status |
| Host->Node | Start scan |
| Node->Host | Status |
| Node->Host | Network Joined/Failed |
| Host->Node | Start form |
| Node->Host | Network formed |

## C.4. Touchlink Initiated by Host

| Direction | Message |
|---|---|
| Host->Node | Erase Persistent Data (Optional) |
| Node->Host | Status (If Erase command issued) |
| Host->Node | Reset |
| Node->Host | Status |
| Node->Host | Node Cluster List (multiple) |
| Node->Host | Node Attribute List (multiple) |
| Node->Host | Node Command ID List (multiple) |
| Host->Node | Get Version |
| Node->Host | Status |
| Node->Host | Version List |
| Host->Node | Set Extended PANID |
| Node->Host | Status |
| Host->Node | Set Channel Mask (Set Primary channels 11,15,20,25) |
| Node->Host | Status |
| Host->Node | Set Security State & Key |
| Node->Host | Status |
| Host->Node | Set Device Type |
| Node->Host | Status |
| Host->Node | Start scan |
| Node->Host | Status |
| Node->Host | Network Joined/Failed |
| Host->Node | Initiate Touchlink |
| Node->Host | Touchlink status |
| Node->Host | Network formed |

## C.5. Warm Restart

| Direction | Message |
|---|---|
| Node->Host | Warm restart status |

## C.6. Join Notification - Device Joining Network Formed by Gateway

| Direction | Message |
|---|---|
| Node->Host | New device joined indication |
| Host->Node | Match descriptor request |
| Node->Host | Status |
| Node->Host | Match descriptor response |
| Host->Node | Add Group |
| Node->Host | Status |
| Host->Node | Identify |
| Node->Host | Status |
| Node->Host | Identify response |

## C.7. Gateway Joins Existing Network

| Direction | Message |
|---|---|
| Host->Node | Match descriptor request (Broadcast) |
| Node->Host | Status |
| Node->Host | Match descriptor response |
| Host->Node | Add Group |
| Node->Host | Status |
| Host->Node | Identify |
| Node->Host | Status |
| Node->Host | Identify response |

## C.8. Binding Control

No sequence required – issue Bind and Unbind commands and get status back

## C.9. Identification

No sequence required – commands and get status back.

For all profiles:

- Identify Send (0x0070)
- Identify Query (0x0071)

For ZLO/ZLL devices:

- Identify Trigger Effect (0x00E0)

## C.10. Scene Management

No sequence required – issue commands and get status back.

For all profiles:

- View Scene (0x00A0)
- Add Scene (0x00A1)
- Remove Scene (0x00A2)
- Remove all scenes (0x00A3)
- Store Scene (0x00A4)
- Recall Scene (0x00A5)
- Scene membership request (0x00A6)

For ZLO/ZLL devices:

- Add Enhanced Scene (0x00A7),
- View Enhanced Scene (0x00A8)
- Copy Scene (0x00A9)

## C.11. Group Management

No sequence required – issue commands and get status back.

- Add Group (0x0060)
- View Group (0x0061)
- Get Group Membership (0x0062)
- Remove Group (0x0063)
- Remove All Groups (0x0064)
- Add Group if identify (0x0065)

## C.12./Off Control

| Direction | Message |
|-----------|---------|
| Host->Node | On / Off Send (0x0090) |
| Node->Host | Status |
| Node->Host | On/Off Indication |

Or

| Direction | Message |
|-----------|---------|
| Host->Node | On / Off Timed Send (0x0091) |
| Node->Host | Status |
| Node->Host | On/Off Indication |

## C.13. Level Control

No sequence required – issue commands and get status back.

- Move to Level (0x0080)
- Move to level with/without On/Off (0x0081)
- Move Step (0x0082)
- Move Stop Move (0x0083)
- Move Stop with On/Off (0x0084)

## C.14. Colour Control

For all profiles:

- Move to Hue (0x00B0)
- Move Hue (0x00B1)
- Step Hue (0x00B2)
- Move to saturation (0x00B3)
- Move saturation (0x00B4)
- Step saturation (0x00B5)
- Move to hue and saturation (0x00B6)
- Move to colour(0x00B7)
- Move Colour (0x00B8)
- Step Colour (0x00B9)

For ZLO/ZLL devices:

- Enhanced Move to Hue (0x00BA)
- Enhanced Move Hue (0x00BB)
- Enhanced Step Hue (0x00BC)
- Enhanced Move to hue and saturation (0x00BD)
- Colour Loop Set (0x00BE)
- Stop Move Step (0x00BF)
- Move to colour temperature (0x00C0)
- Move colour temperature (0x00C1)
- Step colour temperature (0x00C2)

# Revision History

| Version | Notes |
|---------|-------|
| 0 | First KW41Z release |
| 1 | Updates for KW41Z ZigBee 3.0 Alpha/EAR Release |
| 2 | Updates for KW41Z ZigBee 3.0 Beta/PRC Release |
| 3 | Updates for KW41Z ZigBee 3.0 GA/RFP Release |

# Important Notice

**Limited warranty and liability —** Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the *Terms and conditions of commercial sale* of NXP Semiconductors.

**Right to make changes —** NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

**Suitability for use —** NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

**Applications —** Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

**Export control —** This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

All trademarks are the property of their respective owners.

## NXP Semiconductors

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

**www.nxp.com**