DFCM-NNN40-

BLE/WIFI Command Line Interface Document

Index:

Revision History

Version	Date	Reason of change	Maker
1.0	Jan 11, 2015	Initial release	Gill Wei
1.1	Mar 2, 2015	Add Wi-Fi Command Set	Marcus Chiou
1.2	Mar 10, 2015	Add BLE profile overview to introduction and BLE example for creating the sample by command	Gill Wei
1.3	Mar 27, 2015	Modify WIFI Command description	Marcus Chiou
1.4	Mar 30, 2015	Able to add BLE GATT 128bit vendor specific services/characteristics, modify advertise and add service command format.	Gill Wei
1.5	Jun 11, 2015	Modify 5.2 Procedure for creating an Profile about command description	Gill Wei
1.6	Sep 4, 2015	Merge mbed (mbed,BLE_API, RF51822) library, modify BLE name command, Wifi TCP/UDP blocking description	Gill Wei
1.7	Dec 9, 2015	Update GPIO PIN define to EVB version 3 Add BLE central mode API, WIFI device API, simplified command set.	Gill Wei
1.8	Jan 25, 2016	Add BLE data interrupt command; Chap 6.5 Wi-Fi data receive interrupt	Gill Wei
1.9	Jan 26, 2016	Remove Chap 6.5 Wi-Fi data receive interrupt, add Wifi data interrupt command; Modify Chap6 Wifi example program; Modify Chap5.3 BLE example procedure	Gill Wei
1.10	Feb 18, 2016	Modify BLE name, Wifi TCP send, UDP send format	Gill Wei
1.11	Feb 25, 2016	Modify TCP and UDP max data length	Gill Wei
1.12	April 25, 2016	Modify BLE Update command(add raw data type), Fix update	Gill Wei

		and readData command value length issue	
1.13	April 25, 2016	Fix Characteristic property setting issue(Doc. Unchanged)	

Wireless LAN/Bluetooth Low Energy Combo Module Command Line Interface

A Command Line Interface (CLI) used to implement function of Wi-Fi/BLE Combo Module.

1. Introduction

This document describes the function of each command within DFCM-NNN40-DT0R command line interface; include BLE connection, Wi-Fi enable/disable, sleep mode, and so on. This document also makes a demonstration of how to set up environment in PC and create a BLE profile in GATT server; below we use Glucose Profile setting as example, and list the typical procedure of standard profile setting.

Glucose profile is one of the officially defined BLE profile by Bluetooth official group 「SIG」, which is abbreviated of Special Interest Group. See Reference 1.

Profile defines and explain stored data format, one profile contains one or several services, each service contains data for communication.

Glucose profile composed of Glucose Service and Device Information Service(see Figure.1), which with specific data format, for example, the Glucose Service contains data as Time, Sequence number, Glucose concentration or other data related to Glucose measurement.

In this case, NNN40 with Glucose sensor can be seem as 「Glucose Sensor」 (see Figure.1), the Central Device, like smart phone or laptop, can act as 「Glucose Collector」, which collect and display the glucose data transmitted from glucose sensor. Collector and sensor can also be treated as 「Central」 and 「Peripheral」, Reference 2 have BLE GATT Service overview and have explain more detail.

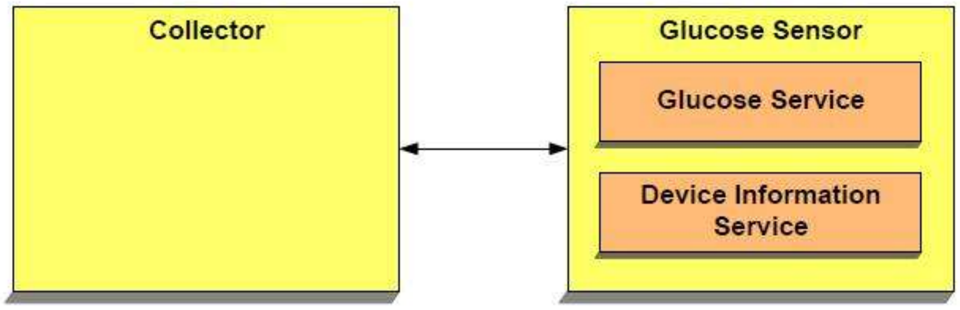


Figure.1 Glucose role and services

After define all the data in Glucose sensor, let ˛ s see how BLE device set up the connection and transmit data, Figure.2 show how BLE device in and out from different state; only in connecting state, sensor data will transfer in profile-defined format.

Reference

1. Glucose Profile
https://developer.bluetooth.org/gatt/services/Pages/ServiceViewer.aspx?u=org.bluetooth.service.glucose.xml

2. SIG BLE GATT Service List
https://developer.bluetooth.org/gatt/services/Pages/ServicesHome.aspx

2. Commands Status Responses

Whenever a command is received by device, an 「OK」 or 「ERROR」 response will be send back to host side. ERROR case will output with specific error string; OK will output with additional response for some specific commands.

2.1 OK

OK

Function: All successful commands will respond with OK message

Example:

```
RESPONSE: OK<cr_lf>
            <cr_lf>
```

2.2 ERROR

ERROR

Function: All failure commands will respond with ERROR response with mapping error message

Command Format: ERROR;<error message>;

Response Values:

Error Message:

"No such command;",

"Wrong number of arguments;",

"Argument out of range;",

"Argument syntax error;",

“No matched argument;”,

"Wrong command order;",

“Invalid state to perform operation;”,

“Function call fail;”

Example:

RESPONSE: ERROR; Argument out of range;<cr_lf>
<cr_lf>

3. BLE Commands Description

This section describes the detail function of each command, including command format, argument format, and command description.

3.1 GPIO

CONTROL GPIO PIN

Function: Set or Clear a specific GPIO pin on BLE module

Command Format: cynb gpio <GPIO-NO> <SET/CLEAR>

Example:

```
COMMAND: cynb gpio 25 set<cr>
RESPONSE: <cr_lf>
                OK<cr_lf>
               <cr_lf>
```

Note: Number of available pins are 0, 4, 5, 6, 13, 21, 23, 24, 25, 29, 30, 31
Currently PIN23、PIN25 set as default UART communication pin, can't be clear unless changed.
*PIN 4,5,6,13 correspond to LED 6,10,9,7.

3.2 System Off

SLEEP MODE

Function: Go to system off mode, and then choose a specific pin; detect pin state change for wake up.

Command Format: cynb sleep <GPIO-NO>

Example:

```
    COMMAND: cynb sleep 23<cr>
    RESPONSE: <cr_lf>
              OK<cr_lf>
              <cr_lf>
```

Note: When the state of pin is changed, module is wake up and reset, Number of available pins are 0, 4, 5, 6, 13, 21, 23, 24, 25, 29, 30, 31
Default set pin is UART RX pin(23). That is, if not configure pin number, wake up pin is PIN23. Once UART received interrupt, module will reset.

3.3 Reset

RESET

Function: Reset BLE module.

Command Format: cynb reset

Example:

```
    COMMAND: cynb reset<cr>
    RESPONSE: <cr_lf>
              OK<cr_lf>
              <cr_lf>
```

3.4 System Information

MODULE INFORMATION

Function: List system information stored in module, which included Firmware version and Module name

Command Format: cynb info

Example:

```
COMMAND: cynb info<cr>
RESPONSE: <cr_lf>
                    OK; DELTA_CLI_V1.7;DFCM-NNN40-DT0R; <cr_lf>
                    <cr_lf>
```

3.5 Device Name

SET BLE DEVICE NAME

Function: Set friendly name for BLE module.

Command Format: cynb name <LENGTH> <NAME>

Example:

```
COMMAND: cynb name 8 DELTA CLI<cr>
RESPONSE: <cr_lf>
                    OK;DELTA CLI <cr_lf>
                    <cr_lf>
```

Note: Please type string length including “space”. Whenever initialize BLE module, default module name is “nRF5x”.

GET BLE DEVICE NAME

Function: Get the current device name for BLE module.

Command Format: cynb name

Example:

```
COMMAND: cynb name<cr>
RESPONSE: <cr_lf>
               OK; nRF5x;<cr_lf>
               <cr_lf>
```

Note: Default name is "nRF5x"

3.6 BLE Start Advertising

START DEVICE ADVERTISING

Function: Start BLE advertising with specific timeout and interval.

Command Format: cynb advStart <INTERVAL> <TIMEOUT>

Example:

```
COMMAND: cynb advStart 64 180<cr>
RESPONSE: <cr_lf>
               OK<cr_lf>
               <cr_lf>
```

Note: If the system is already in advertising state, error message show that the system in invalid state for operation. The unit of INTERVAL is minisecond; The unit of TIMEOUT is second, 180 means that BLE advertising will stop after 180 seconds.
Default interval value is 64 ms, default time-out value is 180 seconds.
INTERVAL range: Maximum: 10240 Minimum:20
TIMEOUT range: Maximum: 16383 Minimum:1

3.7 BLE Stop Advertising

STOP DEVICE ADVERTISING

Function: Stop BLE advertising.

Command Format: cynb advStop

Example:

```
    COMMAND: cynb advStop<cr>
    RESPONSE: <cr_lf>
              OK<cr_lf>
              <cr_lf>
```

Note: If the system is not in advertising state, error message show that the system in invalid state for operation.

3.8 TX Power

RF TX POWER

Function: Set RF TX power for BLE module.

Command Format: cynb txPow <TX POWER>

Example:

```
    COMMAND: cynb txPow 0<cr>
    RESPONSE: <cr_lf>
              OK;<cr_lf>
              <cr_lf>
```

Note: Available TX power are -30, -20, -16, -12, -8, -4, 0, 4. (unit: dBm)

3.9 BLE Address

SET BLE Address

Function: Set BLE MAC address at run time.

Command Format: cynb bleAddr <BLE ADDR>

Example:

```
        COMMAND: cynb bleAddr 0xE6BCA12B322F<cr>
        RESPONSE: <cr_lf>
                             OK<cr_lf>
                             <cr_lf>
```

Note: BLE MAC address should be 12 hex numbers.

GET BLE ADDRESS

Function: Get current BLE MAC address.

Command Format: cynb bleAddr

Example:

```
        COMMAND: cynb bleAddr<cr>
        RESPONSE: <cr_lf>
                             OK;[ E6 BC A1 2B 32 2F];<cr_lf>
                             <cr_lf>
```

3.10 BLE GATT Service

GATT SERVICE SETTING

Function: Add SIG defined or Vendor Specific service to server

Command Format: cynb gattService <SERVICE UUID>

Example:

```
COMMAND: cynb gattService 0x180A<cr>
RESPONSE: <cr_lf>
              OK<cr_lf>
              <cr_lf>
```

Note: Once one service had added on GATT server, characteristics can be added on, but **once registered new service, user can not add new characteristic in previous set service**. Service UUID can be 4 or 32 hex numbers.

GATT CHARACTERISTIC SETTING

Function: Add new characteristic on currently add services, including set characteristic UUID, characteristic property and value for BLE module.

Command Format: cynb gattChar <CHAR UUID> <ATTR PROP> <ATTR VALUE>

Example:

```
COMMAND: cynb gattChar 0x2A19 0xFF 0x1234<cr>
RESPONSE: <cr_lf>
              OK<cr_lf>
              <cr_lf>
```

Note: The characteristic property currently support multiple properties, including

NONE	= 0x00,
BROADCAST	= 0x01,
READ	= 0x02,

```
WRITE_WITHOUT_RESPONSE         = 0x04,
WRITE                          = 0x08,
NOTIFY                         = 0x10,
INDICATE                       = 0x20,
AUTHENTICATED_SIGNED_WRITES = 0x40,
EXTENDED_PROPERTIES            = 0x80
```

Select the appropriate and mandatory properties to specific characteristic, using bit mask, for example, if user want to add one characteristic with Notify and Read property, filled with 0x12.

The total value field length is 10 bytes, equal to 20 hex numbers.

Attribute value type is uint8_t, so the input hex number must be even number, ex. 0, 2, 4, etc.

Characteristic UUID can be 4 or 32 hex numbers.

Cannot add characteristic before not add any service, or error message will show: "Invalid state to perform operation."

REGISTER GATT SERVICE

Function: Register SIG defined or Vendor Specific service to server
Command Format: cynb regService

Example:

```
COMMAND: cynb regService<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
```

3.11 GATT Update Characteristic Value

UPDATE CHARACTERISTIC VALUE

Function: Change characteristic value and make an indication or notification according to the property of characteristic.

Command Format: cynb update <SERVICE UUID> <CHAR UUID> <TYPE> <VALUE>

Example (1):

```
COMMAND: cynb update 0x180D 0x2A39 0 1234<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
```

RESULT: This example update data to 0x31323334, data length have 4 bytes

Example (2):

```
COMMAND: cynb update 0x180D 0x2A39 1 0x1234<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
```

RESULT: This example update data to 0x1234, data length have 2 bytes

Note:

<TYPE> parameter have following values:

0 : Raw Data (Not transferred from ASCII code)

1 : Char Data

If the specified characteristic property contains notify/indicate, data update require BLE Central (ex.mobile phone) enable the notify/indicate descriptor (CCCD) previously.

Attribute value type is uint8_t, so the input hex number must be even number, ex. 2, 4, 6, etc. Maximum data length is 20 bytes.

3.12 GATT Read Characteristic Value

READ CHARACTERITSTIC VALUE

Function: Read current characteristic value in BLE module.

Command Format: cynb readData <SERVICE UUID> <CHAR UUID>

Example:

```
        COMMAND: cynb readData 0x180D 0x2A39<cr>
        RESPONSE: <cr_lf>
                           OK;0x1234;<cr_lf>
                           <cr_lf>
```

Note: Max value field contains with 20 bytes, which equal to 40 hex numbers.

3.13 BLE Scan Start

PERFORM BLE SCAN

Function: Start Scanning

Command Format: cynb scanStart <INTERVAL> <WINDOW> <TIMEOUT>

Example:

```
        COMMAND: cynb scanStart <cr>
        RESPONSE: Start Scan<cr_lf>
                           GOLiFE CARE,ADV,[ED F1 9F 9B C7 31],-95,0;<cr_lf>
                           <cr_lf>
```

Note:
<Input Parameters> Interval value should be larger or equal to window value. The unit of INTERVAL and WINDOW is minisecond; The unit of TIMEOUT is second, which means that BLE Scanning will stop after <TIMEOUT> seconds, if <TIMEOUT> set to 0, disable timeout.
Default interval value is 500 ms, default window value is 400 ms, default time-out value is 5 seconds.
INTERVAL range: Maximum: 10240 Minimum:3

WINDOW range: Maximum: 10240 Minimum:3
TIMEOUT range: Maximum: 16383 Minimum:1
<Output Parameters>
<DEVICE_NAME>, ADV,<BLE ADDR>,<RSSI>,<ADV TYPE>,
<ADV TYPE> available list as below:
0:ADV_CONNECTABLE_UNDIRECTED,
1:ADV_CONNECTABLE_DIRECTED,
2:ADV_SCANNABLE_UNDIRECTED,
3:ADV_NON_CONNECTABLE_UNDIRECTED

3.14 BLE Scan Stop

READ CHARACTERITSTIC VALUE

Function: Read current characteristic value in BLE module.

Command Format: cynb scanStop

Example:

```
    COMMAND: cynb scanStop <cr>
    RESPONSE: <cr_lf>
              OK;<cr_lf>
             <cr_lf>
```

3.15 BLE Connect

PERFORM BLE CONNECT

Function: Connect to specific BLE device

Command Format: cynb connect <NAME>

Example:

```
COMMAND: cynb connect Test<cr>
RESPONSE: Test,ADV,[E4 FE AD 21 8F 5B],-86,0
          <cr_lf>
          OK;<cr_lf>
          <cr_lf>
```

Note: Connect command will trigger module BLE scan and looking for specific device name to connect.

3.16 BLE Disconnect

BLE DISCONNECTION

Function: Disconnect from current BLE connection.

Command Format: cynb disconn

Example:

```
COMMAND: cynb disconn<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
```

Note: If the connection does not create, system show error message for invalid state.

3.17 BLE Data Interrupt

BLE ENABLE DATA INT

Function: Enable client write detection, action included showing write data, give interrupt to GPIO PIN30 and identify write command type.

Command Format: cynb enInt

Example:

```
        COMMAND: cynb enInt<cr>
        RESPONSE: <cr_lf>
                          OK<cr_lf>
                          <cr_lf>
                          ...
                          w2,180F,2A19,1,22; <cr_lf>
```

RESULT: This example print string list in below sequence, (wX: Write command type), (Service UUID), (Characteristic UUID), (Data length), (Data in Hex)

Note:

Write command can be listed as below:

w0: Invalid operation
w1: Write
w2: Write without response
w3: Signed write
w4: Prepare write
w5: Cancel all prepared write
w6: Execute all prepared write

BLE DISABLE DATA INT

Function: Disable client write detection

Command Format: cynb disInt

Example:

```
    COMMAND: cynb disInt<cr>
    RESPONSE: <cr_lf>
              OK<cr_lf>
              <cr_lf>
```

4. Wi-Fi Commands Description

This part describes the usage of each Wi-Fi command. Each Wi-Fi command is mapped to Wi-Fi SDK (please see the SDK document). To use Wi-Fi SDK, you can call the Wi-Fi API on mbed platform, or you can use the Wi-Fi command directly.

4.1 Wi-Fi Device Command

4.1.1 Device Sleep

WIFI DEVICE SLEEP

Function: Disable Wi-Fi and set into sleep mode (no Wi-Fi function is available).

Command Format: cynw device_sleep

Example:

```
    COMMAND: cynw device_sleep<cr>
    RESPONSE: <cr_lf>
              OK;<cr_lf>
              <cr_lf>
```

4.1.2 Device Network

SET WIFI DEVICE NETWORK

Function: Set SSID, password and priority to connect.

Command Format: cynw device_network <SSID> <PASSWORD> <PRIORITY>

Example:

```
        COMMAND: cynw device_network DELTA_CLI 12345678 0<cr>
        RESPONSE: <cr_lf>
                         OK;<cr_lf>
                         <cr_lf>
```

Note: The max length of SSID is 20, PASSWORD is 11. The valid PRIORITY is range from 0 to 2. It should be noted that even if the AP is set to Open Security, the <PASSWORD> still need enter in the command.

4.1.3 Device Information

GET WIFI SP VERSION

Function: Read WIFI Service Pack version

Command Format: cynw device_read_version

Example:

```
        COMMAND: cynw device_read_version<cr>
        RESPONSE: <cr_lf>
                         OK;<cr_lf>
                         <cr_lf>
                          0;20150421;20150714<cr_lf>
```

Note: Read Wifi SP version command have 3 response output, list by below order: Wifi chip ID, Station mode image ID, AP mode image ID.

4.1.4 Enable Coexistence

PERFORM WIFI and BLE Coexistence

Function: Enable the setting of two coexistence control pins (COEX_B and COEX_W) are short to each other.

Command Format: cynw device_coex

Example:

```
    COMMAND: cynw device_coex<cr>
    RESPONSE: <cr_lf>
                        OK;<cr_lf>
                        <cr_lf>
```

Note: Enable antenna auto switch, by check BLE work state to switch the antenna, if BLE status off, switch antenna to Wifi function.

4.1.5 AP Scan

PERFORM WIFI AP SCAN

Function: Scan for AP and get SSID,channel, security information.

Command Format: cynw device_apscan

Example:

```
    COMMAND: cynw device_apscan<cr>
    RESPONSE: <cr_lf>
                        OK;<cr_lf>
                        <cr_lf>
                         1,Delta_AP,-73,1,3<cr_lf>
```

Note: ethernet_init command mulst be performed before AP scan, or terminal will show error message “Wrong command order.”

4.1.6 Set Access Point

Set module in AP mode

Function: Set SSID, password, security type and channel in AP mode.

Command Format: cynw device_setap <SSID> <PASSWORD> <SECURITY> <CHANNEL>

Example:

```
COMMAND: cynw device_setap Delta_AP 01234567 2 1<cr>
RESPONSE: <cr_lf>
                OK;<cr_lf>
                <cr_lf>
```

Note: Command perform must before Wifi initialization, or it will show "Wrong command order", and need to be reset.

The max length of SSID is 20, PASSWORD is 11. It should be noted that even if the AP is set to Open Security, the <PASSWORD> still need enter in the command.

Security type can be configured including

0:OPEN
1:WEP_PSK
2:WEP_SHARED
3:WPA_TKIP_PSK
4:WPA_AES_PSK
5:WPA2_AES_PSK
6:WPA2_TKIP_PSK
7:WPA2_MIXED_PSK

4.1.7 Write to Flash Memory

PERFORM MEMORY WRITE

Function: Write data into embedded flash.

Command Format: device_mem_write <MEMADDR> <DATA>

Example:

```
        COMMAND: cynw device_mem_write 0x00 0x1234<cr>
        RESPONSE: <cr_lf>
                         OK;<cr_lf>
                         <cr_lf>
```

Note: Flash memory must be erased before write executed. <MEMADDR>: flash memory address to be written, range from 0x00 to 0x3FFFF. <DATA>: format will be uint8_t.

4.1.8 Read Flash Memory

Read data from embeeded flash.

Function: Read data from embeeded flash
Command Format: cynw device_mem_read <MEMADDR>

Example:

```
        COMMAND: cynw device_mem_read 0x0 5<cr>
        RESPONSE: <cr_lf>
                         OK;<cr_lf>
                         <cr_lf>
                          FFFFFFFFFF<cr_lf>
```

Note: <MEMADDR>: flash memory address to be written, range from 0x00 to 0x3FFFF. Reading data format will be uint8_t.

4.1.9 Memory Erase

PERFORM MEMORY ERASE

Function: Flash memory will be erased in groups of 4KB sector.

Command Format: cynw device_mem_erase4k <MEMADDR>

Example:

```
    COMMAND: cynw device_mem_erase4k <cr>
    RESPONSE: <cr_lf>
                    OK;<cr_lf>
                    <cr_lf>
```

Note: address range from 0x00 to 0x3F000 (must be a multiple of 0x1000)

4.1.10 Wifi Data Interrupt

ENABLE DATA INTERRUPT

Function: Enable wifi data received detection and print receive data whenever received, also give interrupt to GPIO PIN30

Command Format: cynw device_enint

Example:

```
    COMMAND: cynw device_enint <cr>
    RESPONSE: <cr_lf>
                    OK;<cr_lf>
                    <cr_lf>
                    ...
                    rTCP;test,6,192.168.168.101,5222;
```

Note: Print string list in below sequence, (rX: Communication Type);(Data in string type),(Data length),(Port Number);
Communication Type: TCP or UDP

DISABLE DATA INTERRUPT

Function: Disable wifi data received detection.

Command Format: cynw device_disint

Example:

```
COMMAND: cynw device_disint <cr>
RESPONSE: <cr_lf>
                OK;<cr_lf>
                <cr_lf>
```

4.2 Ethernet Interface Command

4.2.1 Ethernet Interface Initiation

INITIALIZE ETHERNET INTERFACE WITH DHCP

Function: Initialize the interface and configure it to use DHCP (no connection at this step).

Command Format: cynw ethernet_init

Example:

```
COMMAND: cynw ethernet_init<cr>
RESPONSE: <cr_lf>
                OK;<cr_lf>
                <cr_lf>
```

INITIALIZE ETHERNET INTERFACE WITH STATIC

Function: Initialize the interface and use the static configure (no connection at this step).

Command Format: cynw ethernet_init <STATIC IP>

Example:

```
        COMMAND: cynw ethernet_init 192.168.1.2<cr>
        RESPONSE: <cr_lf>
                  OK;<cr_lf>
<cr_lf>
```

Note: The max length of STATIC IP is 15

4.2.2 Ethernet Interface Connection

ETHERNET INTERFACE CONNECTION

Function: Bring the Wi-Fi connection up, start DHCP if needed.

Command Format: cynw ethernet_connect <TIMEOUT MS>

Example:

```
        COMMAND: cynw ethernet_connect 50000<cr>
        RESPONSE: <cr_lf>
                  Connecting..., Waiting for 50000 ms<cr_lf>
                  <cr_lf>
                  OK;<cr_lf>
                  <cr_lf>
```

Note: The <TIMEOUT> is optional argument, default is 35000 ms. It is notice that if the <TIMEOUT> is too short, the connection may not be completed connection procedure.

4.2.3 Ethernet Interface Disconnection

ETHERNET INTERFACE DISCONNECTION

Function: Bring the Wi-Fi interface down.

Command Format: cynw ethernet_disconnect

Example:

```
        COMMAND: cynw ethernet_disconnect <cr>
        RESPONSE: <cr_lf>
                         OK;<cr_lf>
                         <cr_lf>
```

4.2.4 Ethernet Interface MAC Address

GET MAC ADDRESS

Function: Get the MAC address of your Ethernet interface.

Command Format: cynw ethernet_mac

Example:

```
        COMMAND: cynw ethernet_mac<cr>
        RESPONSE: <cr_lf>
                         OK;00:50:c2:5e:10:dd <cr_lf>
                         <cr_lf>
```

4.2.5 Ethernet Interface IP Address

GET IP ADDRESS

Function: Get the IP address of your Ethernet interface.

Command Format: cynw ethernet_ip

Example:

```
COMMAND: cynw ethernet_ip<cr>
RESPONSE: <cr_lf>
                    OK; 192.168.1.24<cr_lf>
                    <cr_lf>
```

4.3 Wi-Fi TCP Socket Server Command

4.3.1 TCP Socket Server Bind

BIND A SPECIFIC PORT

Function: Bind a socket to a specific port.

Command Format: cynw tcp_server_bind <PORT>

Example:

```
COMMAND: cynw tcp_server_bind 5567<cr>
RESPONSE: <cr_lf>
                    OK;<cr_lf>
                    <cr_lf>
```

4.3.2 TCP Socket Server Listen

LISTEN A SPECIFIC PORT

Function: Start listening for incoming connections.

Command Format: cynw tcp_server_listen

Example:

```
COMMAND: cynw tcp_server_listen<cr>
RESPONSE: <cr_lf>
                 OK;<cr_lf>
                 <cr_lf>
```

4.3.3 TCP Socket Server Accept

ACCEPT A NEW CONNECTION

Function: Accept a new connection.

Command Format: cynw tcp_server_accept

Example:

```
COMMAND: cynw tcp_server_accept<cr>
RESPONSE: <cr_lf>
                 OK;<cr_lf>
                 <cr_lf>
```

4.3.4 TCP Socket Server Blocking

SET TCP SOCKET SERVER BLOCKING

Function: Set blocking or non-blocking mode of socket and a timeout on blocking socket operations.

Command Format: cynw tcp_server_blocking <SETTING> <TIMEOUT MS>

Example:

```
COMMAND: cynw tcp_server_blocking 0 2000<cr>
RESPONSE: <cr_lf>
          OK;<cr_lf>
          <cr_lf>
```

Note: The <SETTING> is mandatory argument, given 1 represent block until receive, given 0 represent block until timeout or receive. The <TIMEOUT MS> is optional argument, default is 1500 ms. TCP Socket Server Blocking is applied to TCP Socket Server operation.

4.3.5 TCP Socket Server Close

CLOSE TCP SOCKET SERVER

Function: Close the socket.

Command Format: cynw tcp_server_close <SHUTDOWN>

Example:

```
COMMAND: cynw tcp_server_close 1<cr>
RESPONSE: <cr_lf>
          OK;<cr_lf>
          <cr_lf>
```

Note: <SHUTDOWN> is set to 1 to free left-over data in message queue, and set to 0 to keep the data.

4.4 Wi-Fi TCP Socket Connection Command

4.4.1 TCP Connection Connect

CONNECT TO THE SPECIFIC SERVER

Function: Connects this TCP socket to the server.

Command Format: cynw tcp_connection_connect <IP ADDRESS> <PORT>

Example:

```
COMMAND: cynw tcp_connection_connect 192.168.1.24 5520<cr>
RESPONSE: <cr_lf>
                OK;<cr_lf>
                <cr_lf>
```

Note: Host address only support IP Address, not support DNS.

4.4.2 TCP Connection Is Connected

TCP CONNECTION STATE

Function: Check if the socket is connected.

Command Format: cynw tcp_connection_is_connect

Example:

```
COMMAND: cynw tcp_connection_is_connect<cr>
RESPONSE: <cr_lf>
                OK;TRUE<cr_lf>
                <cr_lf>
```

Note: If TCP connection is connected, the output will be TRUE, otherwise the output is FALSE.

4.4.3 TCP Connection Send

SEND DATA THROUGH TCP CONNECTION

Function: Send data to the remote host.

Command Format: cynw tcp_connection_send <LENGTH> <DATA>

Example:

```
COMMAND: cynw tcp_connection_send 5 Hello<cr>
RESPONSE: <cr_lf>
                OK;<cr_lf>
                <cr_lf>
```

Note: The length of <DATA> should not be exceeding 1400. (TCP max data length is 1400 byte)

4.4.4 TCP Connection Send All

SEND ALL THE DATA THROUGH TCP CONNECTION

Function: Send all the data to the remote host.

Command Format: cynw tcp_connection_send_all <LENGTH> <DATA>

Example:

```
COMMAND: cynw tcp_connection_send_all 5 Hello<cr>
RESPONSE: <cr_lf>
                OK;<cr_lf>
                <cr_lf>
```

Note: The length of <DATA> should not be exceeding 1400. (TCP max data length is 1400 byte).

4.4.5 TCP Connection Receive

RECEIVE DATA THROUGH TCP CONNECTION

Function: Receive data from remote host.

Command Format: cynw tcp_connection_receive <DATA LENGTH>

Example:

```
COMMAND: cynw tcp_connection_receive 10<cr>
RESPONSE: <cr_lf>
          OK;Hello<cr_lf>
          <cr_lf>
```

Note: The <DATA LENGTH> should not be exceeding 1400. (TCP max data length is 1400 byte)

4.4.6 TCP Connection Receive All

RECEIVE ALL THE DATA THROUGH TCP CONNECTION

Function: Receive all the data from remote host.

Command Format: cynw tcp_connection_receive_all <DATA LENGTH>

Example:

```
COMMAND: cynw tcp_connection_receive_all 10<cr>
RESPONSE: <cr_lf>
          OK; Hello <cr_lf>
          <cr_lf>
```

Note: The <DATA LENGTH> should not be exceeding 1400. (TCP max data length is 1400 byte)

4.4.7 TCP Connection Blocking

SET TCP CONNECTION BLOCKING

Function: Set blocking or non-blocking mode of socket and a timeout on blocking socket operations.

Command Format: cynw tcp_connection_blocking <SETTING> <TIMEOUT MS>

Example:

```
    COMMAND: cynw tcp_connection_blocking 0 2000<cr>
    RESPONSE: <cr_lf>
                   OK;<cr_lf>
                   <cr_lf>
```

Note: The <SETTING> is mandatory argument, given 1 represent block until receive, given 0 represent block until timeout or receive. <TIMEOUT MS> is optional argument, default is 1500 ms. TCP Connection Blocking is applied to TCP Connection operation.

4.4.8 TCP Connection Close

CLOSE TCP CONNECTION

Function: Close the socket.

Command Format: cynw tcp_connection_close <SHUTDOWN>

Example:

```
    COMMAND: cynw tcp_connection_close 1<cr>
    RESPONSE: <cr_lf>
                   OK;<cr_lf>
                   <cr_lf>
```

Note: <SHUTDOWN> is set to 1 to free left-over data in message queue, and set to 0 to keep the data.

4.5 Wi-Fi UDP Socket Command

4.5.1 UDP Client Socket Initiation

INITIALIZE UDP CLIENT SOCKET

Function: Initialize the UDP Client Socket without binding it to any specific port.

Command Format: cynw udp_init

Example:

```
    COMMAND: cynw udp_init<cr>
    RESPONSE: <cr_lf>
                    OK;<cr_lf>
                    <cr_lf>
```

4.5.2 UDP Socket Server Bind

BIND UDP SERVER SOCKET

Function: Bind a UDP Server Socket to a specific port.

Command Format: cynw udp_bind <PORT>

Example:

```
    COMMAND: cynw udp_bind 5520<cr>
    RESPONSE: <cr_lf>
                    OK;<cr_lf>
                    <cr_lf>
```

4.5.3 UDP Socket Broadcasting

SET UDP SOCKET BROADCASTING

Function: Set the socket in broadcasting mode.

Command Format: cynw udp_set_broadcasting <IS_BROADCAST>

Example:

```
COMMAND: cynw udp_set_broadcasting 1<cr>
RESPONSE: <cr_lf>
                OK;<cr_lf>
               <cr_lf>
```

Note: < IS_BROADCAST> is set to 1 if broadcasting mode is needed, otherwise the value is set to 0.

4.5.4 UDP Socket Send

SEND UDP SOCKET

Function: Send a packet to a remote endpoint.

Command Format: cynw udp_send_to <LENGTH> <DATA>

Example:

```
COMMAND: cynw udp_send_to 5 Hello<cr>
RESPONSE: <cr_lf>
                OK;<cr_lf>
               <cr_lf>
```

Note: The <DATA LENGTH> should not be exceeding 1400. (UDP max data length is 1400 byte)

4.5.5 UDP Socket Receive

RECEIVE UDP SOCKET

Function: Receive a packet from a remote endpoint.

Command Format: cynw udp_received_from <DATA LENGTH>

Example:

```
COMMAND: cynw udp_received_from 10<cr>
RESPONSE: <cr_lf>
                    OK;Hello<cr_lf>
                    <cr_lf>
```

Note: The <DATA LENGTH> should not be exceeding 1400. (UDP max data length is 1400 byte)

4.5.6 UDP Socket Blocking

SET UDP SOCKET BLOCKING

Function: Set blocking or non-blocking mode of the socket and a timeout on blocking socket operations.

Command Format: cynw udp_blocking <SETTING> <TIMEOUT MS>

Example:

```
COMMAND: cynw udp_blocking 0 2000<cr>
RESPONSE: <cr_lf>
                    OK;<cr_lf>
                    <cr_lf>
```

Note: The <SETTING> is mandatory argument, given 1 represent block until receive, given 0 represent block until timeout or receive. The <TIMEOUT MS> is optional argument, default is 1500 ms. UDP Socket Blocking is applied to UDP Socket operation.

4.5.7 UDP Socket Close

CLOSE UDP SOCKET

Function: Close the socket.

Command Format: cynw udp_close <SHUTDOWN>

Example:

```
    COMMAND: cynw udp_close 1<cr>
    RESPONSE: <cr_lf>
              OK;<cr_lf>
              <cr_lf>
```

Note: <SHUTDOWN> is set to 1 to free left-over data in message queue, and set to 0 to keep the data.

4.6 Wi-Fi UDP Endpoint

4.6.1 UDP ENDPOINT RESET

RESET UDP ENDPOINT

Function: Reset the address of this endpoint.

Command Format: cynw udp_endpoint_reset

Example:

```
    COMMAND: cynw udp_endpoint_reset<cr>
    RESPONSE: <cr_lf>
              OK;<cr_lf>
              <cr_lf>
```

4.6.2 UDP ENDPOINT ADDRESS

SET UDP ENDPOINT ADDRESS

Function: Set the address of this endpoint.

Command Format: cynw udp_endpoint_address <IP ADDRESS> <PORT>

Example:

```
    COMMAND: cynw udp_endpoint_address 192.168.1.24 5520<cr>
    RESPONSE: <cr_lf>
                  OK;<cr_lf>
                  <cr_lf>
```

Note: Host address is only support IP Address, not support DNS.

GET UDP ENDPOINT ADDRESS

Function: Get the IP address of this endpoint.

Command Format: cynw udp_endpoint_address

Example:

```
    COMMAND: cynw udp_endpoint_address<cr>
    RESPONSE: <cr_lf>
                  OK;192.168.1.24<cr_lf>
                  <cr_lf>
```

Note: Host address is only support IP Address, not support DNS.

4.6.3 UDP ENDPOINT PORT

GET UDP ENDPOINT PORT

Function: Get the port of this endpoint.

Command Format: cynw udp_endpoint_port

Example:

```
        COMMAND: cynw udp_endpoint_port<cr>
        RESPONSE: <cr_lf>
                         OK;5520<cr_lf>
                         <cr_lf>
```

5. BLE Example for creating a profile by command

5.1 Tools Preparation

1. **Terminal tool:** Because the command line interface uses UART for configuration and communication, you need to install terminal emulator tool such as TERATERM or PUTTY.

2. **USB to UART convertor:** Connect the convertor to 1.8 V or 3.3 V UART output pin of NNN40. Please see the detail in 5.2.

3. **BLE APP:** To display the BLE GATT profile created by CLI, you need to install BLE APP (in mobile phone or other BLE central system, at least support BLE 4.0) to verify the configuration and input data. Here we use Nordic's Wireless APP name "nRF Master Control Panel", which can be found on GOOGLE PLAY store for Android system, and APPLE STORE for iOS system.

5.2 Procedure for connecting UART

Before connect the USB-to-UART convertor to NNN40, the resistors on SB13 and SB14 should be removed. Connect the convertor TX pin to NNN40 pin STLK_TX and RX pin to NNN40 pin STLK_RX if 3.3V is needed. If 1.8V is requested, then convertor TX pin is connected to NNN40 pin P25 and RX pin to NNN40 pin P23.

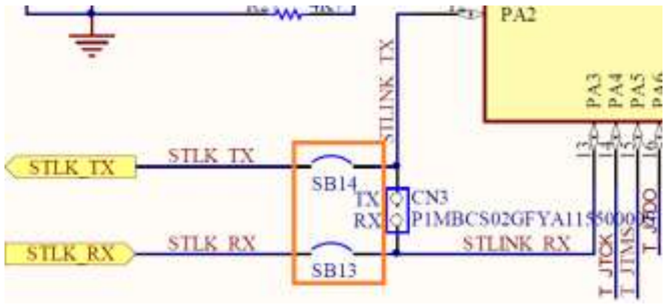
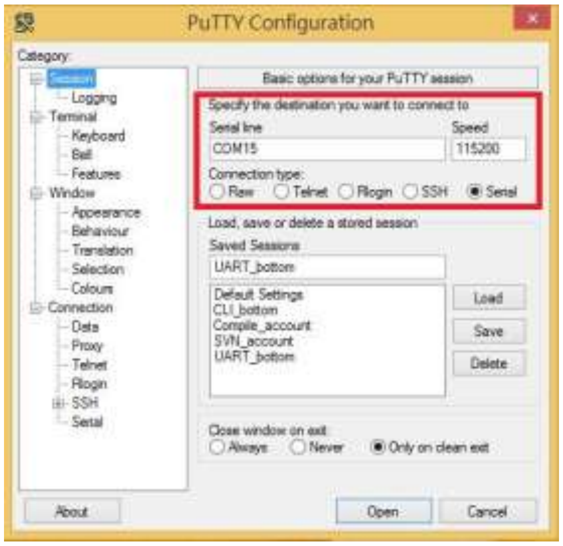


Figure 1. Remove the resistors on SB13 and SB14

Use the terminal to choose the correct COM port (the information show on device manager) and set the data rate to 115200 bps, also set the local echo on as figure 2 and figure 3. Press the reset button in module and reset module.



Specify the destination you want to connect to

Serial line: COM15

Speed: 115200

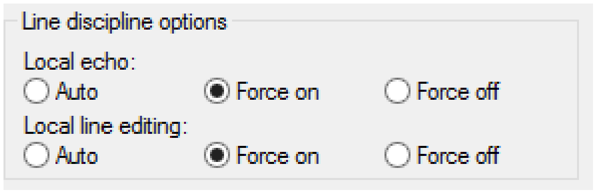
Connection type: Raw / Telnet / Rlogin / SSH / [✓ Serial]

Saved Sessions: UART_bottom

Default Settings
CLI_bottom
Compile_account
SVN_account
UART_bottom

Close window on exit: Always / Never / [✓ Only on clean exit]

Figure 2. Terminal emulator configuration-1


Line discipline options

Local echo: Auto / [✓ Force on] / Force off

Local line editing: Auto / [✓ Force on] / Force off

Figure 3. Terminal emulator configuration-2

Note: For other com port terminal software tool, enter string may not include “\r”, therefore user need

add it in the end of line.

5.3 Procedure for creating an Profile

BLE sensor device (in this case means NNN40 with glucose sensor) have 6 possible states, when module power on, transit to Initializing state.
After initialization, device transit to Configure state, in this state, Collector can configure device name and BLE address and build desired GATT services, etc.

RF is powered off to conserve energy for a specific time internal (defined in SIG profile) before transit to Advertising state where RF is powered on to send out an advertising packet. Once the transmission of advertising packet is completed, it transit back to Standby. The transition between Standby and Advertising will repeat until the sensor device is connected and transit to Connecting state when the advertising packet is received by central (collector) who intent to connect to this BLE sensor device.

In connecting state, all services such as Glucose service and Device Information Service can be accessed by central. The transition go back to Standby when the connection is terminated by central or sensor device.

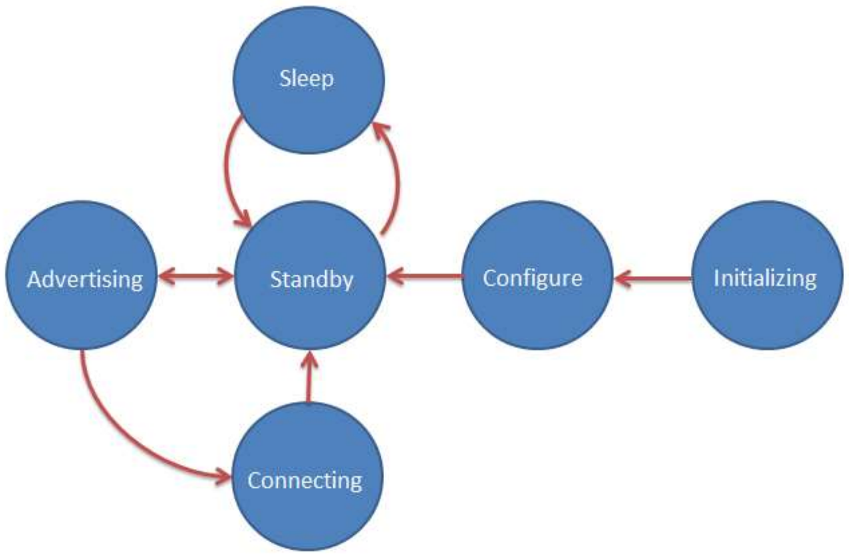


Figure.2 BLE state change diagram

Step1. Sensor power on, transit to **Initializing state**. Typically with software reset first, then check the module information and firmware information.

```
cynb reset

cynb init (must do after reset success)
cynb info
```

Step2. Device transit to **Configure state**, configure by collector, with device name and transmit power, user can also build GATT services and characteristics, here 0x1808 correspond to Glucose Service, and Number string after correspond to their characteristics, and data format refer to Reference.1 "Glucose Profile". On the other hand, user may add own vendor-specific services/characteristics UUID in this step.

```
cynb name <name length>s <device name string>
cynb bleAddr <BLE address 12 Hex>
cynb txPow <Integer Value>
cynb gattService 0x1808
cynb gattChar 0x2A34 <Property 2 Hex> <Hex Value>
cynb gattChar 0x2A51<Property 2 Hex> < Hex Value >
cynb gattChar 0x2A52 <Property 2 Hex> < Hex Value >
……
cynb regService
```

Step3. Glucose Sensor will typically remain powered off between uses, after sleep command execute, device transit to **Sleep state**, and will only advertise and allow a Collector to connect when it is turned on by the user and has data to send, key in any input will make device transit to **Standby state**.

```
cynb sleep
<any key to turn on>
```

Step4. Enter GAP connectable mode, device transit to **Advertising state**, sensor start advertising with specific interval and timeout, device waits until Collector initialize connection request.

```
cynb advStart 64 180
```

Step5. When device transit to **Connecting State**, meanwhile connection is established, the Glucose Sensor sends one or more notifications and indications to the Collector, also, if the property with write, Collector can initiatively update stored value in Sensor. User may also use BLE data interrupt function , inform Host for BLE data update from connected BLE device.

```
cynb update 0x1808 0x2A18 <Value>
cynb readData 0x1808 0x2A18
cynb enInt
```

Step6. When the data transfer is complete the Glucose Sensor typically terminates the connection, device back transit to **Standby state**.

```
cynb disconn
```

6. WIFI Example for connecting to AP by command

6.1 Tools Preparation

1. **Terminal tool and UART convertor:** Same as the sector 5.1.
2. **Wi-Fi Access Point:** A Wi-Fi AP (Access Point) is needed for NNN40 to connect.

6.2 Wi-Fi AP mode Configuration

Below steps show how to use command sets to connect to AP.

Step 1: "cynw device_coex", switch antenna to auto BLE and Wifi coexistence
Step 2: "cynw device_setap Delta_AP 0123456789 2 1", set SSID, PASSWORD, SECURITY TYPE, CHANNEL of the AP mode configuration.

Step 3: "cynw ethernet_init", Initialize the Ethernet interface
Step 4: "cynw ethernet_connect 40000", Connect to the AP with timeout 40000ms, it is notice that if the connection failed, the UART would not be used anymore. Please reset or reboot the module to try to connect again.

6.3 Wi-Fi Station mode Connection

Below steps show how to use command sets to connect to AP.

Step 1: "cynw device_coex", switch antenna to auto BLE and Wifi coexistence
Step 2: "cynw device_network DELTA_AP 12345678 0", set SSID and PASSWORD of the connecting AP
Step 3: "cynw ethernet_init", Initialize the Ethernet interface, **NOTE THAT initialization can't be pre-execute before " device_network" command, or command order error may occur.**
Step 4: "cynw ethernet_connect 40000", Connect to the AP with timeout 40000ms, it is notice that if the connection failed, the UART would not be used anymore. Please reset or reboot the module to try to connect again.
Step 5: "cynw ethernet_mac", To get Ethernet MAC address
Step 6: "cynw ethernet_ip", To get IP address if connection is success.

6.4 Wi-Fi Communication Protocol

The Wi-Fi communication protocol is described on mbed handbook (http://developer.mbed.org/handbook/Socket). To test UDP or TCP communication, the command set should be called according to mapping API. The following chapter list the command set to implement the communication.

6.5 TCP client

Step 1: "cynw tcp_connection_connect 192.168.15.111 5520"
Step 2: "cynw tcp_connection_send test"
Step 3: "cynw tcp_connection_blocking 0 2000"
Step 4: "cynw tcp_connection_receive 10"
Step 5: "cynw tcp_connection_close"
Step 6: "cynw ethernet_disconnect"

Note that you should prepare the TCP Server to correspond the client.
Below is the example python program to implement the TCP Server in this case:

```
import socket
import time

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print 'bind'
s.bind(('', 5222))
print 'listen and accept'
s.listen(1)
conn, addr = s.accept()

# After connection
print 'Start receive data...'
data = conn.recv(1024)
print 'Wait until Send'
time.sleep(15)
print 'Send data'
conn.sendall(data)
conn.close()
print 'Closed.'
print 'Received data', repr(data)
```

6.6 TCP server

Step 1: "cynw tcp_server_bind 5520"
Step 2: "cynw tcp_server_listen"
Step 3: "cynw tcp_server_accept"
Step 4: "cynw tcp_connection_blocking 0 2000"
Step 5: "cynw tcp_connection_receive 10"
Step 6: "cynw tcp_connection_close"
Step 7: "cynw tcp_server_close"

Step 8: “cynw ethernet_disconnect”

Note that you should prepare the TCP Client to correspond the server.
Below is the example python program to implement the TCP Server in this case:

```
import socket
import time

ECHO_SERVER_ADDRESS = "192.168.168.101"
ECHO_PORT = 5520
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((ECHO_SERVER_ADDRESS, ECHO_PORT))
print 'Connect Success....'

time.sleep(30)
print 'Send.....'
s.sendall('Hello, world')
data = s.recv(1024)
s.close()
print 'Received', repr(data)
```

6.7 UDP client

Step 1: “cynw udp_init”
Step 2: “cynw udp_endpoint_address 192.168.168.101 5222”
Step 3: “cynw udp_endpoint_port”
Step 4: “cynw udp_blocking 1 1500”
Step 5: “cynw udp_send_to test”
Step 6: “cynw udp_received_from 10”
Step 7: “cynw udp_close”
Step 8: “cynw ethernet_disconnect”

Note that you should prepare the UDP Server to correspond the client.
Below is the example python program to implement the UDP Server in this case:

```
import socket

ECHO_PORT = 5222

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind(('', ECHO_PORT))

print "Waiting for UDP data packet..."
data, address = sock.recvfrom(256)
print "Received packet from", address, "with data",data
print "Sending    packet back to client"
time.sleep(20)
sock.sendto(data, address)
sock.close()
```

6.8 UDP Server

Step 1: "cynw udp_bind 5520"
Step 2: "cynw udp_received_from 15"
Step 3: "cynw udp_send_to test"
Step 4: "cynw udp_close"
Step 5: "cynw ethernet_disconnect"

Note that you should prepare the UDP Client to correspond the server.
Below is the example python program to implement the UDP Client in this case:

```
import socket
```

```
ECHO_SERVER_ADDRESS = '192.168.168.102'
ECHO_PORT = 5520

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

print 'Connect Success....'
print 'Send.....'

sock.sendto("Hello World\n", (ECHO_SERVER_ADDRESS, ECHO_PORT))
response = sock.recv(64)
sock.close()

print response
```

7. BLE Command Set Summary Table

Command Format	Command Response	Description
cynb gpio <GPIO-NO> <SET/CLEAR>	OK;	Set or reset (high/low) a GPIO pin
cynb sleep <GPIO-NO>	OK;	Go to system off mode, wake up by GPIO pin
cynb disconn	OK;	Disconnect from current connection
cynb reset	OK;	Perform soft reset for BLE module
cynb info	OK;<FW version>;<module name>;	Get BLE module Information
cynb name <LENGTH> <NAME>	OK;	Set device name for BLE module
cynb name	OK;<device name>;	Get device name for BLE module
cynb advStart <INTERVAL> <TIMEOUT>	OK;	Start broadcast advertising packet with specific interval and timeout
cynb advStop	OK;	Stop broadcast advertising packet
cynb txPow <TX POWER>	OK;	Set Tx power to BLE module
cynb bleAddr <BLE ADDR>	OK;	Set BLE address

cynb bleAddr	OK;<BLE ADDR>;	Get BLE address
cynb gattService <service UUID>	OK;	Add GATT service configuration
cynb gattChar <char UUID> <property> <value>	OK;	Add GATT characteristic in current add service
cynb regService	OK	Register new GATT service
cynb update <SERVICE UUID> <CHAR UUID> <value>	OK;	Reset characteristic value to connected device
cynb readData <SERVICE UUID> <CHAR UUID>	OK;<Rx data>	Read characteristic value of BLE module
cynb enInt	OK;	Enable data interrupt
cynb disInt	OK;	Disable data interrupt

8. WIFI Command Set Summary Table

Command Format	Command Response	Description
cynw device_sleep	OK;	Set WIFI module to sleep mode
cynw device_network <SSID> <PASSWORD> <PRIORITY>	OK;	Set SSID and PASSWORD for WIFI module
cynw device_read_version	OK	Read WIFI Service Pack version
cynw device_apscan	OK	Scan for available access point on all channels.
cynw device_mem_erase4k	OK	Erase a 4KB sector of embedded flash
cynw device_mem_read	OK	Read data from embeeded flash
cynw device_mem_write	OK	Write data into embeeded flash
cynw device_enint	OK	Enable Wifi data interrupt
cynw device_disint	OK	Disable Wifi data interrupt
cynw ethernet_init	OK;	Initialize the interface to use DHCP
cynw ethernet_init <STATIC IP>	OK;	Initialize the interface with static
cynw ethernet_connect <TIMEOUT MS>	OK;	Bring up the WIFI connection
cynw ethernet_disconnect	OK	Bring the interface down
cynw ethernet_mac	OK;<MAC Address>	Get MAC addr of Ehternet Interface

cynw ethernet_ip	OK;<IP Address>	Get IP addr of Ehternet Interface
cynw tcp_server_bind <PORT>	OK;	Bind a socket to a port
cynw tcp_server_listen	OK;	Start listening for incoming connections
cynw tcp_server_accept	OK;	Accept a new connection
cynw tcp_server_blocking <SETTING> <TIMEOUT MS>	OK;	Set blocking mode and timeout
cynw tcp_server_close <SHUTDOWN>	OK;	Close the socket
cynw tcp_connection_connect <IP ADDRESS> <PORT>	OK;	Connects TCP socket to the server
cynw tcp_connection_is_connect	OK;<IS_CONNECTED>	Check if the socket is connected
cynw tcp_connection_send <LENGTH> <DATA>	OK;	Send data to the remote host
cynw tcp_connection_send_all <LENGTH> <DATA>	OK;	Send all the data to the remote host
cynw tcp_connection_receive <DATA LENGTH>	OK;	Receive data from the remote host
cynw tcp_connection_receive_all <DATA LENGTH>	OK;	Receive all the data from the remote host
cynw tcp_connection_blocking <SETTING> <TIMEOUT MS>	OK;	Set blocking mode and timeout
cynw tcp_connection_close <SHUTDOWN>	OK;	Close the connection
cynw udp_init	OK;	Initialize UDP Client Socket
cynw udp_bind <PORT>	OK;	Bind UDP Server Socket to a port
cynw udp_set_broadcasting <IS_BROADCAST>	OK;	Set socket in broadcasting
cynw udp_send_to <LENGTH> <DATA>	OK;	Send a packet to a remote endpoint
cynw udp_received_from <DATA LENGTH>	OK;	Receive a packet from a remote endpont
cynw udp_blocking <SETTING> <TIMEOUT MS>	OK;	Set blocking mode and timeout
cynw udp_close <SHUTDOWN>	OK;	Close the socket

cynw udp_endpoint_reset	OK;	Reset the address of this endpoint
cynw udp_endpoint_address <IP ADDRESS> <PORT>	OK;	Set the address of this endpoint
cynw udp_endpoint_address	OK;<ENDPOINT ADDR>	Get the address of this endpoint
cynw udp_endpoint_port	OK;<ENPOINT PORT>	Get the port of this endpoint

9. Simplified Command Set

9.1 Introduction

For code developer, repeated command test and usage is required, the time consuming is very large when keying the long command with 10 or higher alphabets. So the simplified version commands can solve this problem, it’s defined and can be modified in source code ”core_cli.cpp”.

9.2 BLE Command Correspondence

GENERAL	
cynb init	BLE INT
cynb gpio	BLE GIO
cynb sleep	BLE SLP
cynb reset	BLE RST
cynb info	BLE INF
cynb txPow	BLE POW
cynb name	BLE NAM
GATT	
cynb regService	BLE GRS
cynb gattChar	BLE GAC
cynb gattService	BLE GAS
cynb advStart	BLE ADS
cynb advStop	BLE ADP
cynb scanStart	BLE SCS

cynb scanStop	BLE SCP
cynb connect	BLE CON
cynb disconn	BLE DCN
cynb bleAddr	BLE ADR
cynb update	BLE WRT
cynb readData	BLE RED
cynb enInt	BLE EDI
cynb disInt	BLE DDI

9.3 Wifi Command Correspondence

Device	
cynw device_sleep	WIFI DSLP
cynw device_coex	WIFI DCOE
cynw device_network	WIFI DNWK
cynw device_read_version	WIFI DRSV
cynw device_apscan	WIFI DASN
cynw device_setap	WIFI DSAP
cynw device_mem_erase4k	WIFI DME4
cynw device_mem_read	WIFI DMRD
cynw device_mem_write	WIFI DMWT
cynw device_enint	WIFI DEDI
cynw device_disint	WIFI DDDI
Ethernet	
cynw ethernet_init	WIFI EINT
cynw ethernet_connect	WIFI ECON
cynw ethernet_disconnect	WIFI EDCN
cynw ethernet_mac	WIFI EMAC

cynw ethernet_ip	WIFI EGIP
TCP Server	
cynw tcp_server_bind	WIFI TSBD
cynw tcp_server_listen	WIFI TSLN
cynw tcp_server_accept	WIFI TSAC
cynw tcp_server_blocking	WIFI TSBL
cynw tcp_server_close	WIFI TSCL
TCP Socket	
cynw tcp_connection_connect	WIFI TCCN
cynw tcp_connection_is_connect	WIFI TCIC
cynw tcp_connection_send	WIFI TCSE
cynw tcp_connection_send_all	WIFI TCSA
cynw tcp_connection_receive	WIFI TCRC
cynw tcp_connection_receive_all	WIFI TCRA
cynw tcp_connection_blocking	WIFI TCBL
cynw tcp_connection_close	WIFI TCCL
UDP Socket	
cynw udp_init	WIFI UCIN
cynw udp_bind	WIFI UCBI
cynw udp_set_broadcasting	WIFI UCSB
cynw udp_send_to	WIFI UCSE
cynw udp_received_from	WIFI UCRC
cynw udp_blocking	WIFI UCBL
cynw udp_close	WIFI UCCL
UDP Endpoint	
cynw udp_endpoint_reset	WIFI UERS

cynw udp_endpoint_address	WIFI UEAD
cynw udp_endpoint_port	WIFI UEPT