zbcli Quick Start User Guide

Chris Brandson, Dana Sy

Contents:

Basic Operation

Quick start guide on how to use the ZigBee command line interface (zbcli) and how to form or join a simple network.

Starting zbcli

On a BeagleBone device

First you need to ssh into the device. The address is static, usually printed on the side of the device, and usually starts with `192.168.x.x` The default username is `debian` and the password is `temppwd` for the device. Once you are logged in to the device simply type `zbcli` to start an instance of zbcli.

The radios are located on `/dev` directory. The location of a 2.4GHz radio will be `/dev/q7x_2400` and the location of a sub-GHz radio will be `/dev/q7x_subghz`. Alternatively the device can also be found on the `/dev/serial/by-id/` directory.

On a Linux notebook or desktop

Simply execute the zbcli executable.

The radio on a linux machine is usually registered as a `ttyACM` or `ttyUSB` device. The user needs to make sure they're part of the dialout group to properly interact with the devices. Alternatively the device can also be found on the `/dev/serial/by-id/` directory.

Using zbcli

Initialize the stack

Using zbcli you create an instance of the ZigBee stack and interact with it using a series of commands. A typical sequence to start and instance is:

```
# specify the radio location
wpan serial /dev/q7x_2400
# initialize the stack
init
```

NOTE: Lines starting with # indicates that they are comments

Forming a network

Forming a network is done by the following sequence of commands:

```
# specify the radio location
wpan serial /dev/q7x_2400
# initialize the stack
init
# use channel 11 in channel page 0
startup_channellist 0 11
# form (coordinator)
startup_form
```

NOTE: If the network needs to support pre-Z3 BDB network the coordinator needs to set the BIB value to disable automatic TCLK update before forming the network. Run the command below first before forming the network:

```
set_ib bdbTrustCenterRequiresKeyExchange 0
```

Joining a network

Joining a network is done by the following sequence of commands:

```
# specify the radio location
wpan serial /dev/q7x_2400
# initialize the stack
# use channel 11 in channel page 0
startup_channellist 0 11
# form (coordinator)
startup_join
```

Checking the status of a node

You can check the status of a node by typing in the *status* command on zbcli. It will print out various information about the node such as it's IEEE address, children, channel mask and other various information.

Send an IEEE Address Request

An IEEE address request can be sent to see ZDO packets over the air. You just need to specify the destination address and the network address of interest. An example is:

```
# send an IEEE address request to short address 0x1234
# with a network address of interest of 0x5678
zdo_ieee_addr_req 0x1234 0x5678
```

Help with zbcli

The user can type `help` at any time to get help or syntax information about a specific command. For example typing `help zdo_ieee_addr_req` will print out:

```
zbcli[14864]: 000000.000 usage: zdo_ieee_addr_req DSTADDR NWKADDR [START_INDEX]
zbcli[14864]: 000000.000
zbcli[14864]: 000000.000 Send a ZDO IEEE_Addr_req to a remote device
zbcli[14864]: 000000.000
zbcli[14864]: 000000.000 positional arguments:
zbcli[14864]: 000000.000     DSTADDR         Destination address for the command
zbcli[14864]: 000000.000     NWKADDR         Network address of interest
zbcli[14864]: 000000.000     START_INDEX     Start index for extended request types
zbcli[14864]: 000000.000
zbcli[14864]: 000000.000 optional arguments:
zbcli[14864]: 000000.000     --help          Display this message and exit
```

Typing `help -l` on zbcli will print out a list of commands and a short description.

Adding Command Line History with rlwrap

By itself zbcli does not provide command line history. However, on POSIX operating systems such as Debian Linux, the command line utility is very useful for providing command line editing and history.

To use rlwrap, invoke zbcli througth rlwrap using

```
rlwrap ./zbcli
```

Demonstrating and Testing OTA using zbcli

Server

zbcli implements simple OTA server functionality. It is capable of serving only a single file. The file must conform to the OTA file specification, the contents of the header is used to define the image and respond to client requests. A separate utility, ota-image-gen is an example of how an OTA compatible file is created. After forming or joining the network, OTA Server functionality is enabled using the following sequence of commands in zbcli:

```
# create OTA server cluster on endpoint 0x0b
zcl_ota_server --init --endpoint 0x0b

# create OTA server cluster on endpoint 0x0
# with current time 0 and update time 100
zcl_ota_server --init --endpoint 0x0b --end_current_time 0 --end_upgrade_time 100

# load OTA image FILE making it available to clients
zcl_ota_server --register FILE
```

After a file is registered it can be useful to show the particulars, stored in the OTA header of the file, this can be done with the show command:

```
# show info from FILE used in query: mfg code, file version, image type
zcl_ota_server --show
```

To send an image notify command to the client you always need to specify the query jitter, destination address and endpoint. There are 3 optional fields for the image notify payload which modify the payload type flag to tell the client which fields are present in the payload. The payload type are as follow and zbcli populates the field as needed:

```
# send an image notify with:
# query jitter of 10
# manufacturer code of 0x10d7 (Exegin)
# image type of 0xffbf
# file version of 0x1.
zcl_ota_server --image_notify --nwk_addr 0x0 --endpoint 12 --query_jitter 10
--manufacturer_code 0x10d7 --image_type 0xffbf --file_version 0x1
```

Payload Type Values	Description
0x00	Query Jitter
0x01	Query Jitter, Manufacturer Code
0x02	Query Jitter, Manufacturer Code, Image Type
0x03	Query Jitter, Manufacturer Code, Image Type, File Version

Client

In zbcli client functionality is available to create the client endpoint, discover the OTA server, query for a specific image, and initiate the transfer. When the endpoint is created an output filename must be provided, if the transfer is successful this file will contain the image transferred from the server:

```
# create the OTA client cluster and specify the filename of the transferred file
zcl_ota_client --init -f received --endpoint 0x0c --client_image_block_delay 100
#
# the server must be discovered before any further operation is performed
# (assume the coordinator)
```

```
# if the IEEE address is known it can be specified with --upgrade_server (recommended)
# or if the short address is know it can be specified with --discovery_address
zcl_ota_client --discover
#
# check the server to see if it has an image of the type we want
zcl_ota_client --query_next_image --manufacturer_code 0x10d7 --image_type 0xffbf
--file_version 0x00000001 --wait
#
# start the transfer
zcl_ota_client --image_block_start
```

One the transfer begins it proceeds automatically until either the image is successfully transferred (and a file of provided filename will contain the image) or an error occurs. Diagnostic information is displayed during the transfer and if everything is successful the message "Transfer control to BOOTLOADER!" is displayed. In zbcli the transfer does not actually occur, but in a real system this is where it would happen.

Keys

These are commands that deals with keys for ZigBee.

Send a New Transport Key and Switch Key

To change keys while the nodes are operational on a network:

```
aps_transport_key 1111:2222:...:8888 <target EUI> 2
aps_transport_key 1111:2222:...:8888 <your own EUI> 2
aps_switch_key short 0xffff 2
```

Note: The order is important otherwise the target will not be able to send the transport key.

Note: If do not send the transport key to yourself you will get a security failure.

Note: The key id must match for all 3 commands and needs to be greater than the current key id.

BDB TC Link Key Exchange Procedure

Changing the BIB value of the bdbTrustCenterRequiresKeyExchange parameter will change the behaviour of the automatic trust centre link key exchange that was required for BDB. Disabling it will make the joiner not perform an automatic trust centre link key exchange *(used to support pre-Z3 BDB networks)*:

```
# To disable the Trust Centre Link Key Exhange
set_ib bdbTrustCenterRequiresKeyExchange 0

# To enable the Trust Centre Link Key Exhange (enabled by default)
set_ib bdbTrustCenterRequiresKeyExchange 1
```

Printing the network key

During or after the test, use the following command to get the key for current network:

```
nwk_key list
```

This will be very helpful with Touchlink since wireshark doesn't automatically decrypt the network key.

End Device

An end device is a reduced function device.

NOTE: We currently only support the end device keep alive mechanism for our routers and coordinator. The default for zbcli is a full function device (Coordinator / Router). To initialize an end device the following command needs to be entered first before joining the network:

```
startup_config capability fullfunction disable
```

If the endnode is also a sleepy endnode:

```
startup_config capability rxonwhenidle disable
```

Sleepy End-Devices

Configuring a sleepy end device:

```
# IEEE 802.15.4 RFD, reduced function device or ZigBee End Device
startup_config capability fullfunction disable
# For sleepy end-device, advertise that our receiver is off when idle
startup_config capability rxonwhenidle disable
```

Non-sleepy End-Device

Configuring a non-sleepy end device:

```
# IEEE 802.15.4 RFD, reduced function device or ZigBee End Device
startup_config capability fullfunction disable
# For non-sleepy end-device, advertise that our receiver is on when idle
startup_config capability rxonwhenidle enable
```

MAC Data Poll Period

You can configure the stack to automatically poll its parent using the NWK Slow Poll mechanism:

```
# Configure a 2 second poll
set_ib nwkSlowPollPeriod 2000

# Disabling slow polling
set_ib nwkSlowPollPeriod 0
```

Setting the End Device KeepAlive

You can configure the end device timeout time by using the following command:

```
# Set the timeout time to 2 minutes
startup_config end_timeout 1
```

The table below is the valid range for the timeout enumeration and it's associated actual timeout value.

Recipes

These are commands that are tricks to change various parameters with zbcli.

Performing a TC (Trust Center) Rejoin

Used to be known as insecure rejoin, renamed TC rejoin:

```
startup_tc_rejoin
```

Creating a Distributed Network

There is no coordinators in a distributed network only routers. In order to create a distributed network, after *startup_ config prodefaults*:

```
startup_config security trustCenterAddress 0xffffffffffffffff
startup_config security preconfiguredLinkKey d0:d1:d2:d3:d4:d5:d6:d7:d8:d9:da:db:dc:dd:de:df
```

Using Persistence

After forming or joining the network (e.g. startup_form or startup_join), enable persistence:

```
startup_persist enable [FILE]
```

You can then shutdown zbcli, restart and reload the persistence with:

```
startup_persist startup [FILE]
```

Disable BDB Reset PJOIN

bdbTestFlags which when set will NOT send the BROADCAST ZbZdoPermitJoinReq() on either the Coord or joiner/ED:

```
set_ib bdbTestFlags 1
```

Setting the Time Between Link Power Delta Command on subGHz

Set the NIB in seconds:

```
set_ib nwkLinkPowerDelta 16
```

Configure PRO Default Settings

A quick way to configure and view the PRO Defaults is:

```
startup_config prodefaults
```

Testing Fragmented Transmission

In order to tickle the stack to magically drop block 1 during transmission use the command:

```
aps_frag tx_drop_add 1
```

Important Note: apsMaxWindowSize MUST be the same on all devices on the network otherwise the re-transmission will not be successful! The test cases call this out as 3, but it MUST be uniform.

Logging

This will make zbcli start logging to a file:

```
# start logging to file 13_02_DUT_NXP_Atmel_gZC
log -f 13_02_DUT_NXP_Atmel_gZC
```

Test Profile #2 (zcl_tp2)

The test profile #2 (tp2) endpoints is always used for interopt testing. It's a basic ZCL message between devices and it usually used to test that the nodes can communicate with each other. Create the Test Profile #2 endpoints:

```
zcl_tp2 init
```

Send a Buffer Test Request:

```
zcl_tp2 buffer DSTMODE DSTADDR LENGTH [PROFILE] [--aps_sec]
```

Send a Counted Packet:

```
zcl_tp2 counted NWKADDR LENGTH NUM_PKTS DELAY_MS [OPTIONS]
```

Reset the counted packet count on a device:

```
zcl_tp2 reset NWKADDR
```

Retrieve the counted packet count on a device:

```
zcl_tp2 retrieve NWKADDR
```

Changing stack table sizes

NOTE: This command needs to be executed before running the init command:

```
init_tbl_size -n <max # of neighbours> -a <max # of address map> -k <max # of key table>
```

There is also another option "-c" which will set all table sizes to 200.

All parameters are optional and if nothing is entered the default will be used. (Defaults: -n 64; -a 32; -k 32)

Transmit Power Level

Upper limit +10, Lower limit -20:

```
# disable power control on the GB-868 interfaces (if necessary)
startup_config power_mgmt off
# Start the stack, join a network...
# Increase / decrease TX power level:
nwk_txpower +5 to increase the attenuation level
nwk_txpower -5 to decrease the attenuation level
```