



# Application Note: JN-AN-1223

## ZigBee IoT Gateway Control Bridge

The NXP ZigBee IoT Gateway Control Bridge provides a means of controlling ZigBee devices via a serial link which is connected to a host controller. The IoT Gateway Control Bridge supports ZigBee Home Automation and ZigBee Light Link, controlling the network by mostly client cluster commands.

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

The purpose of this Application Note is to show how a ZigBee Gateway can be controlled by an application running on a PC. It is also to demonstrate the different commands that can be sent in the payload that the ZigBee Control Bridge requires. For information on how to use the ZigBee IoT Gateway Control Bridge with the components in the JN516x-EK004 Evaluation Kit, please refer to the *JN516x-EK004 Evaluation Kit User Guide (JN-UG-3108)*.

This guide is intended to show how to set up and use the Gateway in a simple demonstration network of ZigBee Home Automation (ZHA) devices in order to familiarise users with the functions available in the Gateway firmware. 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 JN516x Gateway. The demonstration described in this guide uses the hardware found in the JN516x-EK004 Evaluation Kit. The firmware used in the Gateway is supplied as source code to allow the user to make changes and customise the various components to their needs. Firmware for the devices to be controlled by the Gateway can be built from the Application Notes *ZigBee Home Automation Demonstration (JN-AN-1189)* and the *ZigBee Light Link Solution (JN-AN-1171)*.

## 2 Capabilities

Product Type	Part Number	Build
Evaluation Kit	JN516x-EK004	-
JN516x ZLL/HA SDK	JN-SW-4168	1595
'BeyondStudio for NXP' Toolchain	JN-SW-4141	1308

The main purpose of this Application Note is to provide a JN516x 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 JN516x Control Bridge supports. It is also provided as source code, so developers can reference the protocol data sent to the JN516x Control Bridge to aid faster development.

The ZigBee Control Bridge application has been developed to work with the JN516x-EK004 Evaluation Kit.

## 3 What is Provided

The demonstration package comes with the following components, intended to be used with the hardware in the JN516x-EK004 Evaluation Kit:

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

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:

- ZLL on/off/dimmable colour light (**Light\_ExtendedColorLight\_JN5168\_DR1175.bin**)
- ZHA on/off/dimmable monochrome light (**DimmableLight\_JN5168\_DR1175.bin**)

These binaries are provided in the Application Notes *ZigBee Home Automation Demonstration (JN-AN-1189)* and *ZigBee Light Link Solution (JN-AN-1171)*, and must be loaded into boards of the JN516x-EK004 Evaluation Kit (see Section 5.2.3).

## 4 Configuring the Evaluation Kit Hardware

The Control Bridge is compatible with the Raspberry Pi which is supplied in the JN516x-EK004 Evaluation Kit. Set-up and configuration is described in the *JN516x-EK004 Evaluation Kit User Guide (JN-UG-3108)* which is contained in the Application Note *ZigBee IoT Gateway Host with NFC (JN-AN-1222)*. This Application Note is independent of the Raspberry Pi, which will not be described in this document.

## 5 Running the Demonstration

### 5.1 Programming the JN516x Device

Application Binary	Expansion Board ( + Carrier Board )			Remote Control Unit	USB Dongle
	Generic	LCD	Lighting/Sensor		
ZigbeeControlBridge_JN5168.bin	■				■
ZigbeeControlBridge_JN5169.bin	■				■

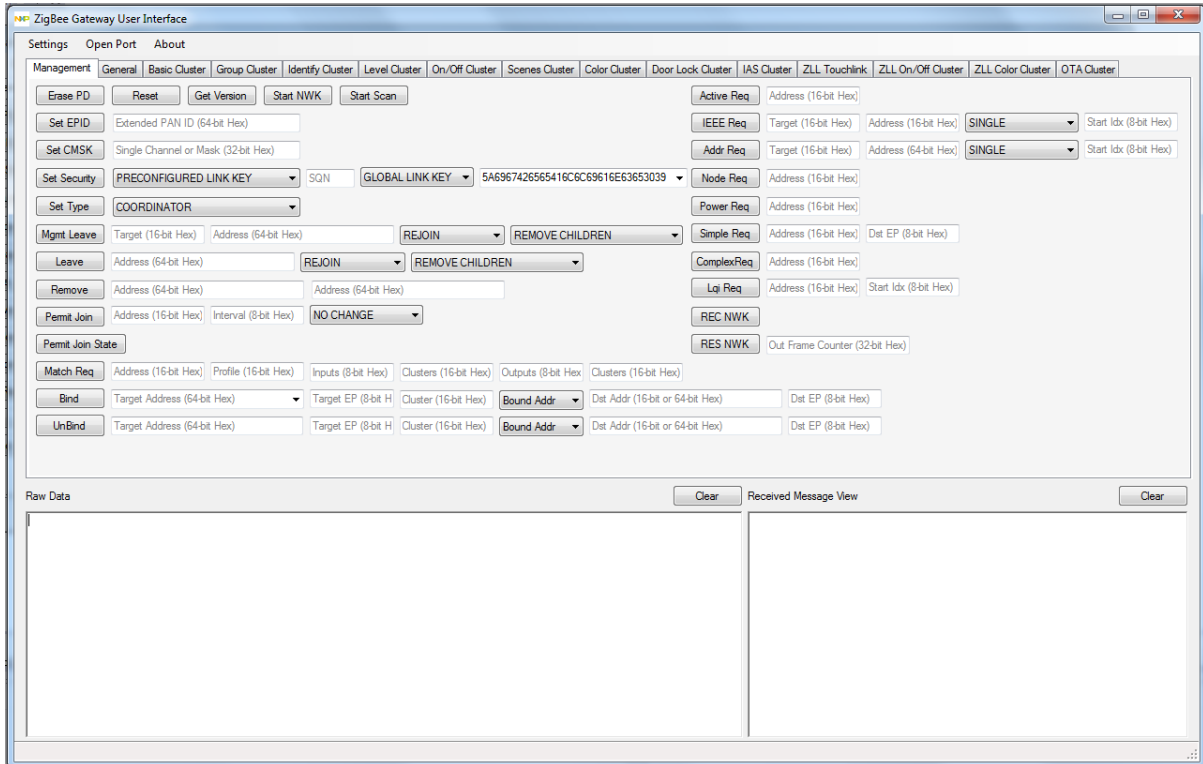
To run the demonstration, the ZigBee Control Bridge binary will need to be programmed into a valid evaluation kit board or USB dongle. This can be done from the ‘BeyondStudio for NXP’ development platform for JN516x devices. For instructions on using BeyondStudio to program an application into a JN516x device, please refer to the *BeyondStudio for NXP Installation and User Guide (JN-UG-3098)*.

By default, the firmware uses the JN516x UART0 to communicate with the host. Debug can also be enabled on UART1, but this can only be used when a DR1174 Carrier Board fitted with a DR1199 Generic Expansion Board is deployed. Debug can be implemented by connecting a serial cable from the PC to the Generic Expansion Board and opening a

terminal with baud rate 115200 on the PC. This cannot be done on a USB Dongle as there is no UART1 connection available.

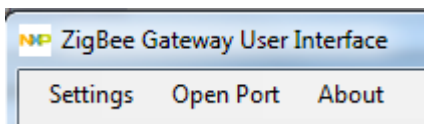
### 5.2 Running the ZGWUI

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**.

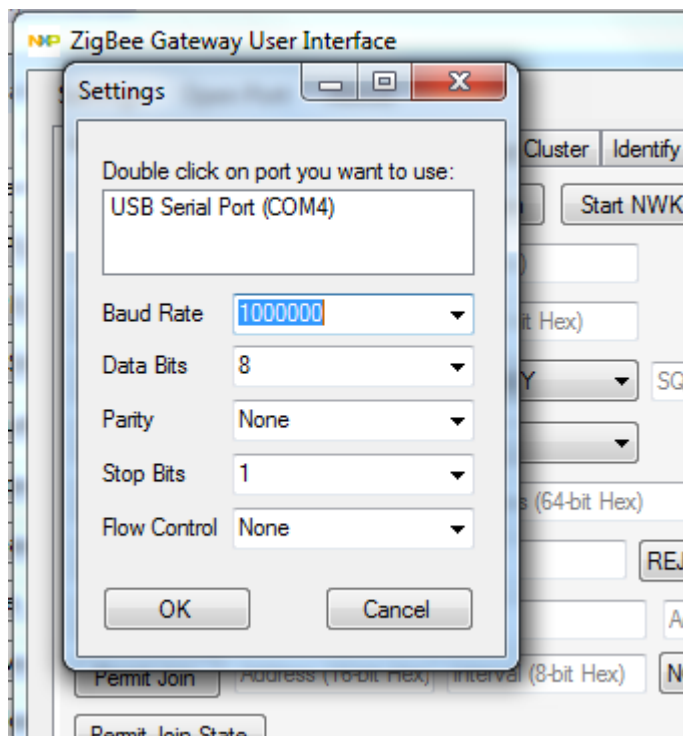


#### 5.2.1 Connecting to the Control Bridge

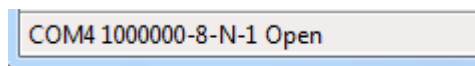
In order to connect to the Control Bridge and issue commands to communicate with ZigBee devices, a serial connection must be set up and opened. To do this, click on **Settings** towards the top-left of the interface.



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.

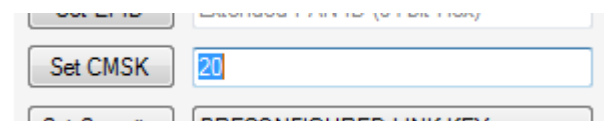


## 5.2.2 Configuring and Starting a Network

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.

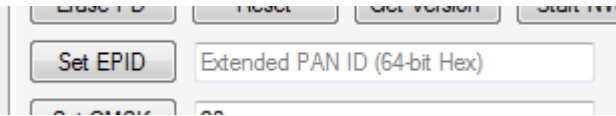
A classical network can be created for both ZigBee Home Automation and ZigBee Light Link. In this case, the Control Bridge starts as a Coordinator and allows devices into the network via MAC association. Before you start the network, there are basic commands that can be optionally issued to create a customised network.

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

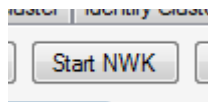


**Indicates the network is to start on channel 20**

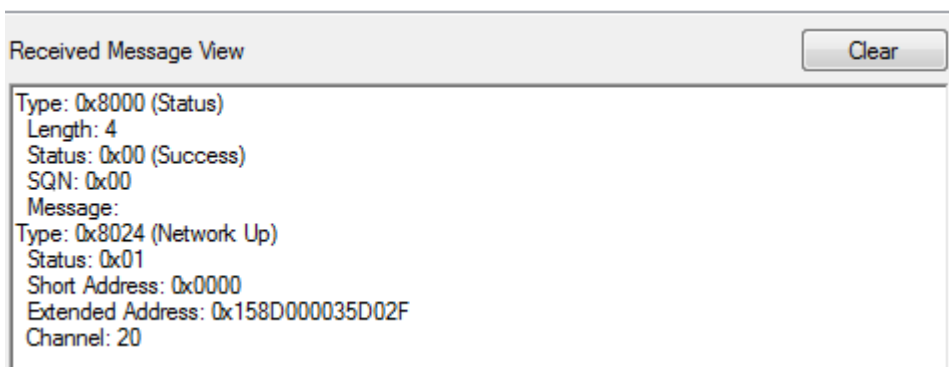
The **Set EPID** textbox can be used to enter a pre-defined Extended PAN ID (EPID) as a 64-bit hexadecimal value. The “Set Extended PAN ID” command can then be issued by clicking the **Set EPID** button.



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



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



### 5.2.3 Setting up the Nodes

The demonstration requires the DR1174 Carrier Boards (supplied with the JN516x-EK004 Evaluation Kit) to be configured as lights which can be controlled. Each Carrier Board therefore needs to be fitted with a DR1175 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 *JN516x-EK004 Evaluation Kit User Guide (JN-UG-3108)* 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

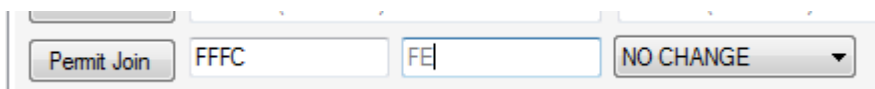
Depending on which type of device and ZigBee network configuration you are demonstrating, you will need to program each light board with the appropriate application binary – one of:

- ZigBee Home Automation monochrome dimmable light  
(**DimmableLight\_JN516x\_DR1175.bin**)
- ZigBee Light Link extended colour light  
(**Light\_ExtendedColorLight\_JN516x\_DR1175.bin**)

These binaries are supplied in the Application Notes *ZigBee Home Automation Demonstration (JN-AN-1189)* and *ZigBee Light Link Solution (JN-AN-1171)*. They must be programmed into the devices using a JN51xx Flash programming tool, such as the one provided within BeyondStudio for NXP and described in the *BeyondStudio for NXP Installation and User Guide (JN-UG-3098)*.

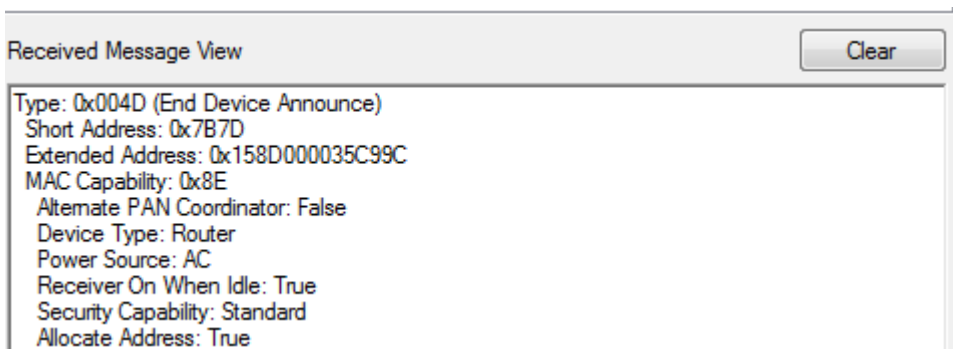
## 5.2.4 Joining Nodes to the Network

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



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

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



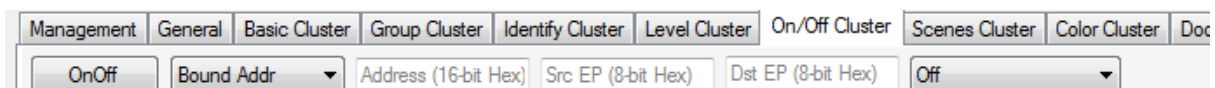
## 5.2.5 Controlling Devices

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.

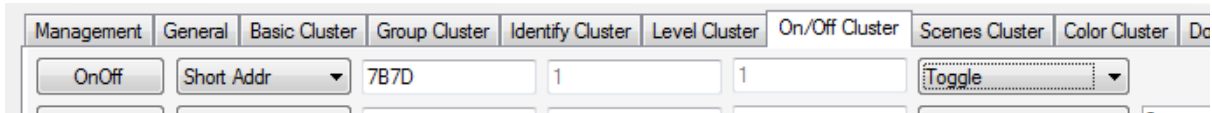
### 5.2.5.1 On/Off Cluster

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

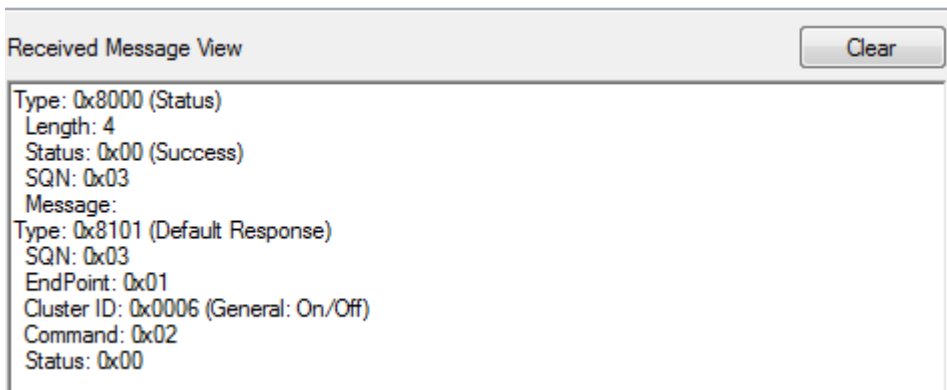
Click on the **On/Off Cluster** tab along the top of the interface.



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, the source endpoint number and the destination endpoint number (all in hexadecimal). Finally, select the type of "On/Off" command that you want to send.

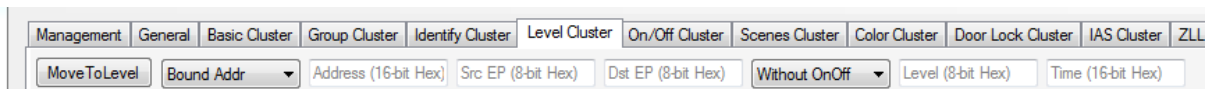


The light will change its on/off state and a Default Response message will be received in the **Received Message View** pane. The Default Response confirms that a device received the “On/Off” command and processed the command. If the command was not sent via unicast, a Default Response will not be received.



### 5.2.5.2 Level Control Cluster

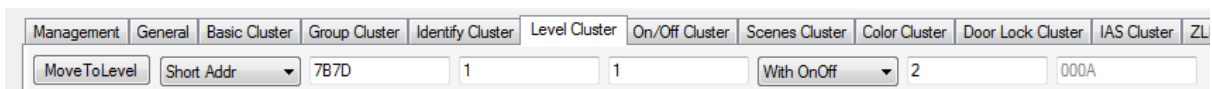
The Level Control cluster allows a bulb’s dimmable light level to be set to a specific value. This value can be between 0 and 254 (inclusive), and can be set on the **Level Cluster** tab.



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.

Received Message View
Clear

```

Type: 0x8000 (Status)
Length: 4
Status: 0x00 (Success)
SQN: 0x09
Message:
Type: 0x8101 (Default Response)
SQN: 0x09
EndPoint: 0x01
Cluster ID: 0x0008 (General: Level Control)
Command: 0x04
Status: 0x00
            
```

### 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.

Management	General	Basic Cluster	Group Cluster	Identify Cluster	Level Cluster	On/Off Cluster	Sc
Add Group	Address (16-bit Hex)	Src EP (8-bit Hex)	Dst EP (8-bit Hex)	Group ID (16-bit Hex)			
View Group	Address (16-bit Hex)	Src EP (8-bit Hex)	Dst EP (8-bit Hex)	Group ID (16-bit Hex)			
Get Group	Address (16-bit Hex)	Src EP (8-bit Hex)	Dst EP (8-bit Hex)	Group Count			
Remove Grp	Address (16-bit Hex)	Src EP (8-bit Hex)	Dst EP (8-bit Hex)	Group ID (16-bit Hex)			
Remove All	Address (16-bit Hex)	Src EP (8-bit Hex)	Dst EP (8-bit Hex)				
Add If Ident	Address (16-bit Hex)	Src EP (8-bit Hex)	Dst EP (8-bit Hex)	Group ID (16-bit Hex)			

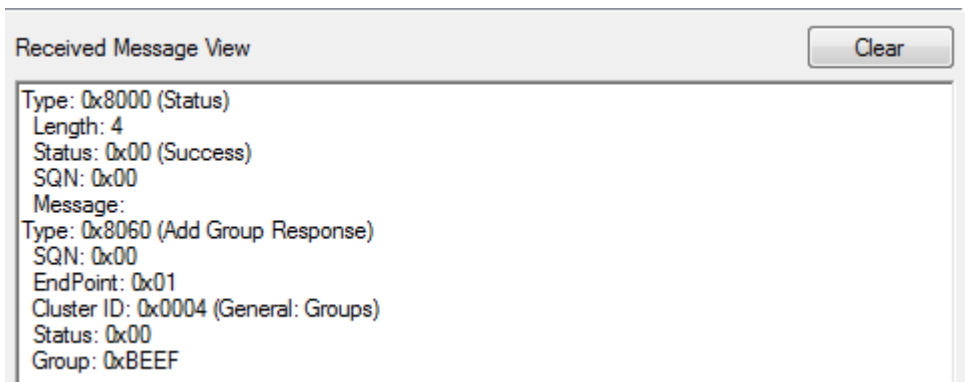
#### 5.2.6.1 Add Group

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

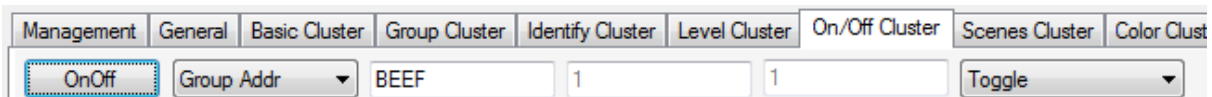
Management	General	Basic Cluster	Group Cluster	Identify Cluster	Level Cluster	On/Off Cluster	Sc
Add Group	<input type="text" value="706A"/>	<input type="text" value="1"/>	<input type="text" value="1"/>	<input type="text" value="BEEF"/>			



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

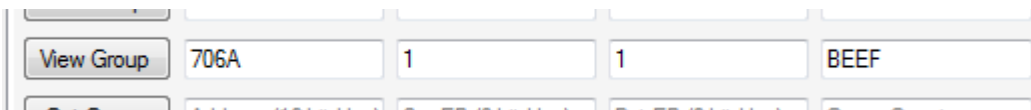


To verify that this group has been added, try sending an “On/Off” command with the group address you have just added. This will toggle the on/off state of the light. Note that since this is a groupcast, a Default Response will not be received.

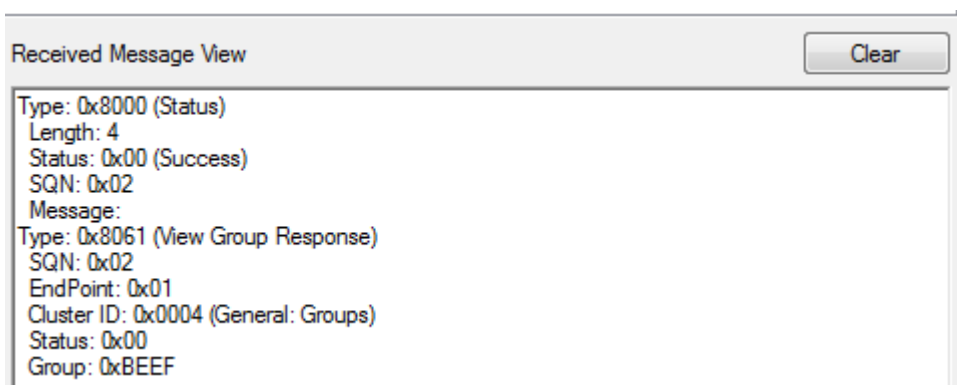


### 5.2.6.2 View Group

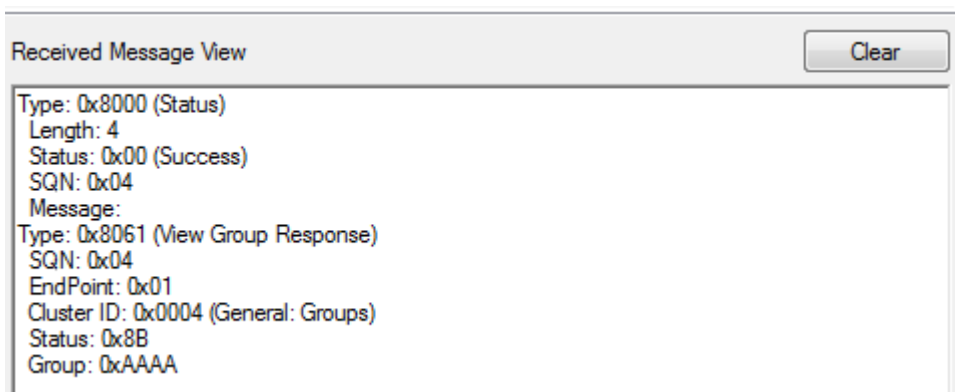
You can find out whether a device is a member of a specific group by sending a “View Group” command to the device. This is done in the **View Group** line of the interface by entering the network address of the device, source endpoint number, destination endpoint number and Group ID of the relevant group,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

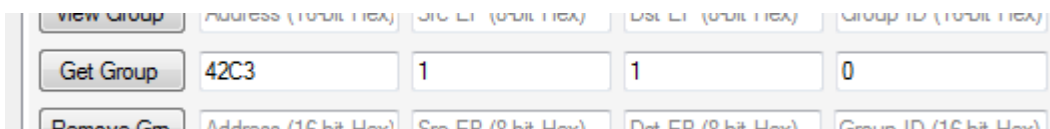


If the device is not a member of that group, you will receive a View Group Response with a status of “Not Found” (0x8B).

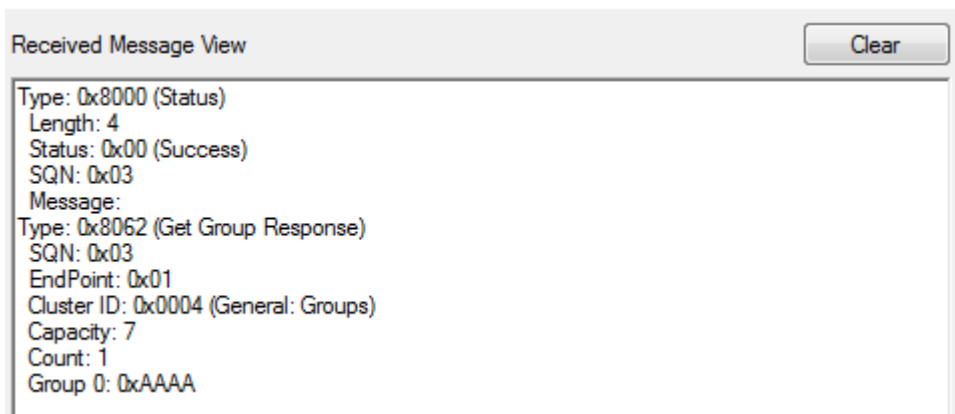


### 5.2.6.3 Get Group Membership

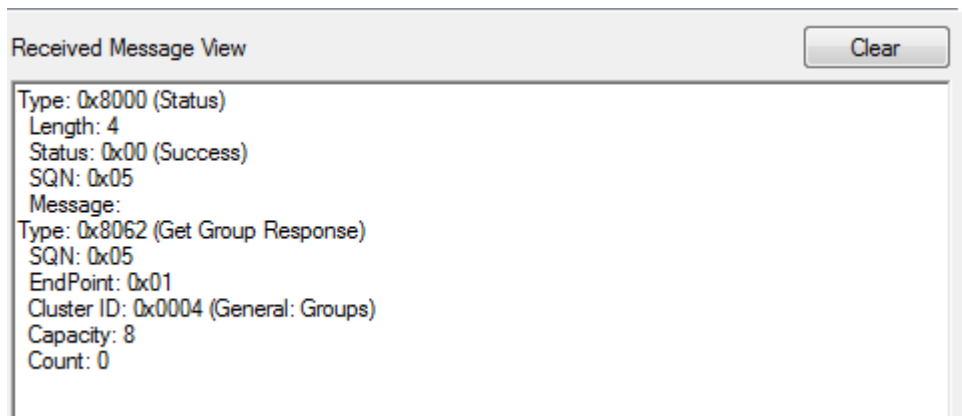
You can find out which groups a specific device is a member of by sending a “Get Group Membership” command to the device. This is done in the **Get Group** line of the interface by entering the network address of the device, source endpoint number, destination endpoint number and group count (number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



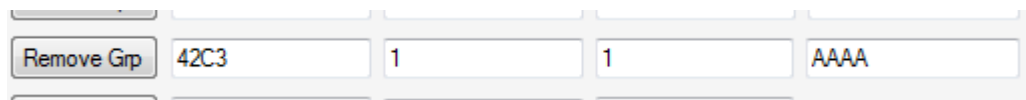
If the device is not a member of any groups, it will respond with an empty group list with a count of 0.



```
Received Message View Clear  
Type: 0x8000 (Status)  
Length: 4  
Status: 0x00 (Success)  
SQN: 0x05  
Message:  
Type: 0x8062 (Get Group Response)  
SQN: 0x05  
EndPoint: 0x01  
Cluster ID: 0x0004 (General: Groups)  
Capacity: 8  
Count: 0
```

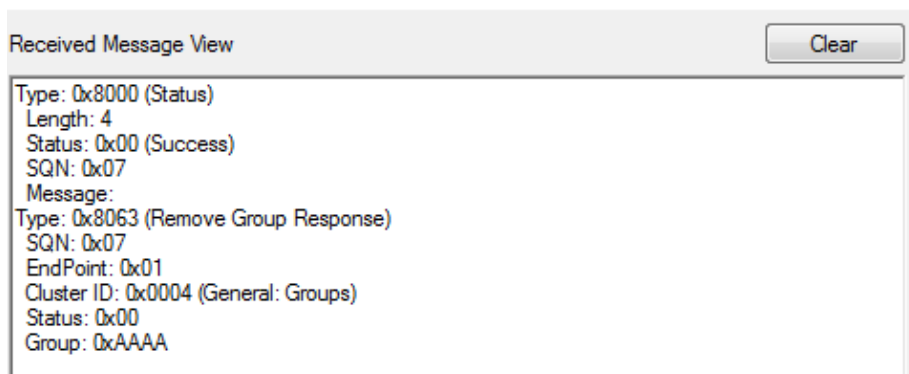
### 5.2.6.4 Remove Group

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.



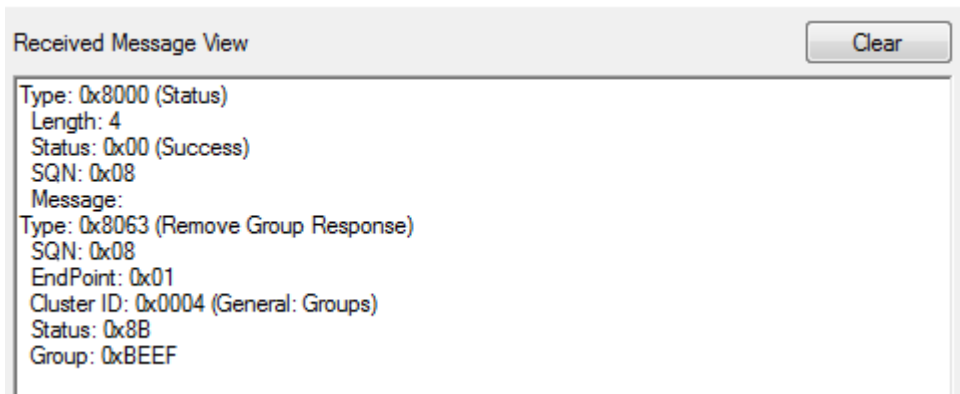
Remove Grp	42C3	1	1	AAAA
------------	------	---	---	------

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



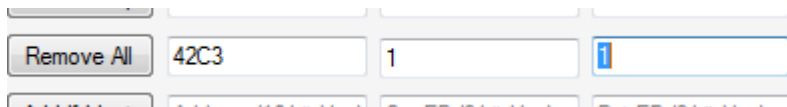
```
Received Message View Clear  
Type: 0x8000 (Status)  
Length: 4  
Status: 0x00 (Success)  
SQN: 0x07  
Message:  
Type: 0x8063 (Remove Group Response)  
SQN: 0x07  
EndPoint: 0x01  
Cluster ID: 0x0004 (General: Groups)  
Status: 0x00  
Group: 0xAAAA
```

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

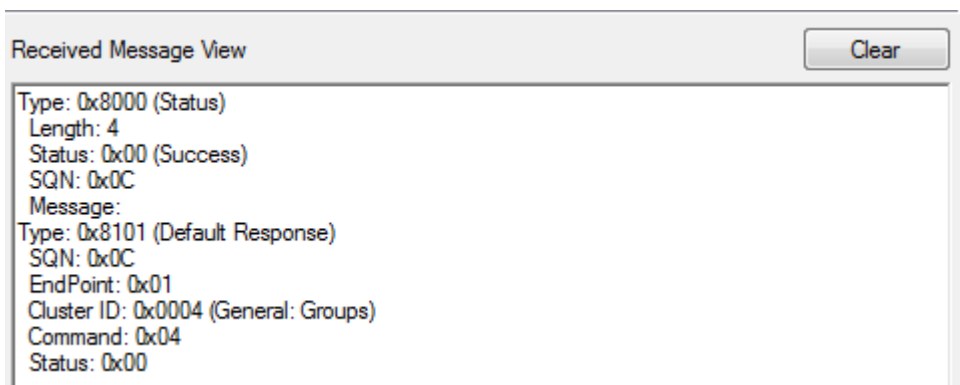


### 5.2.6.5 Remove All Groups

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.

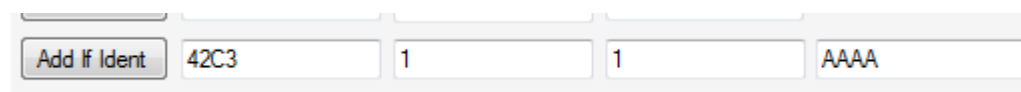


Irrespective of whether the device is associated with any groups, it will always respond with a status of “Success” (0x00).



### 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. You can also add a scene using the "Store Scene" command (see Section 5.2.7.2).

### 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.

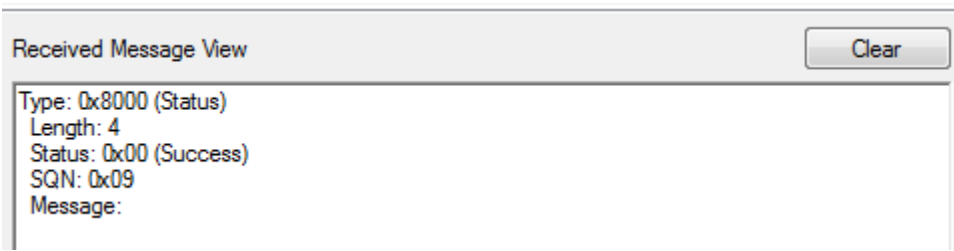
This results in the following "Store Scene Response" command which is displayed in the **Received Message View** pane.

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

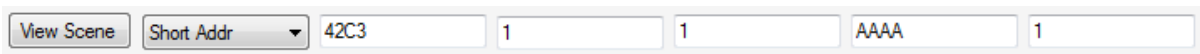
The "Recall Scene" command instructs a device to restore a previously saved scene in the device - for a light bulb, this could be restoring an on/off or level state. This is done in the **Recall Scn** line of the interface by entering the addressing mode, address of the device, source endpoint number,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

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

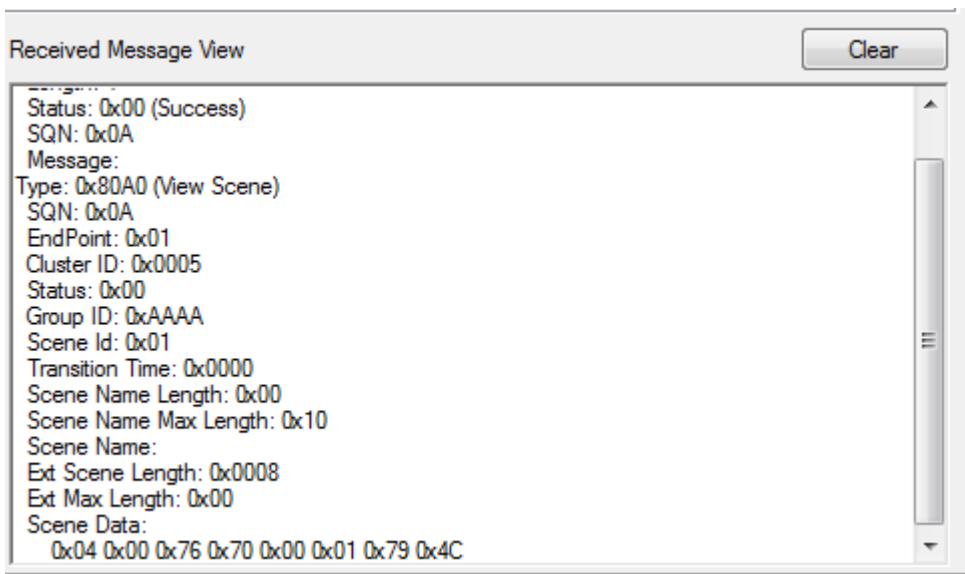


### 5.2.7.4 View Scene

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.

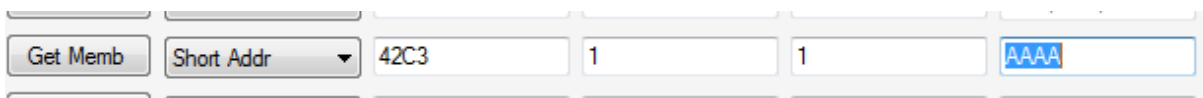


After sending a successful “View Scene” command, a response containing vital information like Transition time, Scene Name Length, Scene Name and Scene Data will be displayed in the **Received Message View** pane.

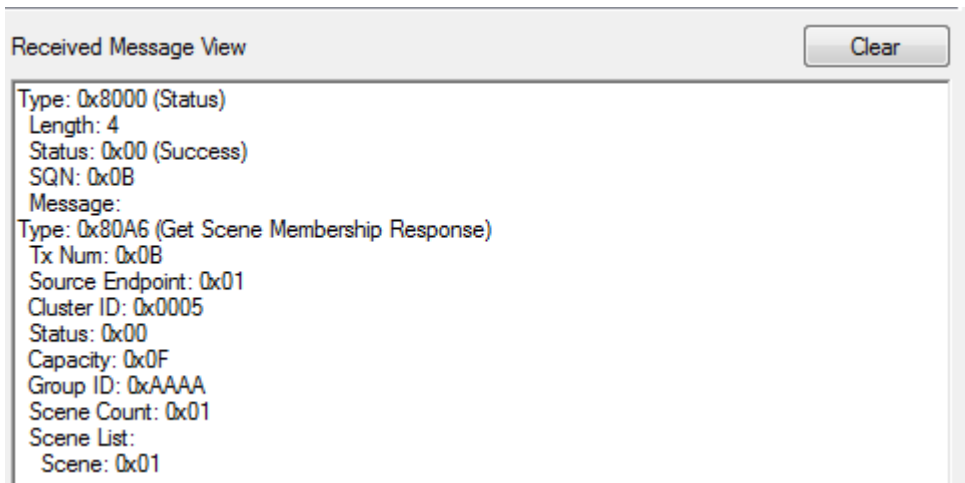


### 5.2.7.5 Get Scene Membership

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.

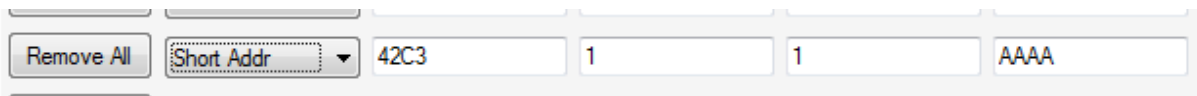


After sending a successful “Get Scene Membership” command, a response listing the number of scenes and the Scene IDs available will be displayed in the **Received Message View** pane.



### 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.

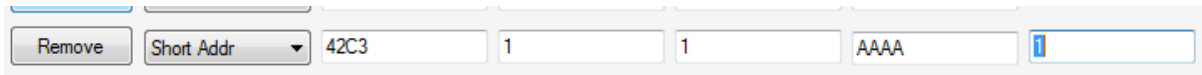


After sending a successful “Remove All Scenes” command, a response indicating whether the removal was successful will be displayed in the **Received Message View** pane.

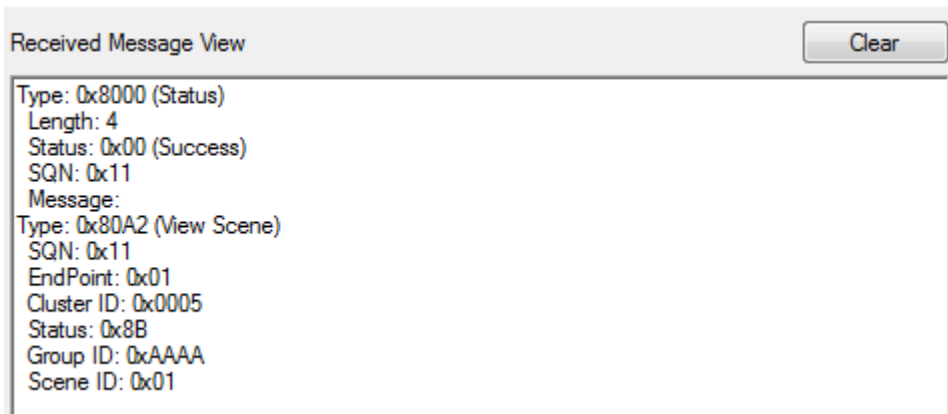


### 5.2.7.7 Remove Scene

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

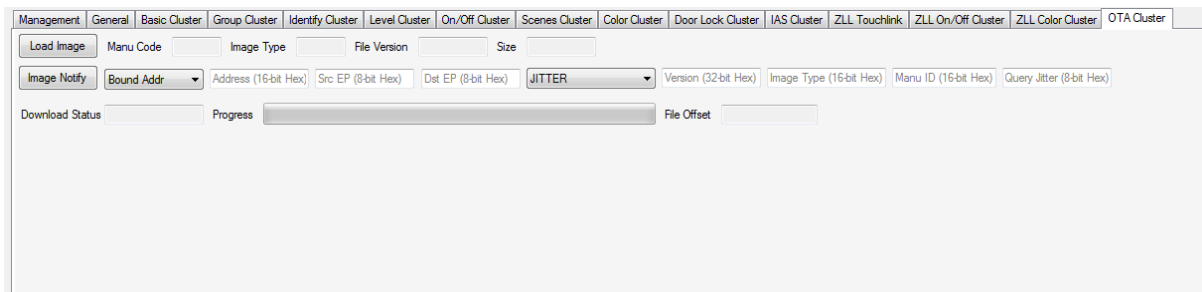
The ZGWUI provides an interface to perform an Over-The-Air (OTA) upgrade. This involves loading an application binary that will be served out ‘over the air’ to devices in the network. The following sections demonstrate how OTA upgrade is executed on the ZGWUI. This demonstration assumes that you have 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\_JN516x\_DR1175\_LED\_EXP\_MONO\_OTA\_Client\_v1.bin**

This application is supplied in the Application Note *ZigBee Home Automation Demonstration (JN-AN-1189)* and must be loaded into a network node (see Section 5.2.3).

### 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:

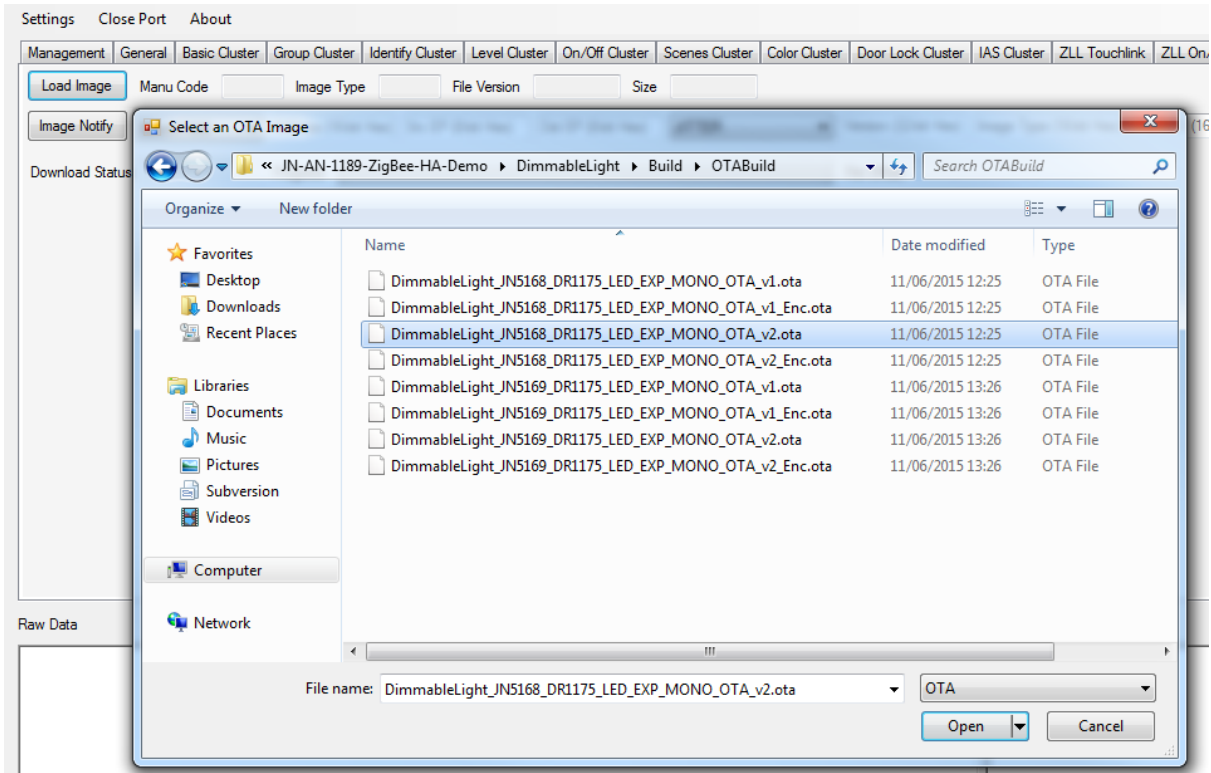




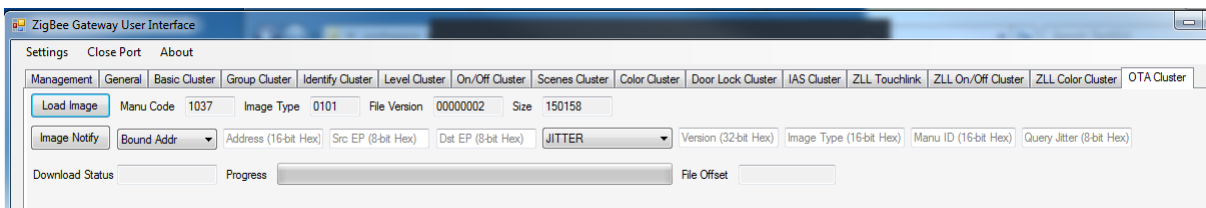
Click the **Load Image** button to bring up the file explorer window. Navigate to the folder which contains the OTA upgrade binary file that is to be used to upgrade the remote device and select the file – this is a **.ota** file, in this case:

**DimmableLight\_JN516x\_DR1175\_LED\_EXP\_MONO\_OTA\_v2.ota**

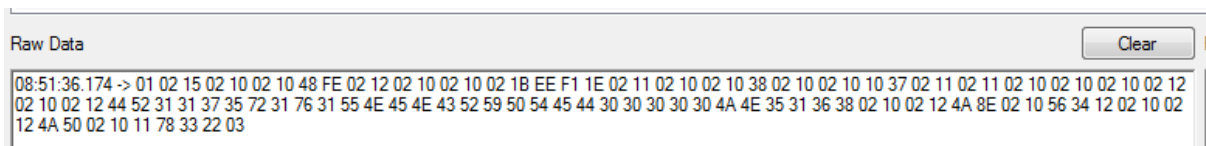
This file is supplied in the Application Note *ZigBee Home Automation Demonstration (JN-AN-1189)*, which must be present on your PC.



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-3103)*.

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.

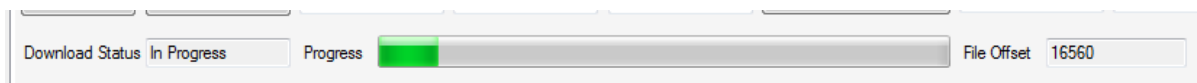


The above command notifies all relevant devices in the network and instructs all of them to upgrade straight away.

### 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.

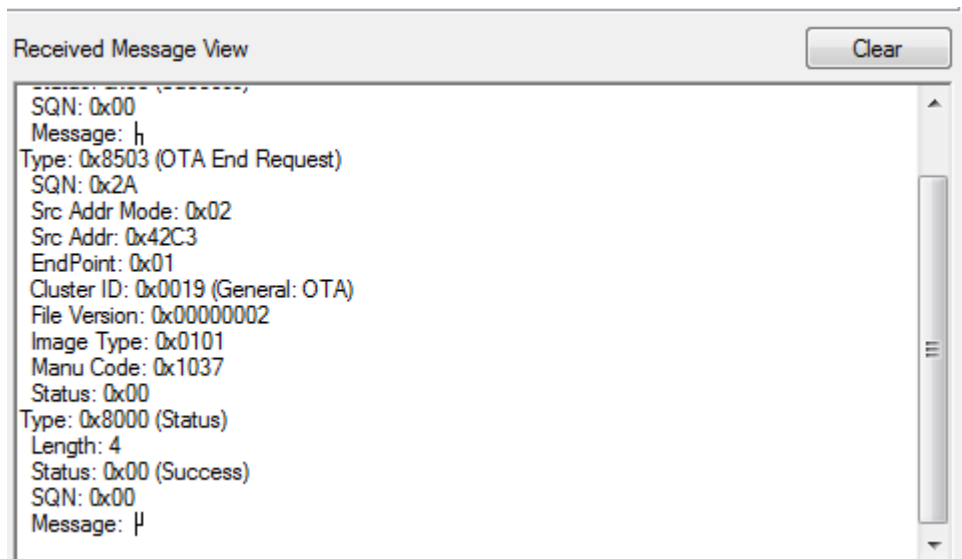


Note that there is only one progress bar and if you have multiple devices upgrading, the bar will appear slightly random, as it will reflect whichever device is requesting a block of data.

When a device has finished upgrading, the download status will change to “Complete” and the progress bar will reset.



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 Release Details

### 7.1 Compatibility

Product Type	Part Number	Build
<b>Version 1014</b>		
<b>Version 1013</b>		
<b>Version 1012</b>		
Evaluation Kit	JN516x-EK004	-
JN516x ZLL/HA SDK	JN-SW-4168	1620
'BeyondStudio for NXP' Toolchain	JN-SW-4141	1308
<b>Version 1011</b>		
Evaluation Kit	JN516x-EK004	-
JN516x ZLL/HA SDK	JN-SW-4168	1595
'BeyondStudio for NXP' Toolchain	JN-SW-4141	1308
<b>Version 1010</b>		
<b>Version 1009</b>		
<b>Version 1008</b>		
Evaluation Kit	JN516x-EK004	-
JN516x ZLL/HA SDK	JN-SW-4168	1470
'BeyondStudio for NXP' Toolchain	JN-SW-4141	1308
<b>Version 1007</b>		
Evaluation Kit	JN516x-EK004	-
JN516x ZLL/HA SDK	JN-SW-4168	1461
'BeyondStudio for NXP' Toolchain	JN-SW-4141	1308
<b>Version 1006</b>		
Evaluation Kit	JN516x-EK004	-
JN516x ZLL/HA SDK	JN-SW-4168	1455
'BeyondStudio for NXP' Toolchain	JN-SW-4141	1308
<b>Version 1005</b>		
<b>Version 1004</b>		
<b>Version 1003</b>		
<b>Version 1002</b>		
<b>Version 1001</b>		
Evaluation Kit	JN516x-EK004	-
JN516x ZLL/HA SDK	JN-SW-4168	1364
'BeyondStudio for NXP' Toolchain	JN-SW-4141	1308

## 7.2 New Features

ID	Feature	Description
<b>Version 1014</b>		
lpsw8374	Added Support for APS and MAC ACKs	Added support to the Control Bridge to write a message when an APS or MAC ACK occurs, if enabled.
lpsw8402	Added Support for Network State command to the ZGWUI	Added support for the Network State command to the ZGWUI, which allows the user to check the network details at any time.
<b>Version 1013</b>		
<b>Version 1012</b>		
lpsw7632	Added Fan Control Cluster Parsing	Added the Fan Control Cluster (0x0202) to the list of clusters that are parsed by the displayClusterId function.
lpsw8324	Made Add Scene Extension Fields Optional	The extension fields for the Add Scene function are now optional and can be shown or hidden by toggling a checkbox.
<b>Version 1011</b>		
lpsw7651	Added Combo Box to ZGWUI	Added a combo box to select the data type when performing a write attribute request.
lpsw7723	Added IAS WD Parsing	Added the IAS WD Cluster (0x0502) to the list of clusters that are parsed in the GUI
lpsw7725	Renamed IAS Cluster Tab	Renamed the 'IAS Cluster' tab to the 'IAS Zone Cluster' tab.
lpsw7735	Added IAS WD Cluster Tab	Added tab for IAS WD Cluster.
lpsw7752	Added Thermostat UI Config Cluster Parsing	Added the Thermostat UI Config Cluster (0x0204) to the list of clusters that are parsed in the GUI.
lpsw8161	Added Discovery Only Response	Added a Discovery Only Response to Control Bridge and ZGWUI to print received beacon details.
lpsw8162	Added Discovery Only Command	Added a Discovery Only command to Control Bridge and ZGWUI.
<b>Version 1010</b>		
lpsw7420	Hardware Debug added to makefile	Changed the makefile to allow a hardware debug version to be built with the correct file name extension in future.
lpsw7504	ZGWUI support for Group Name added	Added group name support to the ZGWUI.
lpsw7636	Fan Control Cluster added	Added Fan Control to the list of cluster that are parsed by the displayClusterId function.
lpsw7793	Enhanced binding layout	Reorganised the bind request layout to be more logical in the ZGWUI.
lpsw8142	Enhanced unbinding layout	Reorganised the unbind request layout to be more logical in the ZGWUI.
lpsw7913	Function to display all devices in network added	Added a Discover Devices function to the ZGWUI.
<b>Version 1009</b>		
lpsw6940	Discover Individual Attribute Response added	Added Discover Individual Attribute Response to the Control Bridge.
lpsw7519	OTA header string field added	Added fields to show more OTA image content when a file is loaded.
lpsw7637	Poll Control cluster client support added	Added Poll Control cluster client to Control Bridge.
lpsw7774	Poll Control cluster client commands added	Added Poll Control cluster client commands to ZGWUI.

lpsw7821	Option to request Default Responses added	Added an option to enable/disable the request for Default Responses to the Control Bridge.
lpsw7831	OTA image content added	Added fields that show additional OTA image content.
lpsw7856	Option to switch on/off Default Response request added to GUI	Added an option to enable/disable the request for Default Responses to the ZGWUI.
lpsw7914	Tooltips added for all fields	Added tooltips to all the fields on the ZGWUI to give more detailed information.
lpsw7915	Restore default text, if empty	Restored the default textbox text when it is empty in the ZGWUI
<b>Version 1008</b>		
<b>Version 1007</b>		
<b>Version 1006</b>		
lpsw7379	Start attribute added	Added a start attribute field to the 'discover all attributes' request. This allows multiple requests to be sent if all the attributes cannot fit in one response.
lpsw7415	ZGWUI parsing for Simple Sensor	The ZGWUI parses the device type 'Simple Sensor'.
lpsw7422	ZGWUI parsing for Binary Input cluster	The ZGWUI parses the cluster 'Binary Input'.
lpsw7439	ZGWUI parsing IAS ACE	The ZGWUI parses the device type 'IAS ACE'
lpsw7574	ZGWUI parsing Illuminance Measurement cluster	The ZGWUI parses the cluster 'Illuminance Measurement'
lpsw7576	Support for latest SDK	Makefile changes to support the latest JN-SW-4168 SDK.
<b>Version 1005</b>		
lpsw6841	Complex Descriptor Support	Added the 'complex descriptor request' to the ZGWUI and Control Bridge.
lpsw6934 lpsw6935	User Descriptor Support	Added the 'user descriptor request' and 'user descriptor set request' to the ZGWUI and Control Bridge.
lpsw7074 lpsw7351	OTA Wait for Data Support	Added support in the Control Bridge and ZGWUI for the 'OTA wait for data' command within the block response payload.
lpsw7350	Display decimal values for version	The ZGWUI displays the version of the application and the SDK on which it was built in decimal instead of hexadecimal format.
lpsw7227	Attribute report indication serial protocol definition	The 'attribute report indication' serial protocol command is now defined in the documentation.
<b>Version 1004</b>		
<b>Version 1003</b>		
<b>Version 1002</b>		
lpsw6924	Raw Data Send	Added feature to send raw payload which supports ZCL on host.
<b>Version 1001</b>		
N/A	SDK Version	Migrated to version 1364 of the JN-SW-4168 JN516x ZLL/HA SDK

## 7.3 Known Issues

ID	Severity	Description
Version 1014		
Version 1013		
Version 1012		
Version 1011		
Version 1010		
Version 1009		
Version 1008		
Version 1007		
Version 1006		
Version 1005		
Version 1004		
Version 1003		
Version 1002		
Version 1001		
None		

## 7.4 Bug Fixes

ID	Description
<b>Version 1014</b>	
lpsw7032	Fixed issue where the status was saved to the status buffer during E_ZCL_CBET_REPORT_INDIVIDUAL_ATTRIBUTE.
lpsw7220	ZGWUI previously had an issue where it would not decode Identify messages. This is now resolved.
lpsw7237	Fixed issue where the OTA progress bar would carry on flashing after an OTA upgrade was complete.
lpsw7711	There were typo errors for Remove Scene and Remove All Scenes responses in the ZGWUI. These are now parsed correctly.
lpsw8372	Previously, the sequence number was not set. Resolved this by setting it equal to eZCL_GetLastSequenceNumber.
lpsw8373	The stack event was not set to APS ACK, and therefore was never triggered. Solved this issue.
lpsw8431	Fixed an issue where a child that changed parents would not be removed from the Router's neighbour table because it was powered off.
<b>Version 1013</b>	
<b>Version 1012</b>	
lpsw8357	Fixed issue where OTA Upgrade Time would use 0xffffffff by default, therefore the image does not upgrade. OTA now upgrades after 5 seconds.
<b>Version 1011</b>	
lpsw8231	Fixed issue where the channel check on Discovery Only response beacons was wrong and would look for a beacon on a different channel.
lpsw8263	Fixed issue where a stack dump would occur when Discovery Only command was sent and no channel was selected on the GUI.
lpsw8267	Fixed issue where a Discovery Only command could only be sent once.
<b>Version 1010</b>	
lpsw7038	Fixed issue where the APS Data Confirm Fail command was not parsed correctly in the GUI.
lpsw7500	Updated the GUI to display the correct name when both the Thermostat Device ID and Cluster ID are received.
lpsw7713	Fixed issue where user input text in a text box would be cleared upon mouse click in the ZGWUI.
lpsw7929	Fixed issue where the APDU was not being freed when service discovery ZPS functions were called.
lpsw7943	Fixed issue where the ZCB Active Endpoints Request command listed incorrect endpoint values.
lpsw7944	Fixed issue where a status message was only copying one byte to the buffer when it should be two bytes.
lpsw7951	Fixed issue where the ZCB Get Membership command listed incorrect scene values.
lpsw7984	Fixed an issue that could cause a CRC failure due to adding a NULL value when failing to read a byte off the queue.
lpsw8110	Fixed issue where debug was not available on UART1.
lpsw8111	Fixed issue with Scene Name where only one character could be passed instead of the full character array.
lpsw8120	Fixed ZGWUI issue where some fields did not format properly when grey text was restored.
<b>Version 1009</b>	
lpsw6761	Fixed issue by which the Get Version command would receive two status messages in response (rather than one).
lpsw7586	Implemented Add Scene function in full.
lpsw7650	Fixed incrementing issue in sendWriteAttribRequest loop.
lpsw7855	Added 'Add Scene' command to the ZGWUI.



## ZigBee IoT Gateway Control Bridge

lpsw7661	Fixed issue with Remove Device command sending the parent IEEE address instead of the child IEEE address
<b>Version 1008</b>	
<b>Version 1007</b>	
<b>Version 1006</b>	
lpsw6953	Fixed String bug when sending attributes of type string. Previously they were sent with a size of 1.
lpsw7241	ZGWUI did not parse the string value correctly.
lpsw7498	The destination and source endpoint for the 'Add Group' command were inconsistent with the rest of the ZGWUI application.
lpsw7569	Fixed Touchlink bug where it would send an encrypted network key as all zeros.
lpsw7568	Fixed Touchlink bug where it would send Touchlink commands at full power.
lpsw7575	Fixed Debug bug where turning on debug in the makefile was using the 'DEBUG' instead of using the 'DEBUG_PORT' flag.
lpsw7577	Fixed bug with ZGWUI where the 'Bind Target Extended Address' tooltip was not working.
<b>Version 1005</b>	
lpsw7238	Fixed versioning bug when loading an OTA image into the ZGWUI.
lpsw7349	Fixed 'get version' command which always responded with version 0.
lpsw7348	Removed a label in the ZGWUI which should not have been present.
lpsw7347	Fixed version number which was being incorrectly parsed from the ZGWUI.
lpsw7148	Updated description of 'move to level with/without on/off' command in Serial Command Set appendix to include missing OnOff byte.
lpsw7342	Removed the ZGWUI-User-Guide.pptx as all the information is within this document.
<b>Version 1004</b>	
N/A	Build release issue that inserted an SDK path and APP path onto the build configuration line which caused a compile error.
<b>Version 1003</b>	
lpsw6964	Fixed such that the Identify Query response from the Control Bridge only returns the full response payload on success. In the case of failure, it only responds with the group, cluster and status.
lpsw6968	Fixed such that the Identify Query command handler in the Control Bridge reads bytes 5 and 6 for the identify time. Previously, it was reading bytes 6 and 7, meaning byte 5 was empty.
lpsw6969	ZGWUI sends the correct Identify Query request command payload. This means it is now sending one less byte, as it was previously sending an empty byte.
lpsw7081	The Control Bridge now resets after receiving an Erase EEPROM command.
lpsw7082	Fixed such that the application state is now consistent with the ZigBee PRO stack state.
lpsw7132	Fixed such that the Write Attribute request in the ZGWUI is now doing a 'Char to Hex' byte copy as opposed to an ASCII array copy.
lpsw7136	The Write Attribute command is no longer adding payload space for a manufacturer code when it is not needed, which caused it to appear as two attributes in the sniffer.
<b>Version 1002</b>	
lpsw6810	Version number has been created to match the application note version.
lpsw6881	Prevented the possibility of sending a group membership request with empty list.
<b>Version 1001</b>	
None	

## 7.5 Protocol Changes

ID	Description
<b>Version 1014</b>	
lpsw7585	Default Response (msg type 0x8101) has been modified and now includes Short Address and Source Endpoint. Please see Serial Command Set.
lpsw8401	Added a command to check network details.
<b>Version 1013</b>	
<b>Version 1012</b>	
<b>Version 1011</b>	
<b>Version 1010</b>	
lpsw7503	Add Group (msg type 0x0060) has been modified and now includes Group Name, Group Name Length, and Group Name Maximum Length. Please see Serial Command Set.
<b>Version 1009</b>	
lpsw7785	Read Attribute, Write Attribute response and Report Attribute payload has been modified which includes two bytes of attribute size. Please see Serial Command Set.
lpsw7840	Add Scene command has been modified and now includes Scene Data Length and Scene Data. Please see Serial Command Set.
lpsw7856	Added an option to enable/disable the request for Default Responses.
lpsw7637	Added a command to set the poll control check-in response parameters.
<b>Version 1008</b>	
<b>Version 1007</b>	
<b>Version 1006</b>	
<b>Version 1005</b>	
<b>Version 1004</b>	
<b>Version 1003</b>	
<b>Version 1002</b>	
<b>Version 1001</b>	
None	

## Appendix A: Serial Protocol

### A.1. Physical Characteristics

The serial link between the ZGWUI (ZigBee Gateway User Interface) and wireless microcontroller runs at 1Mbaud when the JN516x is contained in a USB dongle. 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	<pre>&lt;status:uint8_t&gt; &lt;sequence number: uint8_t&gt; &lt;Packet Type: uint16_t&gt; &lt;Optional additional error information: string&gt;</pre> <p>Status:</p> <ul style="list-style-type: none"> <li>0 = Success</li> <li>1 = Incorrect parameters</li> <li>2 = Unhandled command</li> <li>3 = Command failed</li> <li>4 = Busy (Node is carrying out a lengthy operation and is currently unable to handle the incoming command)</li> <li>5 = Stack already started (no new configuration accepted)</li> <li>128 – 244 = Failed (ZigBee event codes)</li> </ul> <p>Packet Type: The value of the initiating command request.</p>	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	<pre>&lt;log level: uint8_t&gt; &lt;log message : string&gt;</pre> <p>Log Level :</p> <p>Use the Linux / Unix log levels</p> <ul style="list-style-type: none"> <li>0 = Emergency</li> <li>1 = Alert</li> <li>2 = Critical</li> <li>3 = Error</li> <li>4 = Warning</li> <li>5 = Notice</li> <li>6 = Information</li> <li>7 = Debug</li> </ul>	
Node->Host	Data Indication Msg Type = 0x8002	<pre>&lt;status: uint8_t&gt; &lt;Profile ID: uint16_t&gt; &lt;cluster ID: uint16_t&gt; &lt;source endpoint: uint8_t&gt; &lt;destination endpoint: uint8_t&gt; &lt;source address mode: uint8_t&gt; &lt;source address: uint16_t or uint64_t&gt; &lt;destination address mode: uint8_t&gt; &lt;destination address: uint16_t or uint64_t&gt; &lt;payload size : uint8_t&gt; &lt;payload : data each element is uint8_t&gt;</pre>	
Node->Host	Node Cluster List – Sent by gateway node after reset Msg Type = 0x8003	<pre>&lt;source endpoint: uint8_t t&gt; &lt;profile ID: uint16_t&gt; &lt;cluster list: data each entry is uint16_t&gt;</pre>	

## ZigBee IoT Gateway Control Bridge

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
Host->Node	Default response request option Msg Type = 0x0008	<Enabled: bool_t>	
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 HA mode 1 = Router ZLL mode (pure Control Bridge) 2 = Router ZLL with HA compatibility (Control Bridge with HA and ZLL security)	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	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	Network State Request Msg Type = 0x0009	No payload	Status, followed by “Network State” response

Node->Host	Network State Response Msg Type=0x8009	<Short Address: uint16_t> <Extended Address: uint64_t> <PAN ID: uint16_t> <Extended PAN ID: uint64_t> <Channel: uint8_t>	
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 1 - WAIT_START, 2 - NFN_START, 3 - DISCOVERY, 4 - NETWORK_INIT, 5 - RESCAN, 6 - RUNNING  The node is provisioned from previous restart.	
Node->Host	"Factory New" Restart Msg Type=0x8007	Status –  0 - STARTUP 1 - WAIT_START, 2 - NFN_START, 3 - DISCOVERY, 4 - NETWORK_INIT, 5 - RESCAN, 6 - RUNNING  The node is not yet provisioned.	
Host->Node	Erase Persistent Data Msg Type = 0x0012	No payload	Status
Host->Node	Bind Msg Type = 0x0030	<target extended address: uint64_t> <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	



## ZigBee IoT Gateway Control Bridge

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: uint8_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: uint8_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	<p>&lt;Sequence number: uint8_t&gt;                  &lt;Status uint8_t&gt;                  &lt;network address: uint16_t&gt;                  &lt;manufacturer code: uint16_t&gt;                  &lt;max Rx Size: uint16_t&gt;                  &lt;max Tx Size: uint16_t&gt;                  &lt;server mask: uint16_t&gt;                  &lt;descriptor capability: uint8_t&gt;                  &lt;mac flags: uint8_t&gt;                  &lt;max buffer size: uint8_t &gt;                  &lt;bit fields: uint16_t&gt;</p> <p>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))</p> <p>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</p> <p>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</p> <p>Descriptor capability:                      0 - extended Active endpoint list available                      1 - Extended simple descriptor list available                      2 to 7: Reserved</p>	
Host->Node	Simple Descriptor request Msg Type = 0x0043	<target short address: uint16_t> <endpoint: uint8_t>	Status Simple Descriptor response

## ZigBee IoT Gateway Control Bridge

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	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: uint8_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 (0xffff) 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
Node->Host	Read Attribute response Msg Type = 0x8100	<Sequence number: uint8_t> <Src address : uint16_t> <Src endpoint: uint8_t> <Cluster id: uint16_t> <Attribute id: uint16_t> <Attribute status: uint8_t> <Attribute type: uint8_t> <Attribute Size: uint16_t> <Attribute value: depends on type>	
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>	Data Indication Msg Type = 0x8002

		<p>&lt;direction: uint8_t&gt;                  &lt;manufacturer specific: uint8_t&gt;                  &lt;manufacturer id: uint16_t&gt;                  &lt;number of attributes: uint8_t&gt;                  &lt;attributes list: data list of uint16_t each&gt;</p> <p>Direction:                  0 - from server to client                  1 - from client to server</p> <p>Manufacturer specific :                  1 – Yes                  0 – No</p>	
Node->Host	Write Attribute response Msg Type = 0x8110	<p>&lt;Sequence number: uint8_t&gt;                  &lt;Src address : uint16_t&gt;                  &lt;endpoint: uint8_t&gt;                  &lt;Cluster id: uint16_t&gt;                  &lt;Attribute id: uint16_t&gt;                  &lt;Attribute status: uint8_t&gt;                  &lt;Attribute type: uint8_t&gt;                  &lt;Attribute Size: uint16_t&gt;                  &lt;Attribute value: depends on type&gt;</p>	
Host->Node	Attribute Discovery request Msg Type = 0x0140	<p>&lt;address mode: uint8_t&gt;                  &lt;target short address: uint16_t&gt;                  &lt;source endpoint: uint8_t&gt;                  &lt;destination endpoint: uint8_t&gt;                  &lt;Cluster id: uint16_t&gt;                  &lt;Attribute id : uint16_t&gt;                  &lt;direction: uint8_t&gt;                  &lt;manufacturer specific: uint8_t&gt;                  &lt;manufacturer id: uint16_t&gt;                  &lt;Max number of identifiers: uint8_t&gt;</p> <p>Direction:                  0 - from server to client                  1 - from client to server</p> <p>Manufacturer specific :                  1 – Yes                  0 – No</p>	Status Attribute Discovery response
Node->Host	Attribute Discovery response Msg Type = 0x8140	<p>&lt;complete: uint8_t&gt;                  &lt;attribute type: uint8_t&gt;                  &lt;attribute id: uint16_t&gt;</p> <p>Complete:                  0 – more attributes to follow                  1 – this was the last attribute</p>	
Node->Host	Attribute Discovery Individual Response Msg Type = 0x8139	<p>&lt;complete: uint8_t&gt;                  &lt;attribute type: uint8_t&gt;                  &lt;attribute id: uint16_t&gt;</p> <p>Complete:                  0 – more attributes to follow                  1 – this was the last attribute</p>	
Host->Node	Enable Permissions Controlled Joins Msg Type = 0x0027	<p>&lt;Enable/Disable : uint8_t&gt;                  1 – Enable                  2 – Disable</p>	Status
Host->Node	Authenticate Device Msg Type = 0x0028	<p>&lt;IEEE address ; uint64_t&gt;                  &lt;Key : 16 elements byte each&gt;</p>	Status Authenticate response
Node->Host	Authenticate response Msg Type = 0x8028	<p>&lt;IEEE address of the Gateway: uint64_t&gt;                  &lt;Encrypted Key : uint8_t 16 elements&gt;                  &lt;MIC : uint8 4 elements&gt;                  &lt;IEEE address of the initiating node : uint64_t&gt;                  &lt;Active Key Sequence number : uint8_t&gt;</p>	

		<p>&lt;Channel : uint8_t&gt;                  &lt;Short PAN Id : uint16_t&gt;                  &lt;Extended PAN ID : uint64_t&gt;</p>	
Host->Node	<p>Configure Reporting request                  Msg Type = 0x0120</p>	<p>&lt;address mode: uint8_t&gt;                  &lt;target short address: uint16_t&gt;                  &lt;source endpoint: uint8_t&gt;                  &lt;destination endpoint: uint8_t&gt;                  &lt;Cluster id: uint16_t&gt;                  &lt;direction: uint8_t&gt;                  &lt;manufacturer specific: uint8_t&gt;                  &lt;manufacturer id: uint16_t&gt;                  &lt;number of attributes: uint8_t&gt;                  &lt;attributes list: data list of uint16_t each&gt;                  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</p>	<p>Status                  Configure Reporting response</p>
Node->Host	<p>Configure Reporting response                  Msg Type = 0x8120</p>	<p>&lt;Sequence number: uint8_t&gt;                  &lt;Src address : uint16_t&gt;                  &lt;Endpoint: uint8_t&gt;                  &lt;Cluster id: uint16_t&gt;                  &lt;Status: uint8_t&gt;</p>	
Host->Node	<p>Read Reporting request                  Msg Type = 0x0122</p>	<p>&lt;address mode: uint8_t&gt;                  &lt;target short address: uint16_t&gt;                  &lt;source endpoint: uint8_t&gt;                  &lt;destination endpoint: uint8_t&gt;                  &lt;Cluster id: uint16_t&gt;                  &lt;direction: uint8_t&gt;                  &lt;number of attributes: uint8_t&gt;                  &lt;manufacturer specific: uint8_t&gt;                  &lt;manufacturer id: uint16_t&gt;                  Attribute direction : uint8_t                  Attribute id : uint16_t</p>	<p>Status                  Read Reporting response</p>
Host->Node	<p>Read Reporting response                  Msg Type = 0x8122</p>	<p>&lt;Sequence number: uint8_t&gt;                  &lt;Src address : uint16_t&gt;                  &lt;Endpoint: uint8_t&gt;                  &lt;Cluster id: uint16_t&gt;                  &lt;Status: uint8_t&gt;                  Attribute type : uint8_t                  Attribute id : uint16_t                  Min interval : uint16_t                  Max interval : uint16_t</p>	
Node->Host	<p>Attribute Report                  Msg Type = 0x8102</p>	<p>&lt;Sequence number: uint8_t&gt;                  &lt;Src address : uint16_t&gt;                  &lt;Endpoint: uint8_t&gt;                  &lt;Cluster id: uint16_t&gt;                  &lt;Attribute Enum: uint16_t&gt;                  &lt;Attribute Status: uint8_t&gt;                  &lt;Attribute Type: uint8_t&gt;                  &lt;Attribute Size: uint16_t&gt;                  &lt;Attribute value: depends on type&gt;</p>	
Node->Host	<p>Default response                  Msg Type = 0x8101</p>	<p>&lt;Sequence number: uint8_t&gt;                  &lt;Short Address : uint16_t&gt;                  &lt;Source Endpoint : uint8_t&gt;                  &lt;Destination Endpoint: uint8_t&gt;                  &lt;Cluster id: uint16_t&gt;                  &lt;Command Id: uint8_t&gt;                  &lt;Status code: uint8_t&gt;</p>	

### 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> <group name length: uint8_t> <group name maximum length: uint8_t> <group name: string>	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: uint8_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> <destination endpoint: uint8_t> <time: uint16_t> Time: Seconds	Status
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> <destination endpoint: uint8_t> <command ID: uint8_t> Command Id 0 – Off 1 - On 2 - Toggle	Status
Host->Node	On / Off Timed Send Msg Type = 0x0093	<address mode: uint8_t> <target short address: uint16_t> <source endpoint: uint8_t> <destination endpoint: uint8_t> <onoff: uint8_t> <on time: uint16_t> <off time: uint16_t> On / Off: 0 = Off 1 = On Time: Seconds	Status

## B.1.7. Scenes Cluster Commands

Message Direction	Message Description	Message Format	Expected Response
Host->Node	View Scene 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> <scene data length: uint16_t> <scene 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: uint16_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>	

## ZigBee IoT Gateway Control Bridge

		<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. 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: uint16_t> <transition time: uint8_t> <scene name:string> <length: uint8_t> <max length: uint8_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: uint16_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: uint16_t> <to group ID: uint16_t> <to scene ID: uint16_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: uint8_t> <transition time: uint8_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: uint8_t> <minimum temperature: uint8_t> <maximum temperature: uint8_t>	Status Data indication
Host->Node	Step colour temperature	<address mode: uint8_t> <target short address: uint16_t>	Status Data indication

	Msg Type = 0x00C2	<source endpoint: uint8_t> <destination endpoint: uint8_t> <mode: uint8_t> <step size: uint8_t> <transition time: uint8_t> <minimum temperature: uint8_t> <maximum temperature: uint8_t>	
--	-------------------	--	--

## 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.3.2 Poll Control Commands

Message Direction	Message Description	Message Format	Expected Response
Host->Node	Set Poll Control Check-in parameters Msg Type = 0x0A00	<Enable Fast Polling: bool_t> <Fast Poll Timeout: uint16_t>	Status

## 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 JN5168 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 startup. In the tables below, the Node refers to the Control Bridge

Direction	Message
Host->Node	<del>Erase Persistent Data (Optional)</del>
Node->Host	<del>Status (If Erase command issued)</del>
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
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 HA and ZLL:

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

For ZLL bulbs:

- Identify Trigger Effect (0x00E0)

## C.10. Scene Management

No sequence required – issue commands and get status back.

For HA devices:

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

For 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 HA bulbs:

- 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 ZLL colour bulbs:

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

## Revision History

Version	Notes
1000	First internal release
1001	Released on the JN-SW-4168 SDK v1364
1002	Added group membership and Raw Data send feature
1003	Bugs fixed in software, as described in Section 7.4
1004	Compilation issue fixed, as described in Section 7.4
1005	Features added in software and bugs fixed, as described in Section 7.2 and Section 7.4
1006	Rebuilt for JN-SW-4168 SDK v1455. New software features added and bugs fixed, as described in Section 7.2 and Section 7.4
1007	Rebuilt for JN-SW-4168 SDK v1461.
1008	Rebuilt for JN-SW-4168 SDK v1470.
1009	New software features added and bugs fixed, as described in Section 7.2 and Section 7.4
1010	New software features added and bugs fixed, as described in Section 7.2 and Section 7.4
1011	Rebuilt for JN-SW-4168 SDK v1595.
1012	Rebuilt for JN-SW-4168 SDK v1620. New software features added and bugs fixed, as described in Section 7.2 and Section 7.4
1013	Fixed an issue that prevented ZigBeeNodeControlBridge.zpscfig from being edited outside of a text editor.
1014	New software features added and bugs fixed, as described in Section 7.2 and Section 7.4

## Important Notice

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

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

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

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

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

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

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

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

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

All trademarks are the property of their respective owners.

## NXP Semiconductors

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

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