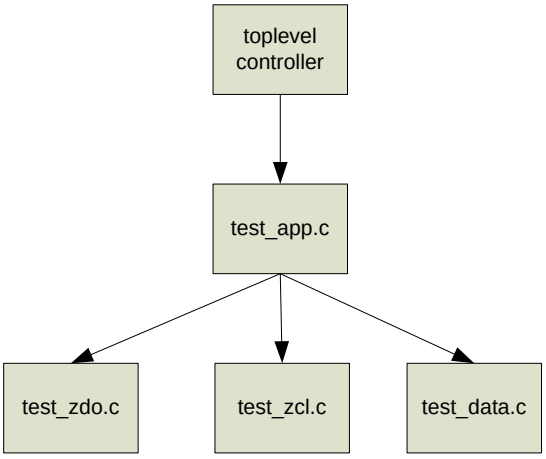
FreakLabs FreakZ Simulator and Hardware Command- Line Interface User Guide

1. Intro

The FreakZ simulator and raven-usb hardware build both have identical command line interfaces that can be used to access many of the Zigbee application level functions. This document is a guide on how to use the commands in the command line interface as well as hardware setup, simulator setup, and command line file hierarchy.

2. File Structure

The files that govern the command line application interface are located in the <root dir>/test/test_sim and <root dir>/test/test_avr_ravenusb directories. They follow this basic hierarchy:



toplevel controller – This file is named test_sim.c for the simulator and test_avr_ravenusb.c for the raven usb sticks. It contains the main communication routines for communicating with the node. In the case of the simulator, it has the OSes pipe communication initialization routines which allows it to talk to the command shell, sim.exe. In the USB sticks, test_avr_ravenusb.c contains the USB/serial rx handler which parses the received data and sends it to the command line located in test_app.

test_app.c – This is the main application code for the command line and contains the command table, menu, and command line parsing. New commands would need to be added to the command table before they could be used by the command line. This file also contains basic functions that are not part of the ZCL or ZDO and are used mainly for debugging and testing.

test_zdo.c – This file specifies the ZDO endpoint and contains all ZDO specific functions. These commands act directly on the ZDO endpoint of the FreakZ Zigbee stack.

test_zcl.c – This file specifies the TEST_ZCL endpoint and contains all ZCL specific functions. The commands in this file act directly on the ZCL endpoint of the FreakZ Zigbee stack.

test_data.c – This file contains a manufacturer specific (to FreakLabs, hehe) endpoint and contains generic data transfer functions. This endpoint exists so that the user can transmit and receive raw data without worrying about going through the ZDO or ZCL. This is useful if they add

any extra functionality that affects the data path in the stack.

3. Simulator Specific Commands

The simulator has a list of commands specific to the simulator and used to add nodes, list nodes, and direct commands to specific nodes. In fact, that's pretty much the complete command line. The scripting functionality present in previous versions is obsolete because the process synchronization is difficult to maintain between different systems. This might get added back in at a later date. The commands are as follows:

3.1 Add Node

Cmd: add [node index]

Args:

node index – This is the index of the node to be added and is just used to simplify sending commands to the node. Otherwise, you'd need to send commands to the node via the node process ID which is difficult to remember. The index number needs to be from 1 to 7.

This creates a new process and starts an instance of the Zigbee stack within the process. Rather than using the process ID, you direct commands to the node via the index number that is specified. Any index can be a coordinator node.

Ex: add 1

Add a node whose index is 1. This will open a terminal window and the FreakZ stack will be executed within it. All commands for this node will need to use the index of 1.

3.2 List Nodes

Cmd: list

Args:

None

This command lists the all active nodes and shows their index number and process ID. This is especially useful for debugging with GDB since you can attach GDB to the process ID and debug the node.

Ex: list

Lists all active nodes, indices, and process IDs.

3.3 Send command to Node

Cmd: cmd [index] [command]

Args:

index – Index number of node where command will be directed

command – Node's command line command and arguments. Please see command line command list below on how to use them and what arguments are required.

This command is very useful and allows you to send commands to any node in the simulator via the main command shell. You need to prefix the command line command with 'cmd n' and then add the command you want to issue. The node's terminal window should give an indication if it received the command and will echo the command line command back, minus the 'cmd n' which gets stripped off by the simulator's shell.

Ex: cmd 1 zs c

Issue a command to node 1 to start as a Zigbee coordinator.

4. Starting and Joining Nodes

Before you start the simulator, you should check the Doxygen documentation on how to build and run the code. Once the code and simulator are built, then follow the instructions on starting the simulator. After the simulator command line :

1) type ‘add 1’ to add a node with an index of 1
2) type ‘add 2’ to add a node with an index of 2
3) type ‘cmd 1 zs c’ to start node 1 as a Zigbee coordinator. Node 1 will form the network and appoint itself the coordinator.
4) type ‘cmd 2 zs’ to start node 2 as a Zigbee router. Node 2 will scan the network and should join node 1.

If you are running on real hardware using the Raven USB sticks, then you would need to use the following procedure:

1) Insert stick 1 and get the COM port number (ie: From Device Manager in Windows)
2) Insert stick 2 and get the COM port number
3) Start a terminal window and select the COM number for stick 1
4) Start a terminal window and select the COM number for stick 2
5) type ‘zs c’ in the terminal window for stick 1
6) type ‘zs’ in the terminal window for stick 2

If all goes well, then either way, you should have two nodes that are joined. To join more nodes, just add more nodes in the simulator and start them as a router or add more sticks and terminals and start them as routers.

5. Command Line Commands

Here is the command description for the current list of commands in the simulator and the hardware command line. For the simulator command line, you need to add a “cmd x” before the command to notify the command shell which node you’re issuing the command to. For example. if you want to issue a command to node 3, you would need to type “cmd 3 <command>”. This isn’t needed when you’re running in real hardware since you’re typing the command directly on the node’s command line.

For some functions, the print output might be missing on the hardware build. This is because printf stores the string in RAM and having a lot of printf’s will eat up the limited RAM available. I’ll be testing out the printf_P function in the future which stores the strings in flash. If this works out well, then the code will be modified so that the hardware builds store the strings in flash.

Unless otherwise specified, all numeric command arguments will be in hexadecimal.

5.1 General App Commands

These are commands that are of general usage to the stack and aren’t specific to the ZDO or ZCL. They mostly operate directly on the node rather than sending out a request to a remote node and either control the node directly or provide information about the internal structures on it.

5.1.1 Hello

Cmd: hello

Args: None

This is a reality check command that prints out the “hello” string to the command line. It’s just used to show that the command line interface is operating properly and communications with the node are enabled.

Ex: hello

This should output the string “hello” in node 1’s terminal window.

5.1.2 Zigbee Start

Cmd: zs [c]

Args:

c – start as coordinator

This is the first command that should be sent to the node. It takes one argument which is optional and will allow the user to choose to start the node as a coordinator or a router. For a coordinator, the command will initiate the network formation sequence. For a router, the command will initiate the network scan and join procedure.

Ex: zs c
Starts the node as a coordinator and initiates network formation.

Ex: zs
Starts the node as a router and initiates a network scan and a join procedure if a potential parent is found.

5.1.3 Dump NIB

Cmd: nib
Args: None

Dumps the Network Information Base (NIB). The NIB contains all the configuration parameters for the Zigbee Network layer.

Ex: nib
Shows the current NIB values.

5.1.4 Dump PIB

Cmd: pib
Args: None

Dumps the 802.15.4 MAC and PHY information base. Yeah, I could have called it MIB. The naming choice was an unfortunate early decision. The PIB contains all the configuration parameters for the MAC and PHY.

Ex: pib
Shows the current PIB values.

5.1.5 Show Used Buffers

Cmd: buf
Args: None

Dumps the current number of buffers in use. Mostly used to check if there is a buffer leak after a transaction sequence.

Ex: buf
Shows the number of buffers currently in use.

5.1.6 Dump Neighbor Table

Cmd: dnt
Args: None

Dump all entries in the neighbor table. This command will scroll through all entries in the neighbor table and dump them out. It's used to check if the neighbor tables are getting updated properly.

Ex: dnt
Shows the current neighbor entries and their fields.

5.1.7 Dump Routing Table

Cmd: drt
Args: None

Dumps all entries in the routing table. Used to check if the routing table is being updated properly and the contents are as expected.

Ex: drt
Shows the current routing entries and their fields.

5.1.8 Dump Binding Table

Cmd: dbt
Args: None

Dumps all entries in the binding table. Used to check if the binding table is being updated properly and binding entries are as expected.

Ex: dbt
Shows the current binding entries.

5.1.9 Dump Group Table

Cmd: dgt

Args: None

Dumps all entries in the group table. Used to check if the group table is being updated properly and the group entries are as expected.

Ex: dgt

Shows the current group entries.

5.1.10 Add Binding Entry

Cmd: abe [src endpoint] [cluster ID] [dest addr] [dest endpoint]

Args:

src endpoint– The source endpoint where the frame originates

cluster ID – The cluster ID that the frame is targeting

dest addr – The destination address that the a frame from the src ep and clust ID should go to

dest endpoint – The destination endpoint on the destination node that the frame should go to

Adds a binding entry to the binding table of the node. If a frame originates from the source endpoint and cluster ID on the node and no destination address is present, the frame will use the destination address, endpoint, and cluster ID from the binding table.

Ex: abe 1 6 0 3

Adds a binding entry to the binding table where the source endpoint is EP 1, the cluster ID is 0x0006, the destination address is the 0x0000 (the coordinator), and the destination endpoint is EP 3. In the case that the endpoint is defined as a Home Automation endpoint, cluster 0x6 is the on/off switch cluster. Hence, any frame originating from the on/off cluster of this endpoint (ie: the switch is turned on) will be forwarded to the coordinator’s endpoint 3.

5.1.11 Remove Binding Entry

Cmd: rbe [src endpoint] [clust ID] [dest addr] [dest endpoint]

Args:

src endpoint– The source endpoint where the frame originates

cluster ID – The cluster ID that the frame is targeting

dest addr – The destination address that the a frame from the src ep and clust ID should go to

dest endpoint – The destination endpoint on the destination node that the frame should go to

Removes a binding entry from the binding table with the given parameters. If none is found, no action will be taken.

Ex: rbe 1 6 0 3
Remove the previously added binding entry whose source endpoint is EP1, cluster ID is 0x0006, destination address is 0x0000 (coordinator), and destination endpoint is EP3.

5.1.12 Add Group Entry

Cmd: ag [group ID] [endpoint]
Args:
group ID – The group ID for this entry.
Endpoint – The endpoint that the frame that contains the group ID will be forwarded to.

Groups are a form of multicasting in Zigbee. Group frames are broadcast and contain a group ID. If a broadcast data frame is received and the group ID contained in the APS header matches an entry in the group table, the frame will be forwarded to the endpoint in the entry. Multiple endpoints can have the same group ID in the case that one node controls multiple identical devices. An example would be a node that controls four identical lights and one non-identical light. A group table entry can be configured for each of the identical endpoints using the same group ID. If a frame arrives with the group ID, ie: to turn off the lights, the command will go to each of the endpoints (lights) that have the same group ID.

Ex: ag 2 1234
This will add a group entry for EP 2 with a group ID of 0x1234. If a broadcast data frame with group ID = 0x1234 arrives, it will be sent to EP 2.

5.1.13 Remove Group Entry

Cmd: rg [group ID] [endpoint]
Args:
group ID – The group ID for the entry to be removed
endpoint – The endpoint for the entry to be removed

The entry with the specified group ID and endpoint will be removed from the group table.

Ex: rg 2 1234

This will removed the group entry for EP 2 with the group ID of 0x1234.

5.2 ZCL Commands

These are commands specific to the ZCL. They usually generate requests to a remote node with a specified destination address and operate on a specific cluster. Normally, you would have to specify the endpoint for a request but in the simulator case, the commands go to a generic endpoint specifically for the ZCL.

5.2.1 Read Attribute

Cmd: ra [dest addr] [cluster ID] [attribute ID 1] …[attribute ID n]
Args:
dest addr – The destination address that the read attribute request frame will go to.
cluster ID – The cluster ID containing the attributes that will be read.
attribute ID – The attribute ID to be read. This field is variable with up to TEST_ZCL_MAX_ATTRIBS that can be read

Generates a ZCL read attribute request frame. The ZCL read attribute request is a general ZCL command that gets processed by the remote node specified by the destination address. The frame will read the attributes specified by the attribute IDs that are contained in the cluster ID. The remote node will return a response frame with the value of each attribute specified in the attribute list.

Ex: ra 0 0 0 1 2
Generate a read attribute request frame with a destination address of 0x0000 (coordinator) and targeted at the cluster with an ID of 0x0000 (Basic cluster). The attribute IDs to be read are 0x0000 (ZCL Version), 0x0001 (Application Version), and 0x0002 (Stack Version).

5.2.2 Write Attribute

Cmd:
wa [dest addr] [cluster ID] [attribute ID 1] [attribute value 1]…[attribute ID n] [attribute Value n]
Args:
dest addr – The destination address the write attribute request frame will go to.
cluster ID – The cluster ID containing the attributes that will be written.
attribute ID n – The attribute ID to be written to.
attribute Value n – The value to be written.

Generates a ZCL write attribute request frame. The ZCL write attribute request is a general ZCL command that gets processed by the remote node with the specified destination address. The command requires the user to specify the cluster ID containing the attributes to be written and a variable length attribute list and the value to be written. Currently, this command is limited to attributes that have a type of unsigned character to simplify this command. This limitation is only for the command line interface and the stack's write attribute request can take multiple data types. The node that processes this command will return a response frame which will inform the user of the status of each write attempt.

Ex: wa 3 6 0 1
Generate a write request frame with a destination address of 0x0003 and a cluster ID of 0x0006 (on/off cluster). The attribute ID is 0x0000 (on/off attribute) and the value is 0x1. This would essentially manually write the value 1 to the on/off cluster's on/off attribute, however this is not recommended. It's best to modify the attribute via the cluster's specific commands rather than using the general write attribute request, since this can lead to a loss of synchronization.

5.2.3 Discover Attributes

Cmd: da [dest addr] [cluster ID] [attribute start ID] [max attributes]
Args:
dest addr – The destination address this request will be sent to.
cluster ID – The cluster ID containing the attributes to be discovered.
attribute start ID – The attribute ID of the starting attribute where the discovery will begin.
max attributes – The maximum amount of attributes to be discovered.

The discover attributes command generates a discover attributes request frame to the remote node with the specified destination address. Discover attributes is used when the attributes supported by a remote node's cluster is unknown. In that case, you can target the remote node's cluster ID and do a general sweep of the attributes. The start ID is used to specify the starting attribute where the sweep will begin and the max attributes is the maximum number of attributes to sweep. The remote node will process this frame and return an attribute ID and attribute data type for each of the attributes in the sweep range that exist. Hence, even if you sweep attributes 0 through 50, you may only find that two attributes are returned in the response frame. That means the remote node only supports two attributes between 0 and 50.

Ex: da 0 0 0 20
Generate a discover attributes request frame with a destination address of 0x0000 (coordinator)

and targeted at the cluster with an ID of 0x0000 (Basic cluster). The starting attribute ID is 0x0000 and the maximum number of attributes to sweep is 20. This will start from attribute 0 and sweep to address 19. If any attribute within the ID range exists, it will be returned in the remote node's discover attributes response frame.

5.2.4 Configure Report

Cmd: cr [dest addr] [cluster ID] [attribute ID] [interval]
Args:
dest addr – The destination address the frame will be sent to.
cluster ID – The cluster ID that is being targeted.
attribute ID – The attribute ID that will be reporting.
interval – The periodic time interval for each report in seconds.

This command will generate a configure report request frame to a remote node. There are some cases where you want to monitor a certain attribute value such as electricity usage or whether a switch is on or off. In that case, you can use a configure report request to send out a periodic report frame with the attribute's value. The cluster ID and the attribute ID specify what gets reported.

Ex: cr 2 6 0 9
Generates a configure report request frame to destination address 0x0002. The cluster ID is 0x0006 (on/off switch), the attribute ID is 0x0000 (on/off attribute), and the interval is 9 seconds. A report frame will be generated and sent to the node that originated the request every 9 seconds with the status of the on/off attribute.

5.2.5 Identify

Cmd: id [dest addr] [timeout]
Args:
dest addr – The destination address the frame will be sent to
timeout – The time that the identify action will be active in seconds

Identify is used to identify a particular node. When the identify request is sent to a node, the node will perform some type of action to identify itself. Normally, it will probably take the form of lighting or flashing an LED for the timeout interval. This command is particularly useful to group multiple nodes together using the "group if identify" command. Although this command is not implemented yet, the "group if identify" command will take all nodes that are currently identifying themselves

and add them to the same group. They can then be addressed using group transmissions.

Ex: id 2 10
This will generate an identify request to the node with an address of 0x0002. The identify action will remain active for 10 seconds.

5.2.6 On/Off

Cmd: onoff [dest addr] [command]
Args:
dest addr – The destination address the frame will be sent to.
command – There are three commands in the ZCL on/off cluster. 0 = off, 1 = on, 2 = toggle.

This command is specific to the ZCL's on/off cluster. This generates an on/off request frame to the node with the specified destination address. The request will contain the command that gets interpreted by the on/off cluster.

Ex: onoff 1 0
Change the attribute at the remote node with an address of 0x0001 to off.

Ex: onoff 4 2
Toggle the attribute at the remote node with an address of 0x0004.

5.2.7 Move to Level

Cmd: movt [dest addr] [level] [transition time]
Args:
dest addr – The destination address the frame will be sent to.
level – This is the desired level that you want the attribute to move to. The level is from 0x00 to 0xff. This value is in decimal.
transition time – This is the time you want to take for the transition in increments of 100 msec. This value is in decimal.

This command is specific to the ZCL's level control cluster. It's most applicable to dimming applications, however it can be used for any device that has a varying range of control rather than a binary on/off control type. Another example would be a volume control. When invoked, this command will generate a "move to level" request that gets sent to the remote node with the specified destination address. The level can vary from 0x00 to 0xff and the transition time is

counted in tenths of a second or 100 msec intervals.

Ex: movt 3 255 100
This command will send a move to level request to the node with an address of 0x0003. When received, the node should transition from its current level which is stored in the attribute, to the level 255. Regardless of the size of the move, the transition time will take ten seconds.

5.2.8 Level Move

Cmd: move [dest addr] [dir] [rate]
Args:
dest addr – The destination address the frame will be sent to.
dir – The direction of the move. 0 = down, 1 = up.
rate – The desired rate that the move should occur in. The rate is specified by increments per second. This number is in decimal.

This command is also specific to the ZCL's level cluster. The move command will instruct the level cluster to increment/decrement the level attribute at a specified rate. There is no max or min limit other than 0 or 0xff which are the limits of the attribute. The move will automatically stop when the limit has been reached or a stop request is issued. The rate is specified in increments per second.

Ex: move 0 1 10
This will generate a move request to node 0x0000 (coordinator). When received, the node's level cluster will increment the level attribute at a rate of 1 increment per 100 msec. It will continue incrementing until a stop command is received.

5.2.9 Level Step

Cmd: step [dest addr] [dir] [step size] [trans time]
Args:
dest addr – The destination address the frame will be sent to.
dir – The direction to step in. 0 = down, 1 = up.
step size – The size of the step. This is limited to a max size of 255. This value is in decimal.
trans time – The transition time of the step. This is the amount of time to take to implement the step in 100 msec increments. This value is in decimal.

The step command is specific to the ZCL's level cluster. It differs from the move to level command in that it implements a fixed size step, whereas the move to level will move the attribute to a

specific value, regardless of the size of the step. The step size is the amount of increments to move in and the direction specifies whether to increment or decrement the level. The transition time is to specify how long for the transition to occur.

Ex: step 1 100 100
This generates a step request to the node with an address of 0x0001. The step size is 100 increments and the transition time is 10 seconds. Regardless of the current level value, it will move 100 increments or it will stop when the attribute limits are reached.

5.2.10 Level Stop

Cmd: stop [dest addr]
Args:
dest addr – The destination address the frame will be sent to.

This command is specific to the ZCL's level cluster. It will stop all activity that is currently going on in the level cluster, purge all transitions that are in progress, and freeze the level attribute at its current value.

Ex: stop 1
Stop all level transitions on node 0x0001.

5.3 ZDO Commands

These commands are specific to the ZDO and control the different functions inside of it. They also generate requests to a remote node.

5.3.1 Network Address Request

Cmd: nar [extended addr]
Args:
extended addr – The extended address of the device from which you'd like to get the network address

This is one of the ZDO discovery commands and allows you to request the network address of a device, given it's extended address. This command is broadcasted over the network and the device to whom the extended address belongs to will respond with its' network address.

Ex: nar 0011223344556677

This would send out a ZDO network address request as a broadcast over the network. If a device has this address, it will reply with it's network address.

5.3.2 IEEE Address Request

Cmd: iar [nwk addr]
Args:
nwk addr – The network address of the device you want the IEEE address of

This request is broadcast to the network in order to determine the IEEE address of a device that you know the network address of.

Ex: iar 2
This would send out a ZDO IEEE address request as a broadcast frame over the network. If a device has this address, it will reply with it's IEEE address.

5.3.3 End Device Bind

Cmd: edb [num in clusters] [in cluster 1..n] [num out clusters] [out cluster 1..n]
Args:
num in clusters – The number of in clusters (server clusters) supported by the endpoint. This number is in decimal.
in cluster n – Variable list of in clusters. The length of this list depends on the number of clusters you specified in num in clusters. Each cluster should be separated by a space.
num out clusters – The number of out clusters (client clusters) supported by the endpoint. This number is in decimal.
out cluster n – Variable list of out clusters. The length of this list depends on the number of clusters you specified in num out clusters. Each cluster should be separated by a space.

The end device bind is a fairly complex command because the edb sequence is also complicated. It's an automatic way to bind two devices together via the coordinator. You would need to have a coordinator to start the network, then create two additional devices that you want bound together. When calling the command, you have to specify the in clusters and out clusters that you want bound. The coordinator will handle the processing and sending out the individual bind request for any clusters that match up together, ie: the cluster ID matches and an in cluster can pair up with an out cluster.

The command requires using the edb command on two nodes within a time interval of 30 seconds.

This is the end device bind timeout that is hardwired in the stack. When one end device bind request is received by the coordinator, it will wait up to 30 seconds for the second one to come in indicating the second device to be paired. If no second request comes in, the end device bind sequence gets cancelled. The timeout interval can be changed via the “ED_BIND_TIMEOUT” definition in zdo_bind_mgr.c.

Ex: Node 1: edb 3 1 2 3 2 1 2
This generates an end device bind request from node 1 to the coordinator. The number of in clusters is 3 and the cluster IDs are 0x1, 0x2, and 0x3. The number of out clusters is 2 and the cluster IDs are 0x1 and 0x2.

Ex: Node 2: edb 2 1 2 3 1 2 3
This generates an end device bind request from node 2 to the coordinator. The number of in clusters is 2 and the cluster IDs are 0x1 and 0x2. The number of out clusters is 3 and the cluster IDs are 0x1, 0x2, and 0x3.
When both are received, the coordinator will check the in cluster list of one device with the out cluster list of the other device and match up identical cluster IDs. For every matched cluster, the coordinator will send a bind request to both devices.

5.3.4 LQI Request

Cmd: znl [dest addr]
Args:
dest addr – The destination address of the node you want the request to go to.

The LQI request actually is used to read the neighbor table of a remote node. This is particularly useful to find out how many children the node has, how many neighbors, the parent, etc…This command would be particularly useful to develop a network hierarchy graph or to maintain some type of manually maintained source routing table in a particular node, ie: a PC. The acronym LQI actually stands for Link Quality Indicator and you can also get the LQI of each neighbor with reference to the target node from this command.

Ex: znl 3
This sends an LQI request to the node with an address of 0x3. When received, the node will return a response with all the neighbors in its neighbor table up to a maximum limit. The maximum limit is imposed so that the payload length won’t exceed the allotted payload size and overflow the frame.

5.3.5 Network Leave Request

Cmd: lv [dest addr] [opt: remove children]

Args:

dest addr – The destination address of the node you want the request to go to.

remove children – This is an optional command to request the node to also inform all its children to leave the network.

This command is slightly similar to the ZDO leave request. The main difference is that the ZDO leave request is used to request a remote node to generate a leave request command to another node. This command generates the leave request directly from the node that the command acts on. It generates a leave command by accessing the nwk_leave function directly and is useful for testing that the leave request is generated properly and is processed correctly by the remote node.

5.3.6 ZDO Leave Request

Cmd: nlr [dest addr] [leave addr] [remove children]

Args:

dest addr – This is the destination address of the node that the request will go to.

leave addr – This is the address of the node that is being requested to leave the network.

remove children – This is a Boolean command (1 = remove children) to indicate if the children should also be requested to leave.

5.3.7 Network Discovery Request

Cmd: ndr [dest addr]

Args:

dest addr – The destination address of the node you want the request to go to.

When received by the destination node, the dest node will perform a network scan. It will then report back all the networks found on each channel up to a specified maximum. The maximum limit is imposed to protect against overflowing the payload.

Ex: ndr 1

Send a network discovery request to node 0x1. When received, the node will perform an active scan, collect the beacons from the different devices, and report them back in the response.

5.3.8 Permit Join Request

Cmd: npj [dest addr] [duration]

Args:

dest addr – The destination address of the node you want the request to go to.

duration – The interval that devices will be allowed to join in seconds. This value is in decimal.

The node that receives this request will enable devices to join for an interval specified in the request. When received, the permit join flag in the network layer will be changed to true and a timer is enabled that counts down the duration. When the timer expires, the permit join flag is changed to false. A value of 255 will enable the permit join to stay on indefinitely.

Ex: npj 3 10

Request node 0x3 to enable its permit join flag for 10 seconds.

5.4 Raw Data Commands

These commands are specific to the manufacturer specific data endpoint. They're mostly used to test the data path and various features of the data path. They're also a useful way to send and receive raw data without going through the complexity of the ZCL and creating special clusters and cluster handlers.

5.4.1 Unicast Data Request

Cmd: udr [dest addr] [data 0..n]

Args:

dest addr – The destination address of the data to be sent

data n – The data to be sent. All data bytes should be separated by a space/

The unicast data request is just the standard data request and can be used to send raw data in a byte array. It accesses the stack's data path and uses MAC level acknowledge only.

Ex: udr 1 11 22 33 44 55

Send a data frame to node 0x1 with data contents: 0x11 0x22 0x33 0x44 0x55

5.4.2 Reliable Unicast Data Request

Cmd: rudr [dest addr] [data 0..n]

Args:

dest addr – The destination address of the data to be sent.

data n – The data to be sent. All data bytes should be separated by a space.

The reliable unicast data request is the same as the unicast data request but uses APS level acknowledge as well.

Ex: rudr 2 66 55 22 11
Send a data frame to node 0x2 using APS acknowledgement as well as MAC level acknowledgement. The data to be sent is 0x66 0x55 0x22 0x11.

5.4.3 Indirect Data Request

Cmd: idr [data 0..n]
Args:
data n – The data to be sent. All data bytes should be separated by a space.

The indirect data request sends data via the binding entry. This command cannot be used unless a binding table entry has been set up for this endpoint, cluster, destination address, and destination endpoint. The endpoint number and cluster ID is located in the following definitions: TEST_DATA_EP and TEST_DATA_IN_CLUST. This command is useful for testing out binding entries and the binding functionality.

Ex: idr 22 33 44 55
This command would send the data 0x22 0x33 0x44 0x55 to the destination address, endpoint, and cluster found in the binding entry attached to this endpoint/cluster combo. If no entry exists, the frame will be discarded.

5.4.4 Group Data Request

Cmd: gdr [group ID] [data 0..n]
Args:
group ID – The group ID that this data belongs to.
data n – The data to be sent. All data bytes should be separated by a space.

This sends the data in a broadcast frame with the specified group ID. When received by other nodes, they will check the group table to see if there is any match to the frame's group ID. If there is a match, the frame will get forwarded to the endpoint specified in the group table.