Application Note: JN-AN-1247

DK006: JN5189 / JN5188 / K32W061 / K32W041

Zigbee 3.0 IoT Control Bridge

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

The *Zigbee 3.0 Getting Started Application Note [JN-AN-1260]* contains instructions for installing MCUXpresso, DK006 microcontroller SDKs and other required tools to develop with this Application Note.

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 DK006 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 DK006 Development Kits. For information on how to use the Zigbee IoT Control Bridge with the components of these kits, please refer to the relevant *DK006 Development Kit User Guide*.

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 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 [JN-AN-1244]* and other NXP Zigbee 3.0 Application Notes.

2 Capabilities

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 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 Control Bridge supports. It is also provided as source code, so developers can reference the protocol data sent to the Control Bridge to aid faster development.

3 What is Provided

The demonstration package comes with the following components, intended to be used with hardware components in the DK006 kits:

- Documentation (this document)
- Application binaries and source code for the following:
    - Zigbee Control Bridge
    - 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. The light bulb binaries provided in the Application Note *Zigbee 3.0 Light Bulbs [JN-AN-1244]* provide the most visual examples and should be loaded into boards of the DK006 kit (see Setting up the Nodes).

4 Development Environment

4.1 Software

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

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

The MCUXpresso software and installation instructions are described in the *Zigbee 3.0 Getting Started Application Note [JN-AN-1260]*. The *JN-AN-1247 Release Notes* (included in this folder) indicate the versions of MCUXpresso and SDK that you will need to work with this Application Note.

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

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

4.1.1 Compilation for Specific Chips/SDKs

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

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

4.2 Hardware

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

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

5 Running the Demonstration

5.1 Programming the DK006 Device

The following table lists the supplied binary files and the components of the DK006 kit on which they may be used.

Application Binary	Expansion Board (+ Carrier Board)		USB Dongle
	Generic	OM5577/PN7120S	
ControlBridge_Full_GpProxy_1000000.bin	■		■
ControlBridge_Full_NciIcode_GpProxy_1000000.bin		■	

Each binary name provides details of the supported device type, optional functionality and baud rate for the Control Bridge – for example:

- **ControlBridge_Full_GpProxy_1000000.bin** is the Control Bridge binary which supports both Router and Coordinator device types, supports Touchlink commissioning and GP Proxy and supports a 1M baud rate.
- **ControlBridge_Full_NciIcode_GpProxy_1000000.bin** adds NFC Commissioning/Decommissioning features when used on a OM15076-3 Carrier Board fitted with a OM5577/PN7120S or OM5578/PN7150 NFC Controller Expansion Board.

Other variants can be built by altering make variables in the makefile or on the build command line.

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

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

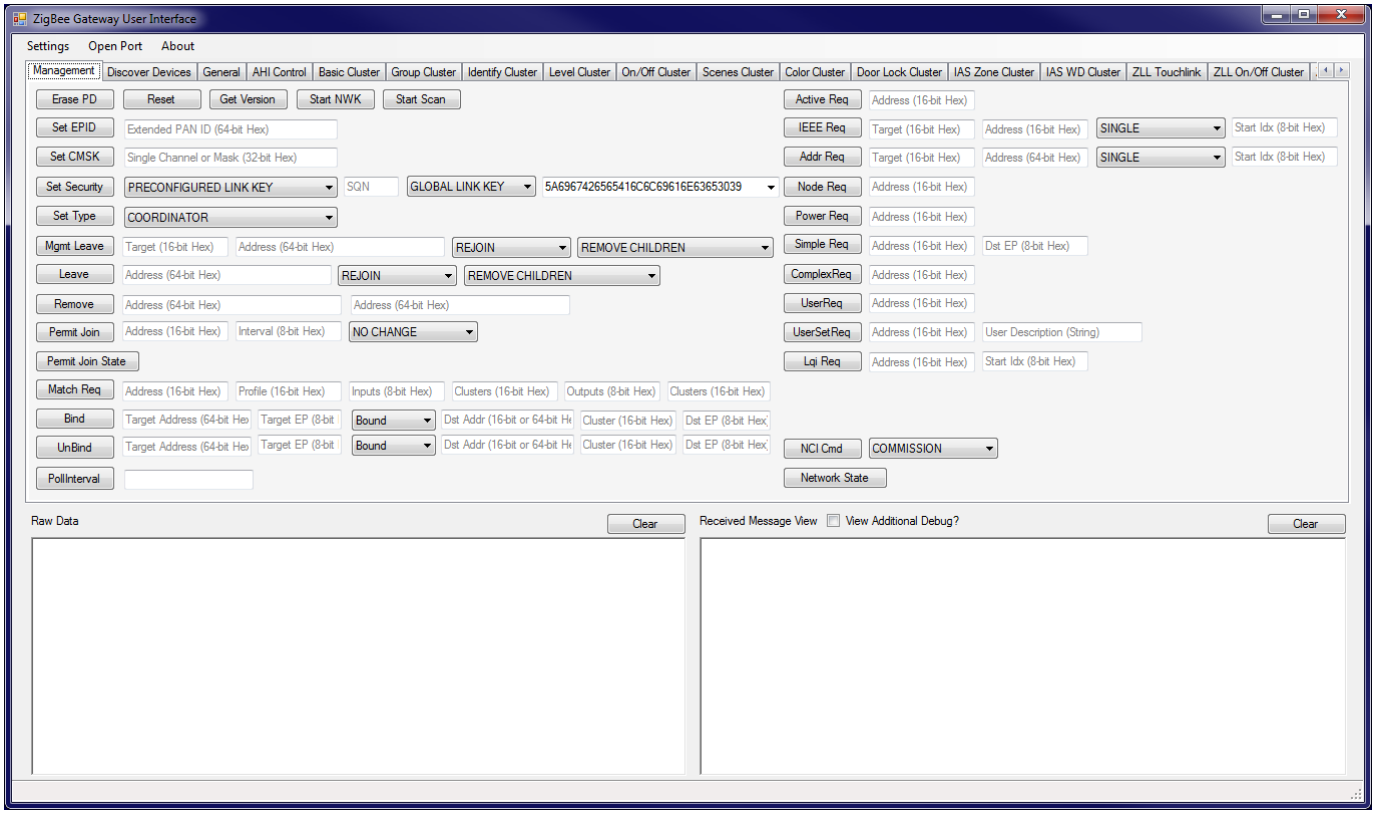
To load an application binary file into a DK006 module on a Carrier Board of a kit, follow the instructions below:

1. Connect a USB port of your PC to the USB Mini B port marked “FTDI USB” on the Carrier Board (next to the 34-pin header) using a ‘USB A to Mini B’ cable. At this point, you may be prompted to install the driver for the USB virtual COM port.
2. Determine which serial communications port on your PC has been allocated to the USB connection.
3. Put the DK006 device into programming mode by holding down the ISP button while pressing and releasing the RESET button.
4. Run the batch (**.bat**) file provided alongside the binary (**.bin**) file to erase the contents of the flash memory including the persistent data stored by the PDM and program the binary file into flash memory. The batch file will prompt for the COM port number to use.
5. Once the download has successfully completed, reset the board or module to run the application.

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

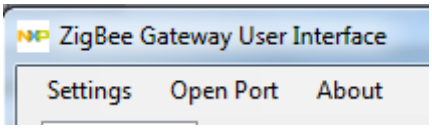
5.2 Running the ZGWUI

The ZGWUI is a C# application 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. The ZGWUI application is located in the folder **Tools/TestGUI/ZGWUI**.

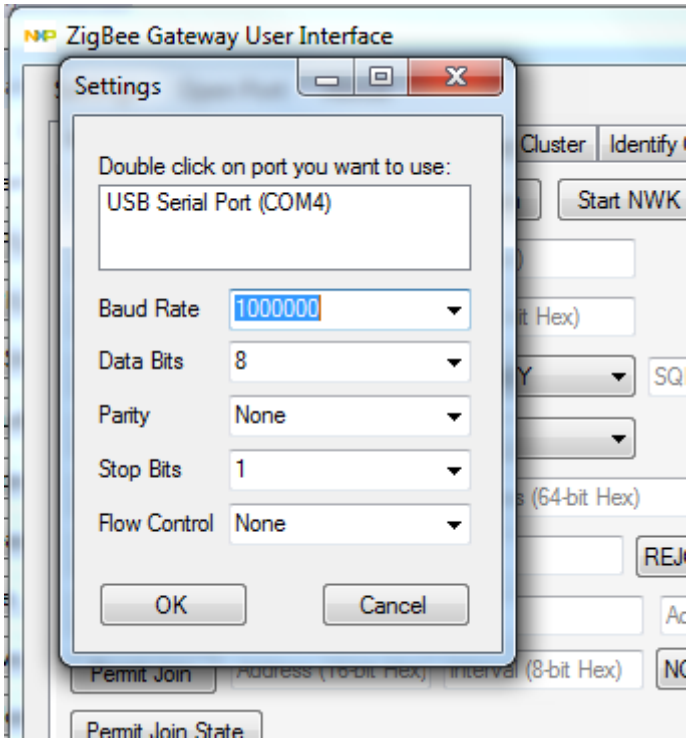


5.2.1 Connecting to the Control Bridge

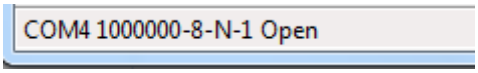
In order to connect to the Control Bridge and issue commands to communicate with Zigbee devices, a serial connection must be set up and opened. To do this, click on **Settings** towards the top-left of the interface.



A pop-up window will appear showing all the available serial connections. Select the correct serial port, configure the baud rate to 1000000, leave all the other settings as default and click **OK**.



Now click the **Open Port** button in the ZGWUI. A serial connection to the Control Bridge will be opened with the status shown in the bottom-left corner of the interface.

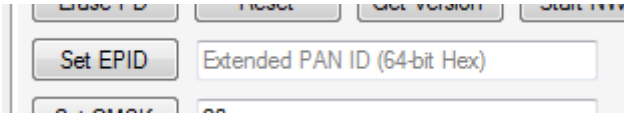


5.2.2 Configuring and Starting a Network

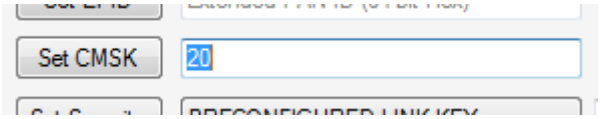
Before initiating a network, some network configuration needs to be done - certain commands need to be run before the network is started, as described below. The description assumes that classical joining will be used to form the network.

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 optional **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.The Control Bridge will select an Extended PAN ID if this command is skipped.

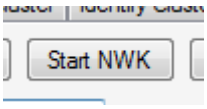


The optional “Set Channel Mask” command informs the Control Bridge which channels the network can start on. The Control Bridge will then choose the best channel available. The **Set CMSK** textbox can be used to specify either a hexadecimal value for a channel mask of possible channels or a decimal channel number if a fixed channel is to be used. The “Set Channel Mask” command can then be issued by clicking the **Set CMSK** button. The Control Bridge will scan all channels if this command is skipped.

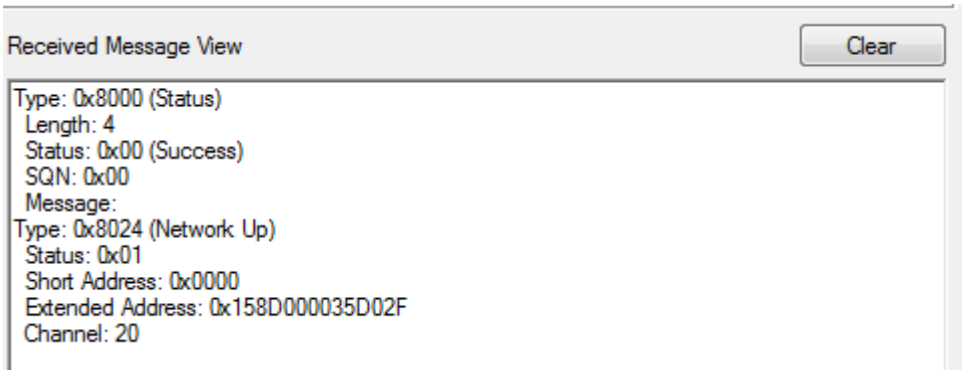


Indicates the network is to start on channel 20 only

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.



5.2.3 Setting up the Nodes

The demonstration requires the OM15076 Carrier Boards (supplied with the DK006 Development Kits) to be configured as lights which can be controlled. Each Carrier Board therefore needs to be fitted with a OM15081 Lighting/Sensor Expansion Board.

Set the jumpers for battery, USB or power supply operation according to how the Carrier boards will be powered during the demonstration. Refer to the *DK006 Development Kit User Guide* for details of the jumper settings.

Plug the Lighting/Sensor Expansion Boards onto the Carrier Boards.

5.2.3.1 Programming the Zigbee Device Binaries

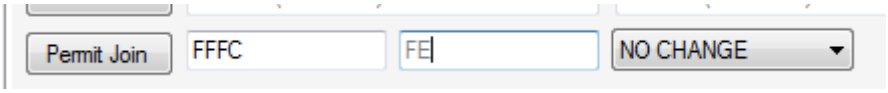
Depending on which type of device and Zigbee network configuration you are demonstrating, you will need to program each light board with an application binary from the Application Note *Zigbee 3.0 Light Bulbs [JN-AN-1244]*. They must be programmed into the devices using the JN518x Flash Programming tool see the relevant SDK Release Notes for details.

5.2.4 Joining Nodes to the Network

The nodes can be joined to the network using either classic join or using NFC (with appropriate hardware).

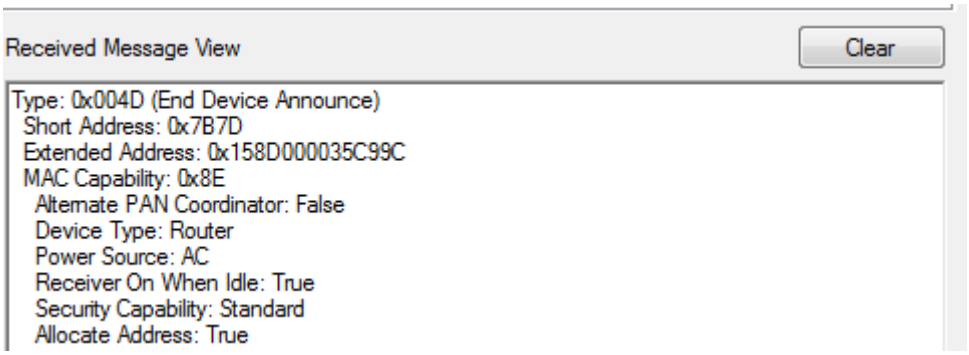
5.2.4.1 Classic Join

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.

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



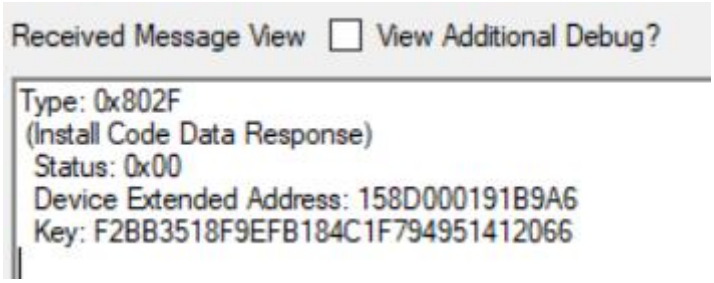
5.2.4.2 Install Codes Join

To successfully join a node to the network using install codes, all steps from **Classic Join** should be done. Before the node attempts to join the network, the MAC address and the

install code of the joining device must be provided to the Trust Centre. In the first (left) **Install Code** textbox, enter the MAC address of the device that will join to the network and in the second (right) **Install Code** textbox, enter the Install Code associated with the node. For information on how to find/change the install code of the joining device, please refer to Zigbee 3.0 Base Device [JN-AN-1243] User Guide or Zigbee 3.0 Light Bulbs [JN-AN1244] User Guide.

Providing the MAC address and the install code of joining node to the Trust Centre

When the MAC address and the Install code of a device is provided to Trust Centre, it will send out an Install Code Data Response message which is captured in the **Received Message View** pane.



By default, only 16 associations between MAC address and Install codes can be provisioned in the Trust Centre at a time. As nodes successfully join the network the MAC address and install code for that device is removed allowing further entires to be added. This number can be customized from **ICODE_MAX_TABLE_SIZE** macro in **app_common.h** file.

The install code table is also held in RAM memory only so power-cycling the Control Bridge will delete these records.

> **!** *The install code for a joining device should be provided to the Control Bridge each time the device is to be newly joined to the network.*

By default, devices can join to the network using classic join or install code join. To permit only the joining of devices that have an install code associated and provided to the Trust Centre, the **JOIN_WITH_INSTALL_CODE_ONLY** macro should set to **TRUE** in **bdb_options.h** file . Also **BDB_JOIN_USES_INSTALL_CODE_KEY** should be set to **TRUE**. To permit only classic join, **BDB_JOIN_USES_INSTALL_CODE_KEY** must be set **FALSE.**

5.2.4.3 NFC Join

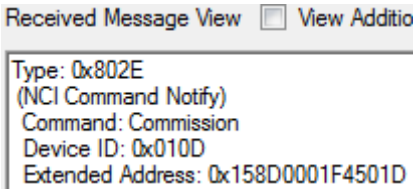
The **ControlBridge_Full_NciIcode_GpProxy_1000000.bin** firmware allows nodes to join the network using NFC. The Control Bridge software should be run on a OM15076-3 Carrier Board fitted with a OM5578/PN7150 NFC Controller Expansion Board. LEDs on the OM5578 expansion board provide feedback.

The Control Bridge will be in NFC Commissioning mode following power on (this can be changed at compile time). When in this mode the green LED will flash slowly indicating that new nodes can be added to the network using NFC.

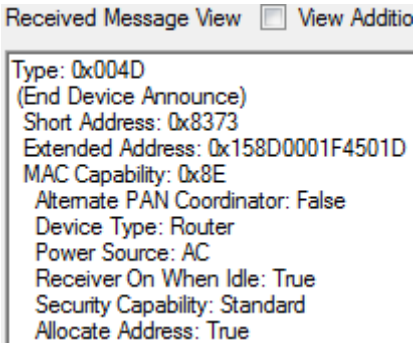
The NFC mode can be altered from the ZGWUI Management tab by selecting the mode from the drop-down then clicking the **NCI Cmd** button:

While in NFC Commissioning mode any of the NTAG enabled devices from the other DK006 Application Notes can be placed over the NFC reader to add them to the network. Whilst the data in the NTAG is being accessed the green LED will flash quickly, if successful it will remain lit until the device is removed from the field when it will revert to flashing slowly. In the case of a failure the green LED will be extinguished until the device is removed from the field (when it will return to flashing slowly).

When the Control Bridge successfully accesses an NTAG it will output a notification on the serial link indicating the type of operation, the Zigbee Device ID and the Extended Address of the device in the **Received Message View** pane. This message could be used to pre-populate databases on the host device in order to present an appropriate GUI for the device.



When the device indicates it is active in the network it will transmit a Device Announce message which will also be displayed in the **Received Message View** pane (in the same way as a Classic Join).



The NFC Decommission mode will factory reset any devices placed into the field (they will leave the network as part of this process). When in this mode the yellow LED is used to provide feedback (with the same indications as used on the green LED for NFC Commissioning mode).

The NFC Disabled mode will not interact with presented NTAGs thus preventing the NFC reader being used to add or remove devices from the network. When in this mode the LEDs are turned off. For extra security it may be desirable to keep the Control Bridge in this mode unless to user specifically enables NFC joining or leaving in some way.

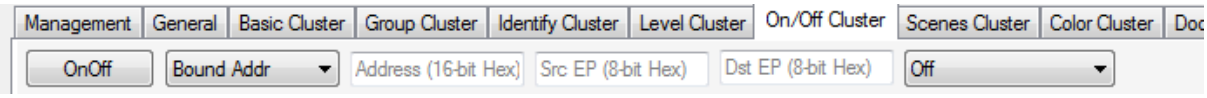
5.2.5 Controlling Devices

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.

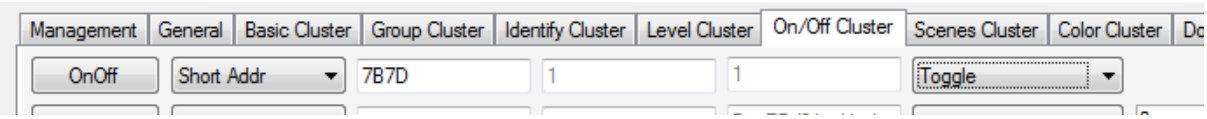
5.2.5.1 On/Off Cluster

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

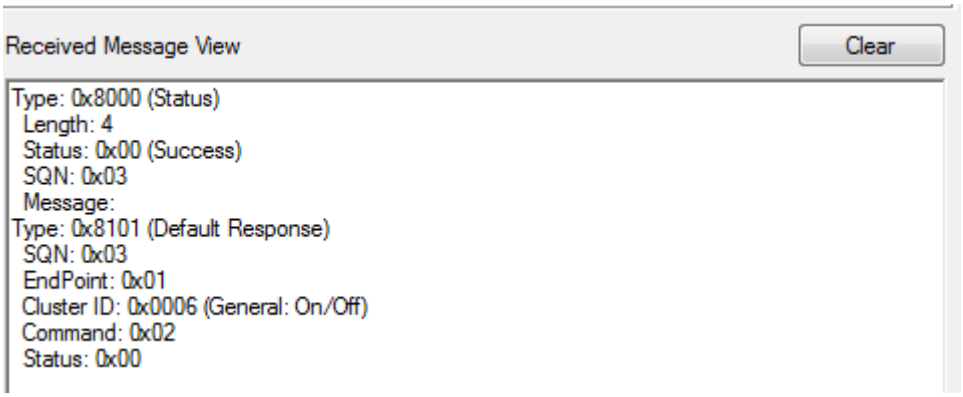
Click on the **On/Off Cluster** tab along the top of the interface.



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 (when using short addresses), the source endpoint number and the destination endpoint number (all in hexadecimal, 1 is used in all the DK006 Application Note devices). Finally, select the type of “On/Off” command that you want to send.



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.



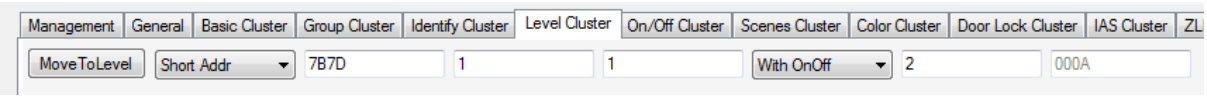
5.2.5.2 Level Control Cluster

The Level Control cluster allows a bulb's dimmable light level to be set to a specific value. This value can be between 0 and 254 (inclusive), and can be set on the **Level Cluster** tab.

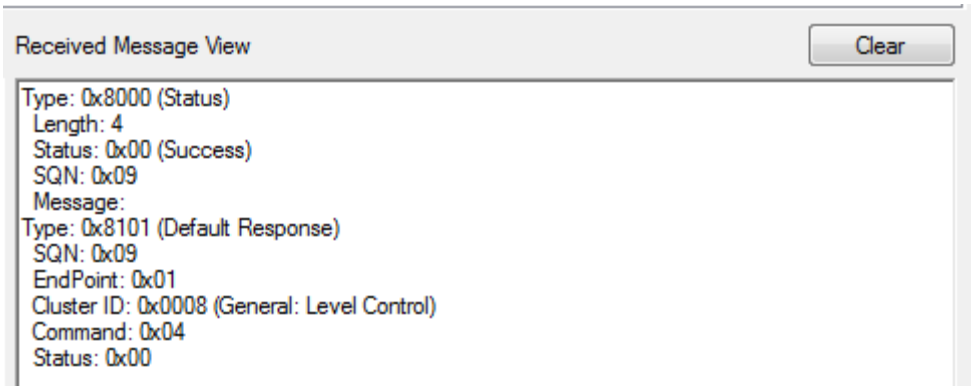
There are a number of attributes that can be passed to the Control Bridge as part of the Level Control cluster's "Move To Level" command:

- 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.

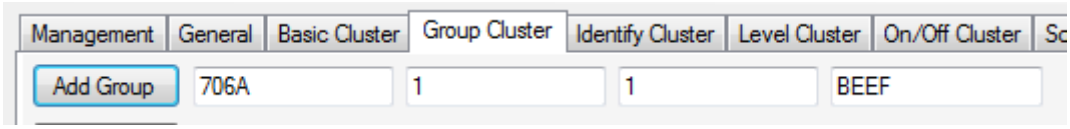


5.2.6 Managing Groups

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.

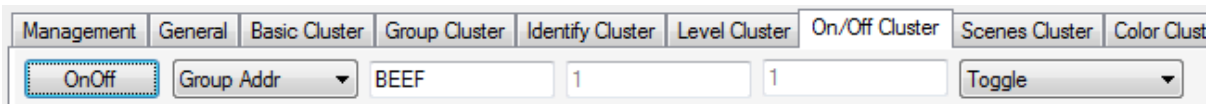
5.2.6.1 Add Group

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.

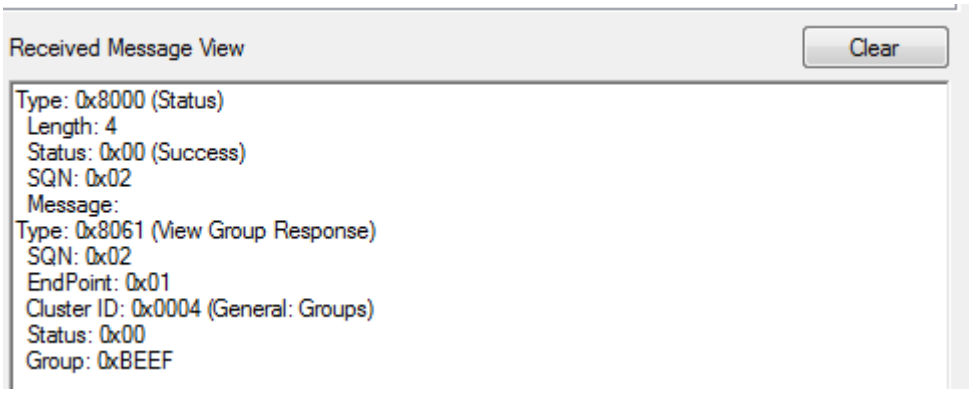
To verify that this group has been added, try sending an “On/Off” command with the group address you have just added. This will toggle the on/off state of the light. Note that since this is a groupcast, a Default Response will not be received.



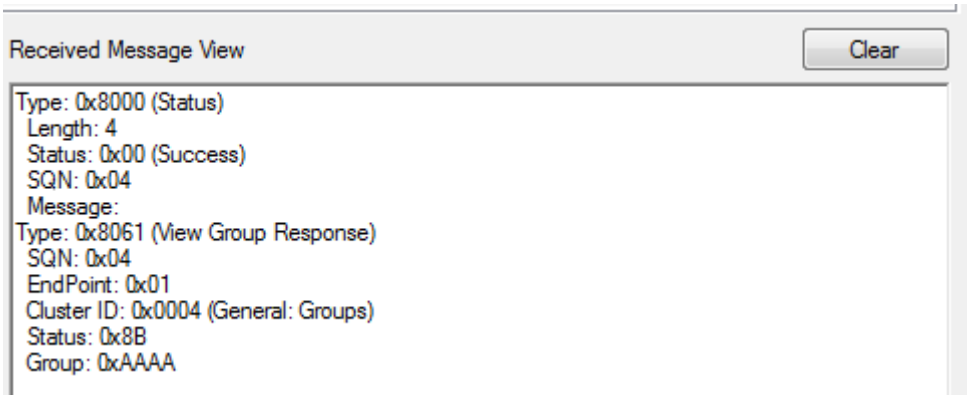
5.2.6.2 View Group

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).

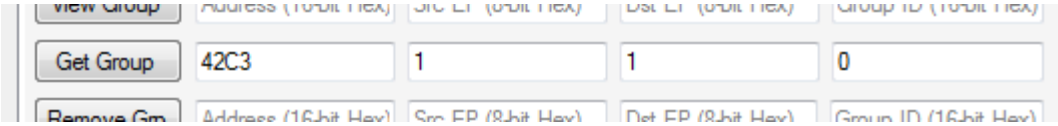


If the device is not a member of that group, you will receive a View Group Response with a status of “Not Found” (0x8B).

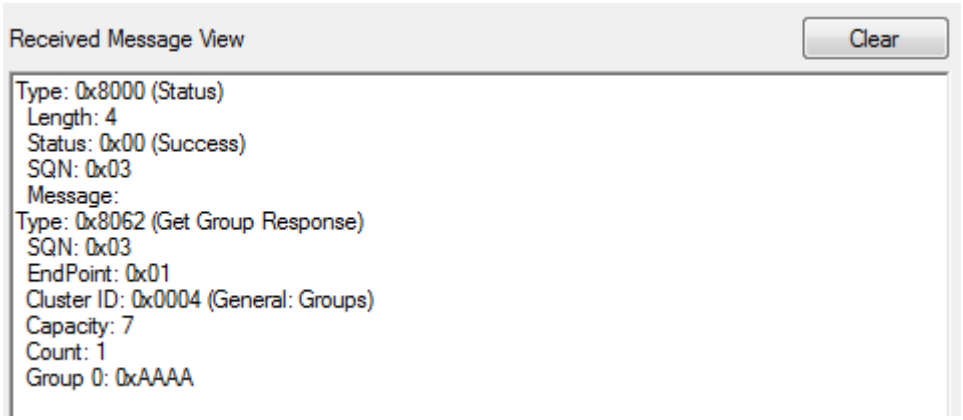


5.2.6.3 Get Group Membership

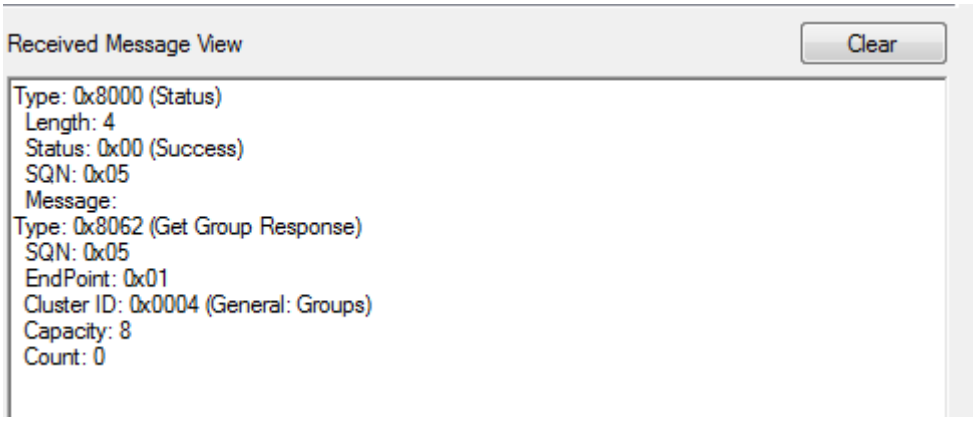
You can find out which groups a specific device is a member of by sending a “Get Group Membership” command to the device. This is done in the **Get Group** line of the interface by entering the network address of the device, source endpoint number, destination endpoint number and group count (number 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.



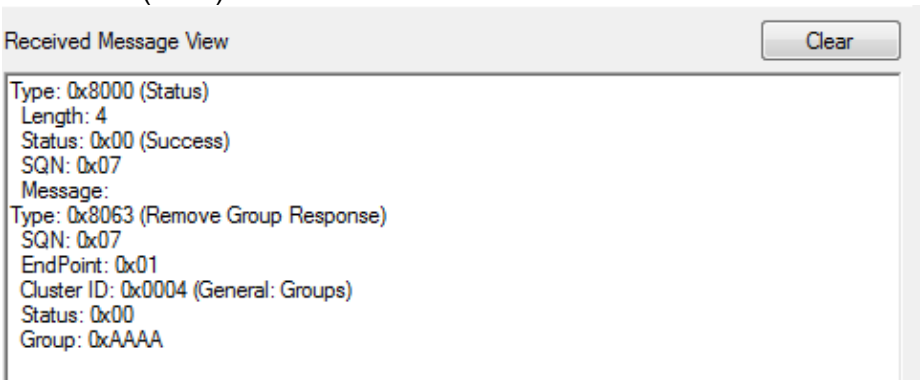
If the device is not a member of any groups, it will respond with an empty group list with a count of 0.



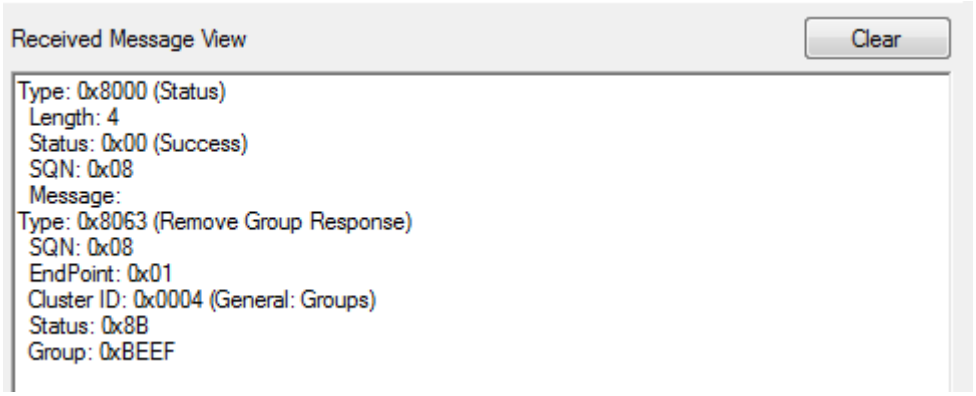
5.2.6.4 Remove Group

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).



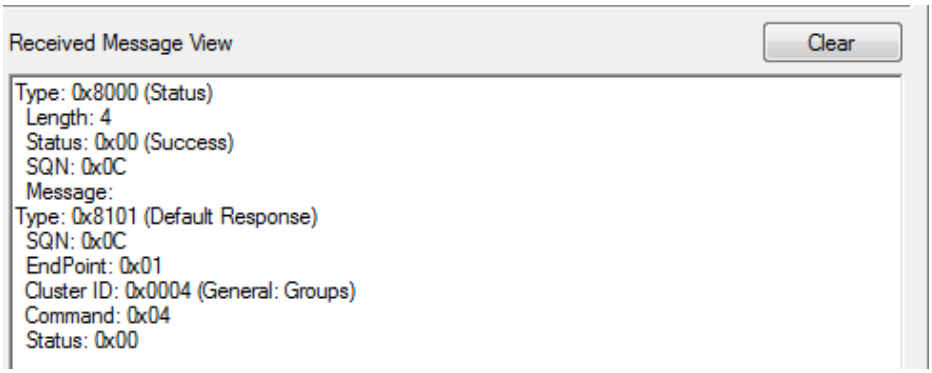
If the group does not exist on the device, it will respond with a status of "Not Found" (0x8B).



5.2.6.5 Remove All Groups

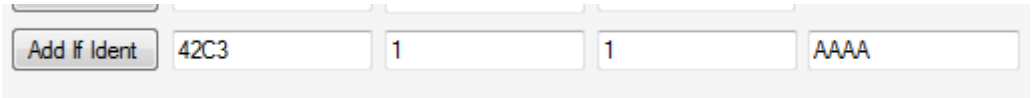
You can remove a device from all groups by sending the “Remove All Groups” command to the device. This is done in the **Remove All** line of the interface by entering the network address of the device, source endpoint number and destination endpoint number, and then clicking the **Remove All** button.

Irrespective of whether the device is associated with any groups, it will always respond with a status of “Success” (0x00).



5.2.6.6 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.

5.2.7 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.

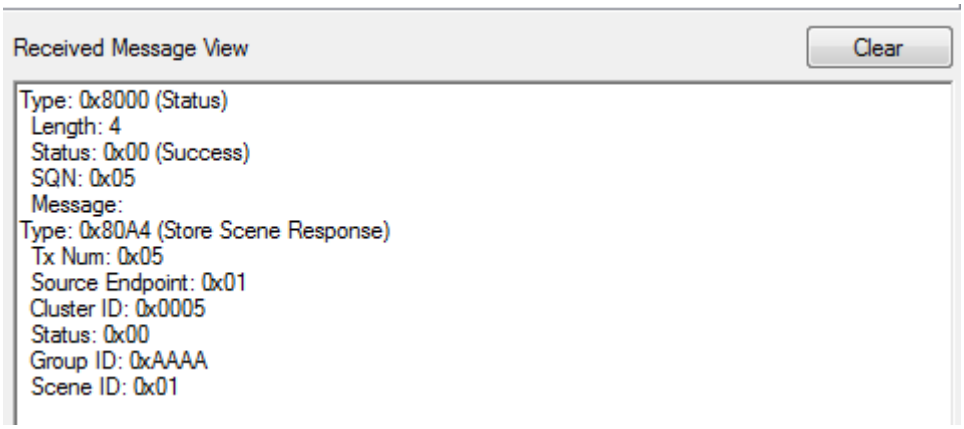
5.2.7.1 Add Scene

The "Add Scene" command allows a scene with specified Scene ID (associated with a particular Group ID) to be added on a remote device. This feature is included in the example code for the ZGWUI application but is not fully implemented in the interface. You can add a scene using the "Store Scene" command (see Store Scene).

5.2.7.2 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.

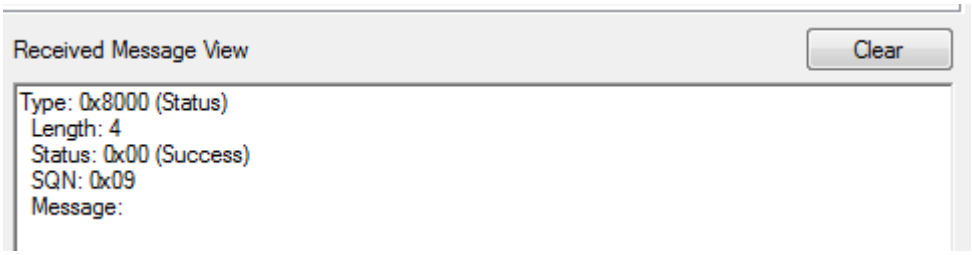


The above output indicates that the device state has been successfully stored in the scene with Scene ID 0x01 associated with the group with Group ID 0xAAAA

5.2.7.3 Recall Scene

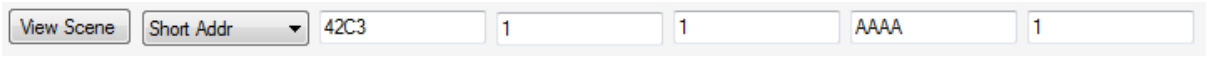
The “Recall Scene” command instructs a device to restore a previously saved scene in the device - for a light bulb, this could be restoring an on/off and level state. This is done in the **Recall Scn** line of the interface by entering the addressing mode, address of the device, source endpoint number,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

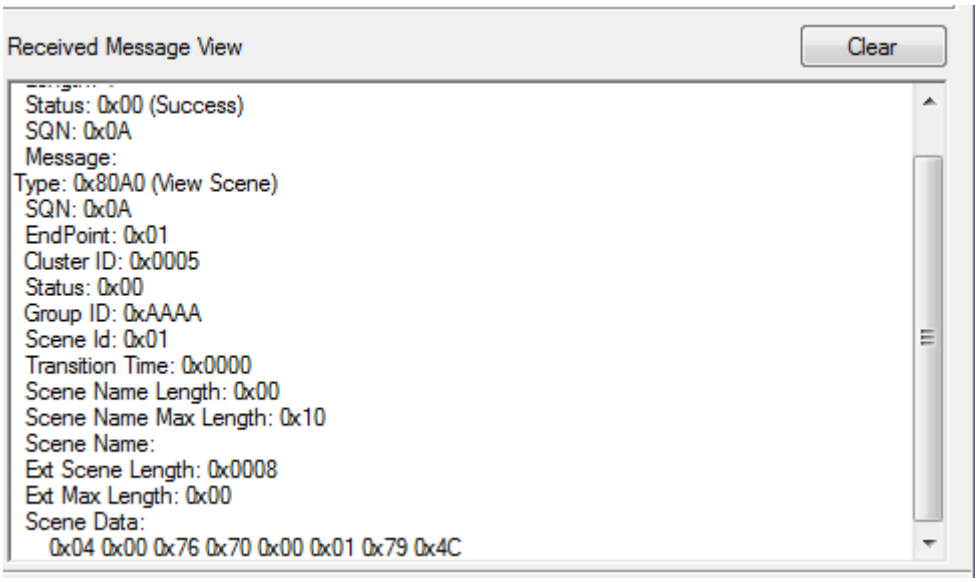


5.2.7.4 View Scene

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 scene information like Transition time, Scene Name Length, Scene Name and Scene Data will be displayed in the **Received Message View** pane.



5.2.7.5 Get Scene Membership

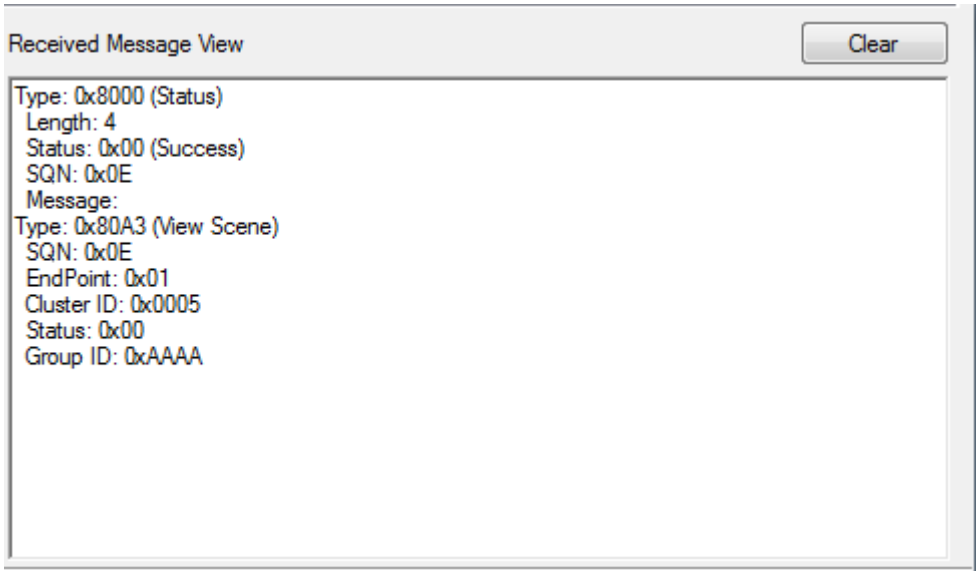
You can find out which scenes associated with a particular group are available on a device by sending a “Get Scene Membership” command to the device. This is done in the **Get Memb** line of the interface by entering the addressing mode, address of the device, source endpoint number, destination endpoint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

5.2.7.6 Remove All Scenes

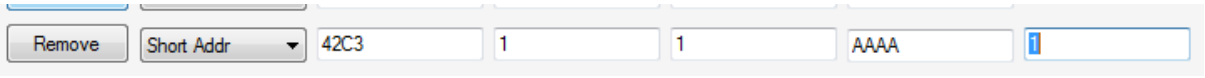
You can remove all scenes associated with a particular group on a device by sending a “Remove all Scenes” command to the device. This is done in the **Remove All** line of the interface by entering the addressing mode, address of the device, source endpoint number, destination endpoint number and Group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

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.



5.2.7.7 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.

5.2.8 Running Over-The-Air (OTA) Upgrade

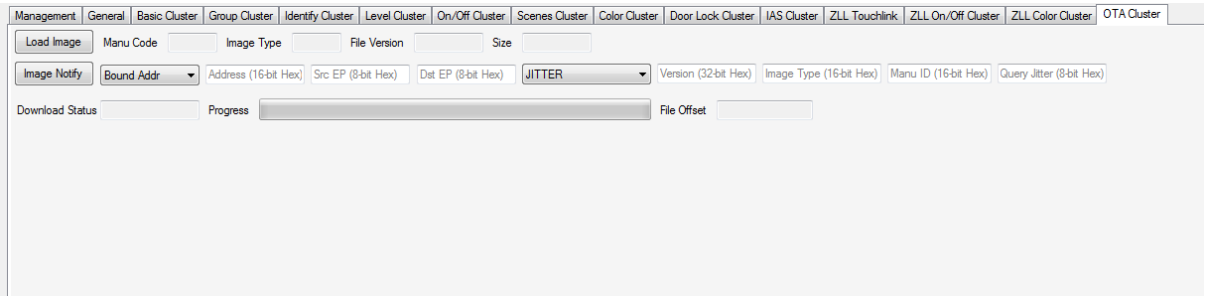
The ZGWUI provides an interface to perform an Over-The-Air (OTA) upgrade. This involves loading an application binary that will be served out 'over the air' to devices in the network. The following sections demonstrates how OTA upgrade is executed on the ZGWUI. This demonstration assumes that you have devices in the network which have the OTA Upgrade client cluster implemented. This document will describe the process of OTA upgrade on a Dimmable Light device. For this example, the following binary is initially used in the Dimmable Light:

DimmableLight_NtagIcode_GpCombo_Ota_OM15081_V1.bin

This application is supplied in the Application Note *Zigbee 3.0 Light Bulbs [JN-AN-1244]* and must be loaded into a network node (see Setting up the Nodes).

5.2.8.1 Loading the Upgrade Binary

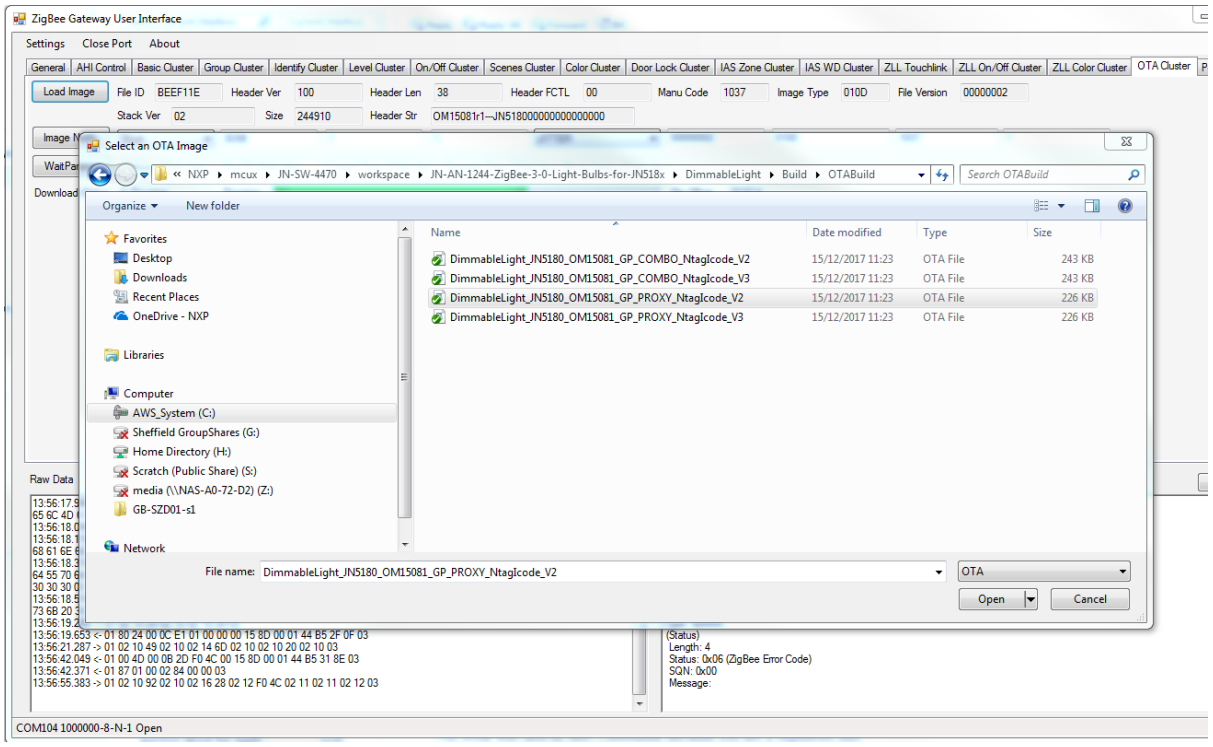
To perform an OTA upgrade, the relevant upgrade binary file needs to be loaded into the ZGWUI application. Click on the **OTA Cluster** tab, which is displayed as follows:



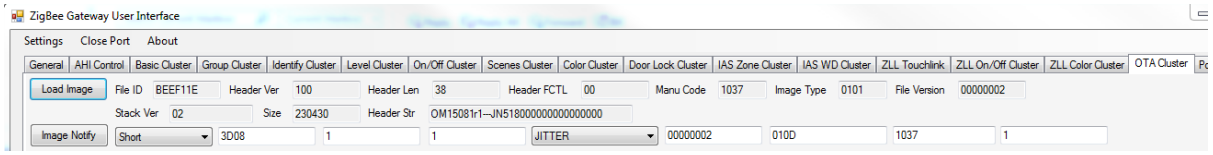
Click the **Load Image** button to bring up the file explorer window. Navigate to the folder which contains the OTA upgrade binary file that is to be used to upgrade the remote device and select the file – this is a **.ota** file, in this case:

DimmableLight_NtagIcode_GpCombo_Ota_OM15081_V2.ota

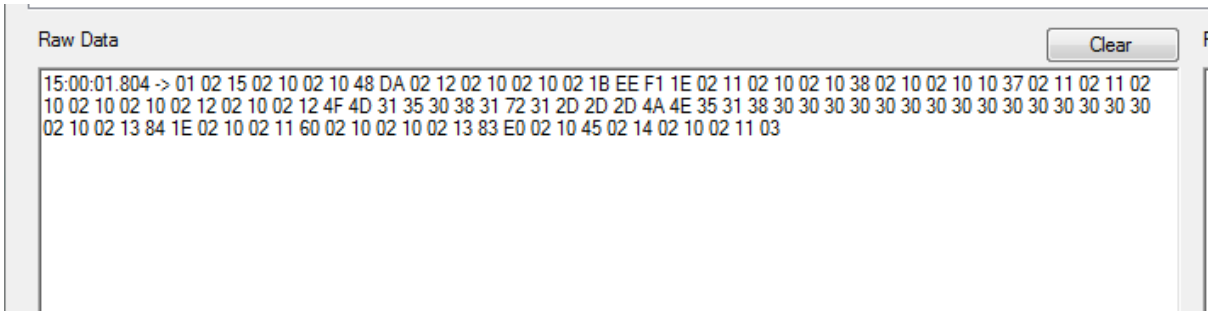
This file is supplied in the Application Note *Zigbee 3.0 Light Bulbs [JN-AN-1244]*, which must be present on your PC.



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.



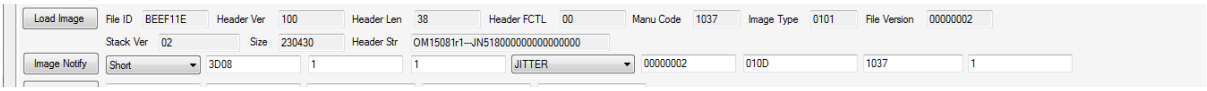
5.2.8.2 Image Notify

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 [JN-UG-3132]*.

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.



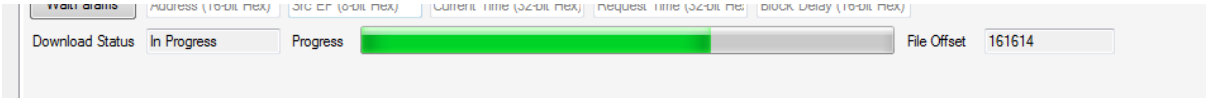
The above command notifies all relevant devices in the network and instructs all of them to upgrade straight away.

Alternatively the devices in the network will periodically query the Control Bridge to determine is a new compatible image is available for download.

5.2.8.3 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.

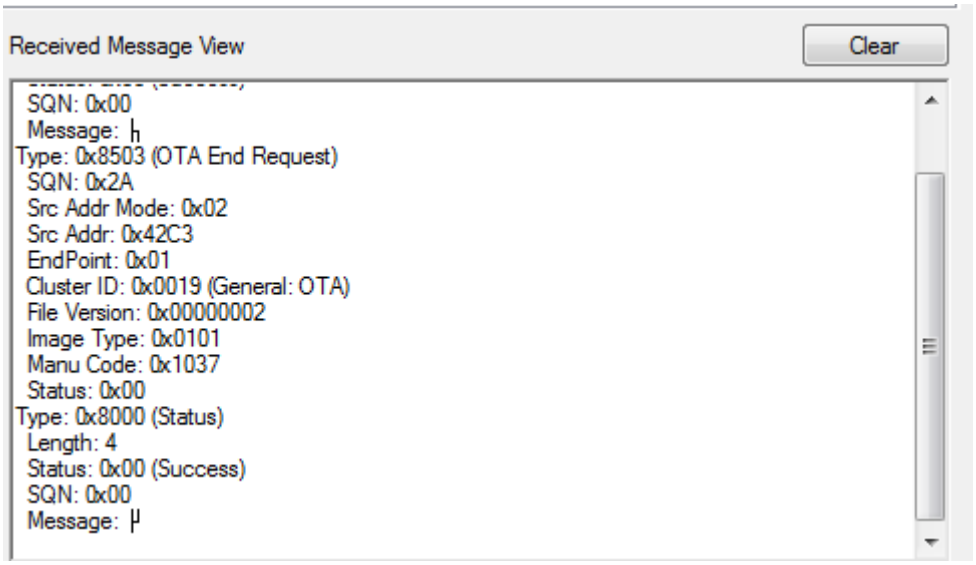
The progress bar in the ZGWUI, shown below, indicates the current status of the upgrading device. The File Offset value is the number of bytes the server has sent to the device so far.



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.

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.



6 ZGWUI Source

The ZGWUI is provided as both executable and source code. It is provided as source code to give the developer information on which data is sent to the Control Bridge and how it is sent. This should speed up application porting and reduce mistakes made during application development. Although it provides most of the functionality supported by the Control Bridge, the ZGWUI does not support all features. Custom features that are added to the Control Bridge by the developer will also need to be added to the ZGWUI for testing purposes.

The ZGWUI application is built using the Visual Studio 2012 IDE which is based on C# code.

7 Related Documents

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

- Zigbee 3.0 Getting Started Application Note [JN-AN-1260]
- DK6 Production Flash Programmer User Guide [JN-UG-3127]
- Zigbee 3.0 Stack User Guide [JN-UG-3130]
- Zigbee 3.0 Devices User Guide [JN-UG-3131]
- Zigbee 3.0 Cluster Library User Guide [JN-UG-3132]
- Zigbee 3.0 Green Power User Guide [JN-UG-3134]
- Encryption Tool User Guide [JN-UG-3135]

Appendix A: Serial Protocol

A.1. Physical Characteristics

The serial link between the ZGWUI (Zigbee Gateway User Interface) and wireless microcontroller runs at 1Mbaud in the pre-built binary files. 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 1: 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 Msg Type = 0x8000	<status:uint8_t> <sequence number: uint8_t> <Packet Type: uint16_t> <Optional additional error information: string> Status: 0 = Success 1 = Incorrect parameters 2 = Unhandled command 3 = Command failed 4 = Busy (Node is carrying out a lengthy operation and is currently unable to handle the incoming command) 5 = Stack already started (no new configuration accepted) 128 – 244 = Failed (Zigbee event codes) 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 Msg Type = 0x8001	<log level: uint8_t> <log message : string> Log Level : Use the Linux / Unix log levels 0 = Emergency 1 = Alert 2 = Critical 3 = Error 4 = Warning 5 = Notice 6 = Information 7 = Debug	
Node->Host	Data Indication Msg Type = 0x8002	<status: uint8_t> <Profile ID: uint16_t> <cluster ID: uint16_t> <source endpoint: uint8_t> <destination endpoint: uint8_t> <source address mode: uint8_t> <source address: uint16_t or uint64_t> <destination address mode: uint8_t> <destination address: uint16_t or uint64_t> <payload size : uint8_t> <payload : data each element is uint8_t>	
Node->Host	Node Cluster List – Sent by gateway node after reset Msg Type = 0x8003	<source endpoint: uint8_t t> <profile ID: uint16_t> <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 – 0 - STARTUP 2 - NFN_START 6 - RUNNING The node is not yet provisioned.	
Host->Node	Erase Persistent Data Msg Type = 0x0012	No payload	Status
Host->Node	NCI Command Set Msg Type=0x002D	<NCI Command: uint8_t> 0x00 – Disabled 0xA0 – Decommission (Factory Reset) 0xA1 - Commission	Status
Node->Host	NCI Command Notify Msg Type=0x802E	<NCI Command: uint8_t> 0x00 – Disabled 0xA0 – Decommission (Factory Reset) 0xA1 – Commission <Zigbee Device ID: uint16_t> <Extended Address: uint64_t>	Status
Host->Node	Bind Msg Type = 0x0030	<target extended address: uint64_t> <target endpoint: uint8_t> <cluster ID: uint16_t> <destination address mode: uint8_t> <destination address:uint16_t or uint64_t> <destination endpoint (value ignored for group address): uint8_t>	Status Bind response
Node->Host	Bind response Msg Type = 0x8030	<Sequence number: uint8_t> <status: uint8_t>	
Host->Node	Unbind Msg Type = 0x0031	<target extended address: uint64_t> <target endpoint: uint8_t> <cluster ID: uint16_t> <destination address mode: uint8_t> <destination address: uint16_t or uint64_t> <destination endpoint(value ignored for group address): uint8_t>	Status Unbind response
Node->Host	Unbind response Msg Type = 0x8031	<Sequence number: uint8_t> <status: uint8_t>	
Node->Host	Device Announce Msg Type = 0x004D	< short address: uint16_t> < IEEE address: uint64_t> < MAC capability: uint8_t> MAC capability Bit 0 - Alternate PAN Coordinator Bit 1 - Device Type Bit 2 - Power source Bit 3 - Receiver On when Idle Bit 4,5 - Reserved Bit 6 - Security capability Bit 7 - Allocate Address	
Host->Node	Network Address request Msg Type = 0x0040	<target short address: uint16_t> <extended address:uint64_t> <request type: uint8_t> <start index: uint8_t> Request Type: 0 = Single Request 1 = Extended Request	Status Network Address response
Node->Host	Network Address response Msg Type = 0x8040	<Sequence number: uin8_t> <status: uint8_t> <IEEE address: uint64_t> <short address: uint16_t> <number of associated devices: uint8_t> <start index: uint8_t> <device list – data each entry is uint16_t>	

Host->Node	IEEE Address request Msg Type = 0x0041	<target short address: uint16_t> <short address: uint16_t> <request type: uint8_t> <start index: uint8_t> Request Type: 0 = Single 1 = Extended	Status IEEE Address response
Node->Host	IEEE Address response Msg Type = 0x8041	<Sequence number: uin8_t> <status: uint8_t> <IEEE address: uint64_t> <short address: uint16_t> <number of associated devices: uint8_t> <start index: uint8_t> <device list – data each entry is uint16_t>	
Host->Node	Node Descriptor request Msg Type = 0x0042	<target short address: uint16_t>	Status Node Descriptor response
Node->Host	Node Descriptor response Msg Type = 0x8042	<Sequence number: uint8_t> <Status uint8_t> <network address: uint16_t> <manufacturer code: uint16_t> <max Rx Size: uint16_t> <max Tx Size: uint16_t> <server mask: uint16_t> <descriptor capability: uint8_t> <mac flags: uint8_t> <max buffer size: uint8_t > <bit fields: uint16_t> Bitfields: Logical type (bits 0-2 0 - Coordinator 1 - Router 2 - End Device) Complex descriptor available (bit 3) User descriptor available (bit 4) Reserved (bit 5-7) APS flags (bit 8-10 – currently 0) Frequency band(11-15 set to 3 (2.4Ghz)) Server mask bits: 0 - Primary trust center 1 - Back up trust center 2 - Primary binding cache 3 - Backup binding cache 4 - Primary discovery cache 5 - Backup discovery cache 6 - Network manager 7 to15 - Reserved MAC capability Bit 0 - Alternate PAN Coordinator Bit 1 - Device Type Bit 2 - Power source Bit 3 - Receiver On when Idle Bit 4-5 - Reserved Bit 6 - Security capability Bit 7 - Allocate Address Descriptor capability: 0 - extended Active endpoint list available 1 - Extended simple descriptor list available 2 to 7 - Reserved	

Host->Node	Simple Descriptor request Msg Type = 0x0043	<target short address: uint16_t> <endpoint: uint8_t>	Status 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 Descritpor 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 Msg Type = 0x804A	<Sequence number: uint8_t> <status: uint8_t> <total transmission: uint16_t> <transmission failures: uint16_t> <scanned channels: uint32_t > <scanned channel list count: uint8_t> <channel list: list each element is uint8_t>	
Host->Node	System Server Discovery request Msg Type = 0x004B	<target short address: uint16_t> <Server mask: uint16_t> Bitmask according to spec.	Status System Server Discovery response
Node->Host	System Server Discovery response Msg Type = 0x804B	<Sequence number: uint8_t> <status: uint8_t> <Server mask: uint16_t> Bitmask according to spec.	
Host->Node	Management LQI request Msg Type = 0x004E	<Target Address : uint16_t> <Start Index : uint8_t>	Status 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 bit 0-1 Device Type (0-Coordinator 1-Router 2-End Device) bit 2-3 Permit Join status (1- On 0-Off) bit 4-5 Relationship (0-Parent 1-Child 2-Sibling) 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> 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> Direction: 0 - from server to client 1 - from client to server	Data Indication Msg Type = 0x8002

		Manufacturer specific : 1 – Yes 0 – No	
Host->Node	Attribute Discovery request Msg Type = 0x0140	<address mode: uint8_t> <target short address: uint16_t> <source endpoint: uint8_t> <destination endpoint: uint8_t> <Cluster id: uint16_t> <Attribute id : uint16_t> <direction: uint8_t> <manufacturer specific: uint8_t> <manufacturer id: uint16_t> <Max number of identifiers: uint8_t> Direction: 0 - from server to client 1 - from client to server Manufacturer specific : 1 – Yes 0 – No	Status Attribute Discovery response
Node->Host	Attribute Discovery response Msg Type = 0x8140	<complete: uint8_t> <attribute type: uint8_t> <attribute id: uint16_t> Complete: 0 – more attributes to follow 1 – this was the last attribute	
Host->Node	Enable Permissions Controlled Joins Msg Type = 0x0027	<Enable/Disable : uint8_t> 1 – Enable 2 – Disable	Status
Host->Node	Authenticate Device Msg Type = 0x0028	<IEEE address ; uint64_t> <Key : 16 elements byte each>	Status Authenticate response
Node->Host	Authenticate response Msg Type = 0x8028	<IEEE address of the Gateway: uint64_t> <Encrypted Key : uint8_t 16 elements> <MIC : uint8 4 elements> <IEEE address of the initiating node : uint64_t> <Active Key Sequence number : uint8_t> <Channel : uint8_t> <Short PAN Id : uint16_t> <Extended PAN ID : uint64_t>	
Host->Node	Configure Reporting request Msg Type = 0x0120	<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> Attribute direction : uint8_t Attribute type : uint8_t Attribute id : uint16_t Min interval : uint16_t Max interval : uint16_t Timeout : uint16_t Change : uint8_t	Status Configure Reporting response
Node->Host	Configure Reporting response Msg Type = 0x8120	<Sequence number: uint8_t> <Src address : uint16_t> <Endpoint: uint8_t> <Cluster id: uint16_t> <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>	
Host->Node	Out of Band Commissioning Data Request Msg Type = 0x0029	<Address of interest : uint64_t> <Key : 16 elements byte each>	Status Out of Band Commissioning Data Response
Node->Host	Out of Band Commissioning Data Response Msg Type = 0x8029	<Device Extended Address: uint64_t> <Key : 16 elements byte each> <MIC : uint32_t> <Host Extended Address : uint64_t> <Active Key Sequence Number : uint8_t> <Channel : uint8_t> <PAN ID : uint16_t> <Extended PAN ID : uint64_t> <Short Address : uint16_t> <Device ID : uint16_t> <Status: uint8_t>	
	Install Code Data Request Msg Type = 0x002F	<Address of interest : uint64_t> <Key : 16 elements byte each>	Status Install Code Data Response
	Instal Code Data Response Msg Type = 0x802F	<Status: uint8_t> <Device Extended Address: uint64_t> <Link Key Sequence Number : 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 Command ID = 0x02	<address mode: uint8_t> <target short address: uint16_t> <source endpoint: uint8_t> <destination endpoint: uint8_t> <group count: uint8_t> <group list:data>	Status Get Group Membership response
Node->Host	Get Group Membership response Msg Type = 0x8062 Command ID = 0x02	<Sequence number: uint8_t> <endpoint: uint8_t> <Cluster id: uint16_t> <capacity: uint8_t> <Group count: uint8_t> <List of Group id: list each data item uint16_t>	
Host->Node	Remove Group Msg Type = 0x0063 Command ID = 0x03	<address mode: uint8_t> <target short address: uint16_t> <source endpoint: uint8_t> <destination endpoint: uint8_t> <group address: uint16_t >	Status Remove Group response
Node->Host	Remove Group response Msg Type = 0x8063 Command ID = 0x03	<Sequence number: uin8_t> <endpoint: uint8_t> <Cluster id: uint16_t> <status: uint8_t> <Group id: uint16_t>	Status
Host->Node	Remove All Groups Msg Type = 0x0064 Command ID = 0x04	<address mode: uint8_t> <target short address: uint16_t> <source endpoint: uint8_t> <destination endpoint: uint8_t>	Status
Host->Node	Add Group if identify Msg Type = 0x0065 Command ID = 0x05	<address mode: uint8_t> <target short address: uint16_t> <source endpoint: uint8_t> <destination endpoint: uint8_t> <group address: uint16_t >	Status

B.1.4. Identify Cluster Commands

Message Direction	Message Description	Message Format	Expected Response
Host->Node	Identify Send Msg Type = 0x0070	<address mode: uint8_t> <target short address: uint16_t> <source endpoint: uint8_t>	Status

		<destination endpoint: uint8_t> <time: uint16_t> Time: Seconds	
Host->Node	Identify Query Msg Type = 0x0071	<address mode: uint8_t> <target short address: uint16_t> <source endpoint: uint8_t> <destination endpoint: uint8_t>	Status

B.1.5. Level Cluster Commands

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

		<destination endpoint: uint8_t> <command ID: uint8_t> Command Id 0 - Off 1 - On 2 - Toggle	
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 Msg Type = 0x00A0	<address mode: uint8_t> <target short address: uint16_t> <source endpoint: uint8_t> <destination endpoint: uint8_t> <group ID: uint16_t> <scene ID: uint8_t>	Status View Scene response
Node->Host	View Scene response Msg Type = 0x80A0	<sequence number: uint8_t> <endpoint : uint8_t> <cluster id: uint16_t> <status: uint8_t> <group ID: uint16_t> <scene ID: uint8_t> <transition time: uint16_t> <scene name length: uint8_t> <scene name max length: uint8_t> <scene name data: data each element is uint8_t> <extensions length: uint16_t> <extensions max length: uint16_t> <extensions data: data each element is uint8_t>	
Host->Node	Add Scene Msg Type = 0x00A1	<address mode: uint8_t> <target short address: uint16_t> <source endpoint: uint8_t> <destination endpoint: uint8_t> <group ID: uint16_t> <scene ID: uint8_t> <transition time: uint16_t> <scene name length: uint8_t> <scene name max length: uint8_t> <scene name data: data each element is uint8_t>	Status Add Scene response
Node->Host	Add Scene response Msg Type = 0x80A1	<sequence number: uint8_t> <endpoint : uint8_t> <cluster id: uint16_t> <status: uint8_t> <group ID: uint16_t> <scene ID: uint8_t>	
Host->Node	Remove Scene Msg Type = 0x00A2	<address mode: uint8_t> <target short address: uint16_t> <source endpoint: uint8_t> <destination endpoint: uint8_t> <group ID: uint16_t> <scene ID: uint8_t>	Status Remove Scene response
Node->Host	Remove Scene response Msg Type = 0x80A2	<sequence number: uint8_t> <endpoint : uint8_t> <cluster id: uint16_t> <status: uint8_t> <group ID: uint16_t> <scene ID: uint8_t>	
Host->Node	Remove all scenes Msg Type = 0x00A3	<address mode: uint8_t> <target short address: uint16_t> <source endpoint: uint8_t> <destination endpoint: uint8_t> <group ID: uint16_t>	Status Data indication
Node->Host	Remove All Scene response Msg Type = 0x80A3	<sequence number: uint8_t> <endpoint : uint8_t> <cluster id: uint16_t> <status: uint8_t> <group ID: uint16_t>	

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

B.1.8. Colour Cluster Commands

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

Host->Node	Move Colour Msg Type = 0x00B8	<address mode: uint8_t> <target short address: uint16_t> <source endpoint: uint8_t> <destination endpoint: uint8_t> <colour X: int16_t> <colour Y: int16_t>	Status Data indication
Host->Node	Step Colour Msg Type = 0x00B9	<address mode: uint8_t> <target short address: uint16_t> <source endpoint: uint8_t> <destination endpoint: uint8_t> <step X: int16_t> <step Y: int16_t> <transition time: uint16_t >	Status 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 Msg Type = 0x00D0	No Payload	Status
Host->Node	Touch link factory reset target Msg Type= 0x00D2	No Payload	Status
Node->Host	Touchlink Status Msg Type = 0x00D1	<status: uint8_t> <joined node short address: uint16_t> Status 0 = Success 1 = Failure	

B.2.2. Identify Cluster Commands

Message Direction	Message Description	Message Format	Expected Response
Host->Node	Identify Trigger Effect Msg Type = 0x00E0	<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 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> <target short address: uint16_t> <source endpoint: uint8_t> <destination endpoint: uint8_t> <effect ID: uint8_t> <effect gradient: uint8_t>	Status Data indication
Host->Node	On / Off Timed 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: uint8_t> <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> <target short address: uint16_t> <source endpoint: uint8_t> <destination endpoint: uint8_t> <group ID: uint16_t> <scene ID: uint8_t> <transition time: uint16_t> <scene name: string> <length: uint16_t> <max length: uint16_t> <data: data>	Status Data indication
Host->Node	View Enhanced Host->Node Scene Msg Type = 0x00A8	<address mode: uint8_t> <target short address: uint16_t> <source endpoint: uint8_t> <destination endpoint: uint8_t> <group ID: uint16_t> <scene ID: uint8_t>	Status Data indication
Host->Node	Copy Scene Msg Type = 0x00A9	<address mode: uint8_t> <target short address: uint16_t> <source endpoint: uint8_t> <destination endpoint: uint8_t> <mode: uint8_t> <from group ID: uint16_t> <from scene ID: uint8_t> <to group ID: uint16_t> <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 Msg Type = 0x00BA	<address mode: uint8_t> <target short address: uint16_t> <source endpoint: uint8_t> <destination endpoint: uint8_t> <direction: uint8_t> <Enhanced Hue: uint16_t> <transition time: uint16_t>	Status Data indication
Host->Node	Enhanced Move Hue Msg Type = 0x00BB	<address mode: uint8_t> <target short address: uint16_t> <source endpoint: uint8_t> <destination endpoint: uint8_t> <mode: uint8_t> <rate: uint8_t>	Status Data indication
Host->Node	Enhanced Step Hue Msg Type = 0x00BC	<address mode: uint8_t> <target short address: uint16_t> <source endpoint: uint8_t> <destination endpoint: uint8_t> <mode: uint8_t> <step size: uint8_t> <transition time: uint8_t>	Status Data indication
Host->Node	Enhanced Move to hue and saturation Msg Type = 0x00BD	<address mode: uint8_t> <target short address: uint16_t> <source endpoint: uint8_t> <destination endpoint: uint8_t> <enhanced hue: uint32_t> <saturation: uint32_t> <transition time: uint8_t>	Status Data indication
Host->Node	Colour Loop Set Msg Type = 0x00BE	<address mode: uint8_t> <target short address: uint16_t> <source endpoint: uint8_t> <destination endpoint: uint8_t> <update flags: uint8_t> <action: uint8_t> <direction: uint8_t> <time: uint8_t> <start hue: uint32_t>	Status Data indication
Host->Node	Stop Move Step Msg Type = 0x00BF	<address mode: uint8_t> <target short address: uint16_t> <source endpoint: uint8_t> <destination endpoint: uint8_t>	Status Data indication
Host->Node	Move to colour temperature Msg Type = 0x00C0	<address mode: uint8_t> <target short address: uint16_t> <source endpoint: uint8_t> <destination endpoint: uint8_t> <colour temperature: uint16_t> <transition time: uint16_t>	Status Data indication
Host->Node	Move colour temperature Msg Type = 0x00C1	<address mode: uint8_t> <target short address: uint16_t> <source endpoint: uint8_t> <destination endpoint: uint8_t> <mode: uint8_t> <rate: uint16_t> <minimum temperature: uint16_t> <maximum temperature: uint16_t> <options mask: uint8_t> <options override: uint8_t>	Status Data indication

Host->Node	Step colour temperature Msg Type = 0x00C2	<address mode: uint8_t> <target short address: uint16_t> <source endpoint: uint8_t> <destination endpoint: uint8_t> <mode: uint8_t> <step size: uint16_t> <transition time: uint16_t> <minimum temperature: uint16_t> <maximum temperature: uint16_t>	Status 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 Msg Type = 0x00F0	<address mode: uint8_t> <target short address: uint16_t> <source endpoint: uint8_t> <destination endpoint: uint8_t> <lock/unlock: uint8_t> 0 = Lock 1 = Unlock	Status Data indication

B.3.2 IAS Cluster Commands

Message Direction	Message Description	Message Format	Expected Response
Host->Node	IAS Zone enroll response Msg Type = 0x0400	<address mode: uint8_t> <target short address: uint16_t> <source endpoint: uint8_t> <destination endpoint: uint8_t> <Enroll response code: uint8_t> <Zone id : uint8_t>	Status
Node->Host	Zone status change notification Msg Type = 0x8401	<sequence number: uint8_t> <endpoint : uint8_t> <cluster id: uint16_t> <src address mode: uint8_t> <src address: uint64_t or uint16_t based on address mode> <zone status: uint16_t> <extended status: uint8_t> <zone id : uint8_t> <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 4Kbytes on a JN5169 device and can restrict network size. To overcome this it is possible to export the data persistence to the host device. 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> <Record Id: uint16_t> <total size: uint32_t> <total number of blocks: uint32_t> <current block: uint32_t> <block size: uint32_t> <data: variable list each item is uint8_t> status: 0- no record found 1- Record recovered	Status
Node->Host	Save Record request Msg Type = 0x0200	<Record Id: uint16_t> <total size: uint32_t> <total number of blocks: uint32_t> <current block: uint32_t> <block size: uint32_t> <data: variable list, each item is uint8_t>	Save Record response
Host->Node	Save Record response Msg Type = 0x8200	<Record Id: uint16_t> <total size: uint32_t> <total number of blocks: uint32_t> <current block: uint32_t> <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 Msg Type = 0x0530	<address mode: uint8_t> <target short address: uint16_t> <source endpoint: uint8_t> <destination endpoint: uint8_t> <profile ID: uint16_t> <cluster ID: uint16_t> <security mode: uint8_t> <radius: uint8_t> <data length: uint8_t> <data: auint8_t>	Status
Node->Host	Router Discovery Confirm Msg Type = 0x8701	<status: uint8_t> <nwk status: uint8_t>	
Node->Host	APS Data Confirm Fail Msg Type = 0x8702	<status: uint8_t> <src endpoint: uint8_t> <dst endpoint: uint8_t> <dst address mode: uint8_t> <destination address: uint64_t> <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. On/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)

Important Notice

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer’s applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EMBRACE, GREENCHIP, HITAG, ICODE, JCOP, LIFE VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, AltiVec, CodeWarrior, ColdFire, ColdFire+, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, Tower, TurboLink, EdgeScale, EdgeLock, eIQ, and Immersive3D are trademarks of NXP B.V.

All other product or service names are the property of their respective owners. AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, µVision, Versatile are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

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

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 17-Apr-2020

Document identifier: JN-AN-1247