



User Manual

epSDK Zephyr Agora53 User Manual

Document Revision History

Date	Description	Author	Version
09/25/2026	First Draft	M. Golob	1.0

TABLE OF CONTENTS

1.	ACRONYMS, ABBREVIATIONS, AND DEFINITIONS	4
2.	OVERVIEW	5
3.	CELLULAR LIBRARY	5
3.1	CONFIGURATION	6
3.2	API CALLS	7
3.2.1	cellInit()	7
3.2.2	registerCellular()	7
3.2.3	regStatus()	8
3.2.4	setGNSS()	8
3.2.5	statusGNSS()	9
3.2.6	connectCell()	9
3.2.7	closeCell()	10
3.2.8	sendHttpMsg()	10
3.2.9	queryCellularDiag()	11
3.2.10	addGnssQueueMsg	12
3.3	Cellular Queue	13
3.4	Cellular Thread	13
3.5	NRF Connect SDK Resources	13
3.6	Dependencies	14
4.	TIME LIBRARY	14
4.1	API Calls	14
4.1.1	set_time()	14
4.1.2	get_time_s ()	14
4.1.3	get_time_ms()	14
4.1.4	Example	15
4.2	NRF Connect SDK Resources	15
4.3	Dependencies	15
5.	LED LIBRARY	15
5.1	API Calls	15
5.1.1	led_init ()	15
5.1.2	led_mode()	16
5.1.3	led_pause ()	16
5.1.4	led_resume ()	16
5.1.5	Example	17
5.2	NRF Connect SDK Resources	17
5.3	Dependencies	17
6.	BLE	17
6.1	BLE Peripheral Library	18
6.1.1	CONFIGURATION	18
6.1.2	API Calls	18
6.1.3	NRF Connect SDK Resources	19
6.1.4	Dependencies	19
7.	COMPACT PAYLOAD	19
7.1	CONFIGURATION	19
7.2	API Calls	19
7.2.1	compact_payload_init	19
7.2.2	compact_payload_add_data	19
7.2.3	compact_payload_get_packet	20
7.3	NRF Connect SDK Resources	21
7.4	Dependencies	21
8.	EP BSP	21
8.1	API Calls	21
8.1.1	agora53_bsp_init ()	21
8.1.2	ep_bsp_read_battery_voltage ()	21
8.2	NRF Connect SDK Resources	21
8.3	Dependencies	22
9.	SENSOR LIBRARY	22

10. EXAMPLE PROJECTS	22
10.1 Connected Sensor	22
10.2 Additional Example	22
11. INTEGRATION	22
11.1 GENERAL INFORMATION	22
11.2 LOW POWER OPERATION	23
11.3 NRF Connect SDK	23
11.3.1 Zephyr Power Management	23
11.4 EXAMPLE FOLDER STRUCTURE	23
12. PERFORMANCE & RESOURCE USAGE	24
13. STATIC ANALYSIS RESULTS	24
APPENDIX A. SUPPORT & RESOURCES	25
APPENDIX B. LICENSES	25

1. ACRONYMS, ABBREVIATIONS, AND DEFINITIONS

Term	Description
EPI	Embedded Planet, Inc.
HTTP	Hypertext Transfer Protocol
OTA	Over-the-air Update
BSP	Board Support Package
FUOTA	Firmware Update Over-The-Air Update
DFU	Device Firmware Upgrade
epSDK	EP SDK
Zephyr	Zephyr OS and associated libraries

2. OVERVIEW

The epSDK for Zephyr provides library files for sensors and communication interfaces, significantly reducing development time and testing costs. The epSDK integration with a user's application is through simple user-friendly API calls.

The intended use of the epSDK is with EP supplied hardware and Zephyr OS running on a Nordic nRF5340 microcontroller. Users can configure the communication interfaces and drivers through compiler definitions that get passed to the libraries. Sensors, Utilities and Communication Interfaces are contained within the epSDK. Users develop their individual applications but are provided with examples to get started. Individual drivers and communication interfaces are portable to other hardware using Zephyr on an nRF5340 microcontroller. Contact EP to discuss the full epSDK and your application.

App	User Created				
Comm Interfaces	BLE	HTTP over Cellular (CatM1/NB IoT)	I2C	SPI	QSPI
	Humidity	Temperature	Pressure	Gas	GNSS
Sensors	Time of Flight	6-axis IMU			
	Compact Payload	Crypto	LED		
General					
Bootloader	mcuBoot				
OS	Zephyr OS				
Vendor Drivers	nRF Connect SDK				
Hardware	NRF5340				

Table 1: EP Zephyr Structure

3. CELLULAR LIBRARY

The cellular library uses several open-source libraries along with EP proprietary code to provide a complete cellular solution.

EP can provide demo servers on Thingsboard. Contact EP for more information or additional endpoints. The current state of the library is shown below. Contact EP for more information and customization.

The open-source libraries include:

- <http-parser>

The EP proprietary code provides the UART communication interface for the Telit ME910 modems, along with the algorithm for the cell logic. These functions include:

- Initializing cellular modem

- Registering and maintaining cellular connection
- Receiving and transmitting queue data over cellular
- Maintaining HTTP socket
- Collecting cellular diagnostics
- Maintaining GNSS fix

3.1 CONFIGURATION

The configuration settings available for modification within the cellular library are below. The configuration settings are inputs to the later to be discussed cellInit function. It is recommended to keep these defines in main.h.

Configuration Setting	Define	Values	Notes
Cellular Active	CELLULAR_ACTIVE	True False	Can be used in app, not used in libraries
Application version	VERSION_MAJOR VERSION_MINOR VERSION_BUILD	0-99 0-99 0-99	Used for display purposes on the debug UART
GNSS Enable/Disable	TELIT_GNSS_STATUS	True False	Enable or disable modem GNSS engine
GNSS attempt interval	GNSS_ATTEMPT_INTERVAL	32-bit unsigned number in seconds	Interval to which modem is given dedicated time to find GNSS fix. Relevant to units using low power as unit may power off too quickly to get a fix. As well as for Galaxis projects, as they use an ME310 modem that requires GNSS to be given priority over cellular.
GNSS attempt timeout	GNSS_TIMEOUT	32-bit unsigned number in 10s of seconds	Timeout given when attempting a GNSS fix. If no dedicated time is desired for attempting a fix, then set to 0.
GNSS minimum satellite threshold timeout	LOW_SAT_TIMEOUT	16-bit number in 10's of seconds	Timeout given for time to reach minimum number of satellites on GNSS fix attempt in 10's of seconds
GNSS minimum satellite threshold number	LOW_SAT_NUMBER	8-bit number of satellites	Number representing number of satellites to exceed minimum threshold
GNSS minimum satellite threshold status	LOW_SAT_STATUS	True False	Enable or disable GNSS minimum satellite timeout
Data Transmission Interval	MIN_TRANS_INTERVAL	Up to 32 bit unsigned number in seconds	Lets cell library know if units should be powered down between transmissions. If this define is above PWR_OFF_THRESHOLD in cell_helper, then modem will be powered off between transmissions when not using SSL/mbedtls protocols
Cellular Queue Size	CELL_QUEUE_SIZE	Limited by available memory	Cellular queue elements
Modem Sleep State	MODEM_SLEEP_STATE	SOCKET_OPEN MODEM_ON MODEM_OFF	Sleep for modem. Maintain socket connection, close connection and keep modem powered on, close connection and power down modem
Server Address and Protocol	THINGSBOARD_HTTP_INTEGRATION	Endpoint address and protocol selection	Define one of the items to enable communication over that link. epSDK support HTTP to Thingsboard. Define one option will enable other configuration settings required in main.h

Along with the configuration table above, there are additional configurations provided in the cellularTask header file. These resources shall not be modified and can be shared between the cellular library and application code. The description of these resources can be found in the header file.

3.2 API CALLS

3.2.1 cellInit()

Description

Function is responsible for initializing the cellular modem and starting the required Zephyr drivers. The function may take about 30s to return

Prototype

```
int cellInit( void )
```

Parameters

N/A

Return Value

0 Success
< 0 Failure

Example:

```
/* Initialize cell library and startup modem hardware, don't need to call on first run */
if( cellInit() != 0 ){
    LOG_ERR("Failed to initialize cell, delay and retry\r\n");
    closeCell(MODEM_SLEEP_STATE);
    k_sleep(K_MSEC(5000));
    continue;
}
```

3.2.2 registerCellular()

Description

Function is responsible for registering modem to cellular network. The function returns when there is a communication error with the modem, the registration process is successful, or the function exceeds the timeout parameter input. Activates eDRX, if available with carrier. The function will return upon an error, successful registration, or when the registration timeout has expired.

Prototype

```
int registerCellular( uint8_t cellCarrier, uint32_t regTimeout)
```

Parameters

cellCarrier: unsigned byte containing the cell carrier. Define options are located in cell_helper.h:

ELEMENT	DESCRIPTION
ATT	AT&T
VERIZON	Verizon
ROW	Rest of World
AU	Australia

regTimeout: Registration timeout in seconds

Return Value

0 Success
< 0 Failure

Example:

```
/* Enable eDRX and register cell */
if( registerCellular(CELLULAR_CARRIER, REG_TIMEOUT) != 0 ){
    LOG_ERR("Failed to start cell registration process, delay and retry\r\n");
    closeCell(MODEM_SLEEP_STATE);
    k_sleep(K_MSEC(5000));
    continue;
}
```

3.2.3 regStatus()

Description

Function to check cellular registration status. Function can be used prior to sending or receiving data over cellular.

Prototype

CellRegStatus_t regStatus(void)

Parameters

N/A

Return Value

CellRegStatus_t enum

Example:

```
/* Check registration status */
regStatusFlag = regStatus();
if(regStatusFlag != REG_STATUS_REGISTERED_HOME && regStatusFlag != REG_STATUS_ROAMING_REGISTERED){
```

Check cellular is registered or roaming before moving forward with cellular activities

3.2.4 setGNSS()

Description

Function used to set GNSS status. When set to true, GNSS activities have priority over cellular. When set to false, cellular activities have priority over GNSS.

Prototype

int setGNSS(bool status)

Parameters

status: Set GNSS priority. True: GNSS has priority over cellular. False: Cellular has priority over GNSS

Return Value

0 Success
< 0 Failure

Example:

```
/* Return back to cell comms */  
setGNSS(false);
```

3.2.5 statusGNSS()

Description

Function is used to retrieve the GNSS status.

Prototype

```
uint8_t statusGNSS( uint8_t *satellites )
```

Parameters

satellites: Number of satellite vehicles seen by modem on fix attempt

Return Value

Fix value. If fix value is greater than 1, valid fix has been obtained

Example:

```
/* Is it time to attain GNSS fix, will run on second pass */  
if(TELIT_GNSS_STATUS == true && (currentTime - lastGnssTime) > GNSS_ATTEMPT_INTERVAL){  
    if( setGNSS(true) != 0 ){  
        LOG_ERR("Failed to enable GNSS, delay and retry\r\n");  
        closeCell(MODEM_SLEEP_STATE);  
        k_sleep(K_MSEC(5000));  
        continue;  
    }  
  
    /* Check registration status, advance when registered or timeout */  
    for(uint16_t i = 0; i < GNSS_TIMEOUT; i++){  
        gnssFix = statusGNSS(&satellites);  
        if(gnssFix >= 2){  
            /* App callback to add GNSS fix if desired */  
            addGnssQueueMsg(true);  
            break;  
        }  
        /* Not enough satellites aquired after a specified amount of time, break loop */  
        if(LOW_SAT_STATUS == true && satellites < LOW_SAT_NUMBER && i == LOW_SAT_TIMEOUT){  
            LOG_WRN( ">>> GNSS aquire satellite timeout <<<\r\n" );  
            break;  
        }  
        k_sleep(K_MSEC(10000));  
    }  
  
    /* If fix failed, then add to queue*/  
    if(gnssFix < 2){  
        /* App callback to add GNSS fix if desired */  
        addGnssQueueMsg(false);  
    }  
}
```

3.2.6 connectCell()

Description

Function to open cellular connection.

Prototype

int connectCell(uint8_t *endpointAddr, bool gtpStatus)

Parameters

endpointAddr: pointer containing the complete address. Including the protocol and port address. An example of a valid address is: "http://thingsboard.cloud:80"

gtpStatus: True – Use GTP functionality on modem to obtain location through cellular, False – Do not use GTP functionality

Return Value

0 Success

< 0 Failure

Example:

```
if(connectCell(IOT_BROKER_ADDRESS_POST, GTP_STATUS) != cellStatus){  
    LOG_ERR("Failed to connect to cell socket, delay and retry\r\n");  
    closeCell(MODEM_SLEEP_STATE);  
    k_sleep(K_MSEC(5000));  
    continue;  
}
```

3.2.7 closeCell()

Description

Function is responsible for closing cellular connection

Prototype

void cellClose(uint8_t sleepMode)

Parameters

sleepMode: Variable for sleep mode option. Defines are in cell_helper.h

ELEMENT	DESCRIPTION
SOCKET_OPEN	Keep cellular socket open
MODEM_ON	Close cellular socket
MODEM_OFF	Close cellular socket Shut down modem to power

Return Value

N/A

3.2.8 sendHttpMsg()

Description

Function is responsible for sending and receiving data over cellular HTTP. When sending data, the application must place messages in the xCellQueue.

Prototype

```
int sendHttpMsg( uint8_t method, uint8_t *endpointAddr, uint8_t *pathPrefix, uint8_t* pathSuffix, bool tokenInPath, bool
*fotaAvailable )
```

Parameters

method: METHOD_POST for sending queue messages, METHOD_GET for receiving data from server

endpointAddr: pointer containing the complete address. Including the protocol and port address. An example of a valid address is: "http://thingsboard.cloud:80"

pathPrefix: path prefix, SERVER_ADDR_PREFIX_* set in main.h through setting endpoint

pathSuffix: path prefix, SERVER_ADDR_SUFFIX_* set in main.h through setting endpoint. Only used if tokenInPath variable is set to true

tokenInPath: True – access token present in path, between prefix and suffix. False – no access token used in path, path only contains prefix

fotaAvailable: True – new version detected through GET attributes. False – no new version received through GET

Return Value

0 Success

< 0 Failure

Example:

```
/* Send HTTP Post (without TLS) */
sendHttpMsg(METHOD_GET, IOT_BROKER_ADDRESS_GET, SERVER_ADDR_PREFIX_GET, SERVER_ADDR_SUFFIX_ATTR, SERVER_PATH_ACCESS_TOKEN_GET_ENABLED, &fotaAvailable);

/* Send HTTP Post (without TLS) */
sendHttpMsg(METHOD_POST, IOT_BROKER_ADDRESS_POST, SERVER_ADDR_PREFIX_POST, SERVER_ADDR_SUFFIX_TELE, SERVER_PATH_ACCESS_TOKEN_POST_ENABLED, NULL);
```

3.2.9 queryCellularDiag()

Description

Function is used by the application to query cellular diagnostics. These diagnostics can be used by the application to send through the cellular queue for advanced diagnostics on the server.

Prototype

```
int queryCellularDiag( cellDiag *params )
```

Parameters

params: Structure used to receive cellular diagnostics with structure elements:

ELEMENT	DESCRIPTION
int16_t rssi	Cellular RSSI in dBm
int16_t rsrq	Cellular RSRQ in dB
char longitude[12]	GNSS longitude in format dddmm.mmmm Bytes 0-2 are degrees: 000...180 Bytes 3-8 are minutes: 00.0000...59.9999 Byte 9 is the direction: E or W
char latitude[11]	GNSS latitude in format ddmm.mmmm Bytes 0-1 are degrees: 000...90 Bytes 2-7 are minutes: 00.0000...59.9999 Byte 8 is the direction: N or S

float long_float	Longitude represented as numerical decimal degrees format.
float lat_float	Latitude represented as numerical decimal degrees format.
uint8_t fix	Gnss fix value: 0/1 invalid fix, 2 2D fix, 3 3D fix
uint32_t fixTime	GNSS fix time in epoch seconds
uint8_t satellites	Total number of satellites in use for fix
float gtpAccuracy	GTP Accuracy of fix
float gtpLongFloat	GTP Longitude of fix
float gtpLatFloat	GTP Latitude of fix
char iccid[21]	ICCID of SIM
char imei[16]	IMEI of cellular modem

Return Value

0 Success
< 0 Failure

Example

The following example creates a structure to store the cell diagnostics and then queries them:

```
//Get latest cell values
cellDiag parameters = {'\0'};
if( queryCellularDiag(&parameters) != CELLULAR_SUCCESS )
{
    /* Do not send if unable to attain values */
    return;
}
```

3.2.10 addGnssQueueMsg

Description

Function is a callback to the application to signify a new GNSS fix is available. The function will be called after a fix attempt.

Prototype

```
void addGnssQueueMsg( bool status )
```

Parameters

status: True – passed fix attempt. False – failed fix attempt

Return Value

N/A

3.3 Cellular Queue

The cellular queue is used to send data from the application over cellular. The example sends a compact payload message into the queue. The queue handle is "xCellQueue".

An example of the queue sending sensor data is shown below.

Takes the compact payload and sends the data to the cell queue:

```
//Receive and print compact payload
cell_msg.size = compact_payload_get_packet(cell_msg.data);
LOG_INF("Payload Size: %d", cell_msg.size);
k_msgq_put(&xCellQueue, &cell_msg.data, K_SECONDS(5));
```

Set wait period between queue updates:

```
// Stop and delete the BME thread
k_thread_abort(&bme_thread_data);
sensorPwrEnConfig(false);
k_msleep(cellTransInt*1000);
sensorPwrEnConfig(true);
```

3.4 Cellular Thread

Users are recommended to create a cellular thread to handle the cellular functions. The cellular thread is configured and started as seen below:

```
/* Forward declaration of the thread function */
void cellular_task(void);

/* Define and initialize the thread statically */
K_THREAD_DEFINE(thread0_id, STACK_SIZE, cellular_task, NULL, NULL, NULL, THREAD0_PRIORITY, 0, SYS_FOREVER_MS);

int main(void)
{
    /* Start Cell Task */
    k_thread_start(thread0_id);
}
```

3.5 NRF Connect SDK Resources

The cellular library uses UART2, as defined in the *agora_cpuapp_comman.dtsi* board file. The cellular library uses the modem chat uart backend provided within the NRF Connect SDK.

Within proj.conf the following must be enabled:

- CONFIG_SERIAL=y
- CONFIG_UART_INTERRUPT_DRIVEN=y
- CONFIG_MODEM_MODULES=y
- CONFIG_MODEM_PIPE=y
- CONFIG_MODEM_CHAT=y
- CONFIG_MODEM_BACKEND_UART=y
- CONFIG_MODEM_LOG_LEVEL_DBG=y
- CONFIG_MODEM_MODULES_LOG_LEVEL_DBG=y

3.6 Dependencies

The cell library requires several other EP BSP libraries.

- Time Library: Used for GNSS and Cell to sync epoch time

4. TIME LIBRARY

The time library provides several APIs to access an RTC. The library uses the Zephyr RTC for tracking uptime. The time library works in coordination with the cell library to synchronize an epoch timer. The cell library will sync time to the GNSS time if available, otherwise the time will be pinged over cellular and synced. If the cellular library is not available or time is not available to be synced, this function can be used as a runtime timer.

The EP proprietary code contained within the library specifically:

- Initializes the timer
- Sets real time
- Reads real time

4.1 API Calls

4.1.1 set_time()

Description

The set time function sets the system time. The first call to this function initializes the RTC and external crystal. Typically, on power up this function would be called with the input parameter as 0 to initialize the time library.

Prototype

```
void set_time( uint32_t seconds)
```

Parameters

seconds: Epoch seconds since January 1, 1970.

Return Value

N/A

4.1.2 get_time_s ()

Description

Gets the epoch time or runtime in seconds.

Prototype

```
uint32_t get_time_s( void )
```

Parameters

N/A

Return Value

A uint32_t of the number of seconds since an epoch of January 1, 1970

4.1.3 get_time_ms()

Description

Gets the epoch time or runtime in milliseconds.

Prototype

uint32_t get_time_ms(void)

Parameter

N/A

Return Value

A uint32_t of the number of milliseconds since an epoch of January 1, 1970

4.1.4 Example

The example below shows a setting the initial time to 0, setting the epoch time from the GNSS fix, and reading the epoch time for the compact payload packet.

```
// Start RTC epoch timer
set_time(0);

// If fix is valid, then update epoch time
if(fix == VALID){
    set_time(epochSeconds);
}

// Get epoch time
uint32_t ts = get_time_s();
LOG_INF("TIME:%u",ts);
```

4.2 NRF Connect SDK Resources

The time library uses the Zephyr RTC.

4.3 Dependencies

The time library does not require any additional EP libraries for support.

5. LED LIBRARY

The LED library provides several APIs to access LED(s). The library provides predefined blink patterns to a background Zephyr thread that can be used in the application. The board config files provide the definition to the static library for the LED GPIOs.

The EP proprietary code contained within the library specifically:

- Initializes the LEDs
- Handles flash pattern control

5.1 API Calls

5.1.1 led_init ()

Description

The led initialization function initializes the LEDs, creates a semaphore to prevent corruption, creates a timer to control flash patterns and creates a Zephyr thread to run in the background.

Prototype

bool led_init(void)

Parameters

N/A

Return Value

Bool: True for success, false for failure.

5.1.2 led_mode()

Description

Sets the active mode of the LEDs.

Prototype

```
void led_mode( led_mode_enum mode, bool latch, uint8_t led_mask )
```

Parameters

mode: Enum to select LED mode. The definition of enum is:

```
LED_OFF,           /** < LED off */
LED_ON,            /** < LED on */
LED_ALIVE_BLINK,   /** < LED turns on for LED_ALIVE_BLINK_LENGTH ms every LED_ALIVE_BLINK_INTERVAL ms */
LED_SLOW_BLINK,    /** < LED turns on for LED_SLOW_BLINK_LENGTH ms every LED_SLOW_BLINK_INTERVAL ms */
LED_FAST_BLINK,    /** < LED turns on for LED_FAST_BLINK_LENGTH ms every LED_FAST_BLINK_INTERVAL ms */
LED_EXTRA_FAST_BLINK, /** < LED turns on for LED_EXTRA_FAST_BLINK_LENGTH ms every LED_EXTRA_FAST_BLINK_INTERVAL ms */
LED_SINGLE_BLINK,  /** < LED flashes on for LED_MULTI_BLINK_LENGTH ms once every LED_MULTI_BLINK_INTERVAL ms */
LED_DOUBLE_BLINK,  /** < LED flashes on for LED_MULTI_BLINK_LENGTH ms twice every LED_MULTI_BLINK_INTERVAL ms */
LED_TRIPLE_BLINK,  /** < LED flashes on for LED_MULTI_BLINK_LENGTH ms three times every LED_MULTI_BLINK_INTERVAL ms */
LED_QUADRUPLER_BLINK, /** < LED flashes on for LED_MULTI_BLINK_LENGTH ms four times every LED_MULTI_BLINK_INTERVAL ms */
LED_QUINTUPLE_BLINK /** < LED flashes on for LED_MULTI_BLINK_LENGTH ms five times every LED_MULTI_BLINK_INTERVAL ms */
```

latch: True will latch the mode until function is called again. False will activate the mode for one pass of the Zephyr LED thread. The mode will return to the last latched mode on the next pass of the thread.

led_mask: Binary mask to set active LEDs. Bit 0 represents the green LED. Bit 1 represents the red LED. Bit 2 represents the blue LED.

Return Value

N/A

5.1.3 led_pause ()

Description

The led pause function pauses the operation of the LED library.

Prototype

```
bool led_pause( void )
```

Parameters

N/A

Return Value

Bool: True for success, false for failure.

5.1.4 led_resume ()

Description

The led resume function restarts the LED library after it has been paused.

Prototype

bool led_pause(void)

Parameters

N/A

Return Value

Bool: True for success, false for failure.

5.1.5 Example

The example below shows a use case for the LED library within a Zephyr thread to flash the cellular thread status. A separate LED thread must be created within the application to poll the cellular status.

Start the LED app thread and LED library:

```
/* Start LED task */
led_init();
k_thread_start(led_task_id);
```

Application LED thread to pulse once for cellular successfully registered, otherwise pulse three times:

```
void LEDTask( void )
{
    for(;;)
    {
        if(regStatusFlag == REG_STATUS_REGISTERED_HOME || regStatusFlag == REG_STATUS_ROAMING_REGISTERED)
        {
            led_mode(LED_SINGLE_BLINK, true, 1);
        }
        else
        {
            led_mode(LED_TRIPLE_BLINK, true, 1);
        }

        k_sleep(K_MSEC(5000));
    }
}
```

5.2 NRF Connect SDK Resources

The LED library does not need any additional definitions in the SDK.

The LED library uses a Zephyr timer and a Zephyr thread to handle the flash patterns.

5.3 Dependencies

The LED library does not require any additional EP libraries for support.

6. BLE

The BLE library consists of one implementation:

- BLE Peripheral: The unit runs as a BLE peripheral device. Advertises and connectable with shared to GATT server. DFU services is activated.

6.1 BLE Peripheral Library

6.1.1 CONFIGURATION

There are several configuration definitions located in proj.conf.

```
CONFIG_BT=y  
CONFIG_BT_PERIPHERAL=y  
CONFIG_BT_DEVICE_NAME="AGORA53"  
CONFIG_NCS_SAMPLE_MCUMGR_BT_OTA_DFU=y
```

6.1.2 API Calls

6.1.2.1 ble_peripheral_init

Description

The function is responsible for initializing the BLE peripheral library

Prototype

```
int ble_peripheral_init ( void )
```

Parameters

N/A

Return Value

0: Success

<0: Failure

6.1.2.2 ble_peripheral_adv_start

Description

The function is responsible for starting the BLE advertising operation for the specified time

Prototype

```
void ble_peripheral_adv_start ( uint32_t seconds )
```

Parameters

seconds: Number of seconds for advertising to remain active after boot

Return Value

N/A

6.1.2.3 ble_peripheral_adv_get_status

Description

The function retrieves the advertising status

Prototype

```
bool ble_peripheral_adv_get_status ( void )
```

Parameters

N/A

Return Value

0: Advertising inactive

1: Advertising active

6.1.3 NRF Connect SDK Resources

The BLE peripheral library uses the network core of the NRF5340. The build configuration generates a net core image that contains the BLE peripheral functionality.

6.1.4 Dependencies

The BLE peripheral library does not have any dependencies.

7. COMPACT PAYLOAD

The compact payload library enables users to minimize data usage over cellular, reducing the cost for the end user. Users shall refer to document P399...024 EP_Compact_Payload for the formatting structure of the compact payload.

7.1 CONFIGURATION

The compact payload requires a configuration header in the application, `compact_payload_types.h`. The header file contains the sensor ID's and packet sizes according to the document discussed above. A table containing the slot types for the application is also included. These values shall not be modified as they are shared between the library and application.

7.2 API Calls

The following functions are available for use in the application. Other functions in the header file may be available for application use in future releases.

7.2.1 `compact_payload_init`

Description

The function is responsible for initializing the compact payload structure and all subpacket structures.

Prototype

```
void compact_payload_init(void)
```

Parameters

N/A

Return Value

N/A

Example:

```
//Initialize compact payload
compact_payload_init();
```

7.2.2 `compact_payload_add_data`

Description

Function is responsible for populate compact payload structure with desired data. Functions set flag to active for the sensor provided in the input parameter.

Prototype

EPCP_ERROR_CODE agora_add_data(EPCP_SENSOR_ID sensor_id , EPCP_SENSOR_SLOT_TYPE sensor_slot, void * sensor_data)

Parameters

sensor_id: Sensor id for data being entered.

sensor_slot: Sensor slot for data being added.

sensor_data: Pointer to send data going into compact payload.

Return Value

EPCP_ERROR_CODE:

```
typedef enum epcp_error_code{
    EPCP_SUCCESS,
    EPCP_INVALID_SENSOR,
    EPCP_INVALID_DATA_SLOT,
    EPCP_INVALID_DATA,
    EPCP_NO_ALARM_MAP,
} EPCP_ERROR_CODE;
```

Example:

The example below retrieves the latest ICM20602 sensor data, manipulates the data, and adds the data to the compact payload:

```
icm_data = icm20602_get_data(&icm, &err);
LOG_INF("ICM20602 Data:");
LOG_INF("Accel X: %2.2f, Accel Y: %2.2f, Accel Z: %2.2f", (double)icm_data.accel_x, (double)icm_data.accel_y, (double)icm_data.accel_z);
LOG_INF("Gyro X: %2.2f, Gyro Y: %2.2f, Gyro Z: %2.2f", (double)icm_data.gyro_x, (double)icm_data.gyro_y, (double)icm_data.gyro_z);
// Add ICM data to compact payload
ct.i32 = icm_data.accel_x * 100;
compact_payload_add_data(EPCP_ICM20602, EPCP_ACCEL_X, &(ct.ui32));
ct.i32 = icm_data.accel_y * 100;
compact_payload_add_data(EPCP_ICM20602, EPCP_ACCEL_Y, &(ct.ui32));
ct.i32 = icm_data.accel_z * 100;
compact_payload_add_data(EPCP_ICM20602, EPCP_ACCEL_Z, &(ct.ui32));
ct.i32 = icm_data.gyro_x * 100;
compact_payload_add_data(EPCP_ICM20602, EPCP_GYRO_X, &(ct.ui32));
ct.i32 = icm_data.gyro_y * 100;
compact_payload_add_data(EPCP_ICM20602, EPCP_GYRO_Y, &(ct.ui32));
ct.i32 = icm_data.gyro_z * 100;
compact_payload_add_data(EPCP_ICM20602, EPCP_GYRO_Z, &(ct.ui32));
```

7.2.3 compact_payload_get_packet

Description

Function builds the compact payload message. The size of the message is returned through the size parameter.

Prototype

uint16_t compact_payload_get_packet(char* ret_packet)

Parameters

ret_packet: packet to be returned

Return Value

uint16_t: size of compact payload message.

Example:

```
//Receive and print compact payload
cell_msg.size = compact_payload_get_packet(cell_msg.data);
LOG_INF("Payload Size: %d", cell_msg.size);
```

7.3 NRF Connect SDK Resources

The compact payload library requires enough memory to generate a compact payload.

7.4 Dependencies

The compact payload does not have any dependencies.

8. EP BSP

The EP BSP handles configuration of the Agora53 board. The library initializes DC/DC high voltage regulator, starts the external low frequency crystal, puts the LoRa module to sleep, and retrieves battery voltage ADC reading.

8.1 API Calls

8.1.1 agora53_bsp_init ()

Description

The init function initializes the NRF5340 power system. Peripherals are set back to their initial settings and GPIO are set as default inputs. Several GPIO are excluded and set to output low to reduce standby current. These pins are the sensor power enable and battery monitor enable. The external low frequency crystal is started and the LoRa module is put to sleep.

Prototype

```
int agora53_bsp_init( void )
```

Parameters

N/A

Return Value

0: Success

< 0: Failure

8.1.2 ep_bsp_read_battery_voltage ()

Description

Read the battery voltage ADC on the Agora board. The function enables the battery monitor GPIO pin, enables the SAADC peripheral, takes a reading, initializes the SAADC, and disables the battery monitor GPIO pin.

Prototype

```
int ep_bsp_read_battery_voltage( void )
```

Parameters

N/A

Return Value

int: represents the battery voltage in millivolts

8.2 NRF Connect SDK Resources

The BSP requires ADC to be activated within the board configuration files.

The ADC is activated in proj.conf with:

```
CONFIG_ADC=y
```

The BSP needs the following set in proj.conf to run with the low frequency clock:

```
CONFIG_CLOCK_CONTROL_NRF_K32SRC_XTAL=y
```

```
CONFIG_CLOCK_CONTROL_NRF_K32SRC_RC=n
```

8.3 Dependencies

The EP BSP library does not require any additional EP libraries for support.

9. SENSOR LIBRARY

The following sensors and chips are supported by the EP SDK:

- BME680
- HTU21D/SI7021
- ICM20602
- VL5310X

The documentation folder of the BSP repository contains readmes detailing how to integrate the sensors into the application.

10. EXAMPLE PROJECTS

The library is provided with an example project. The current support projects are based on Embedded Planet products. Additional examples may be available upon request.

10.1 Connected Sensor

The product brief and user guide for Connected Equipment can be found here: [Connected Sensor IoT Module | Embedded Planet](#)

The example project provided is compatible with the Connected Equipment hardware.

10.2 Additional Example

Contact EP for additional examples or questions.

11. INTEGRATION

11.1 GENERAL INFORMATION

EP recommends users use the following environment for the integration, as this was used to develop the libraries:

- **IDE:** Visual Studio Code V1.139.1 or greater
- **Debugger:** Segger J-Link
- **nRF Connect VS Code Extension:** v2026.7.2128
- **nRF Connect Toolchain:** v3.4.0
- **nRF Connect SDK:** v3.4.0

It is also recommended that users start with the example project for the smoothest experience getting the project and compiler running successfully. The example provides users a configured project in vscode, board configuration files, and an example application.

11.2 LOW POWER OPERATION

In the provided example application, there are several lines that enable the lowest power operation of the Agora53 device. The sleep current in the provided example is below 40uA with a 3.3v supply. Within the application, all peripherals that are enabled must be disabled when not in use. The power management Zephyr driver manages this operation automatically. When a thread needs sensor power enable, the function must be called with the true parameter. When a function does not need the pin, the function must then be called with the false parameter. If the parameter is true, the pin will be enabled. If the parameter is false, the pin will only be set low when every thread that has called the function has called a corresponding false.

11.3 NRF Connect SDK

The BSP uses NRF Connect SDK version 3.4.0. The SDK contains the NRF drivers, Zephyr OS, and Zephyr drivers.

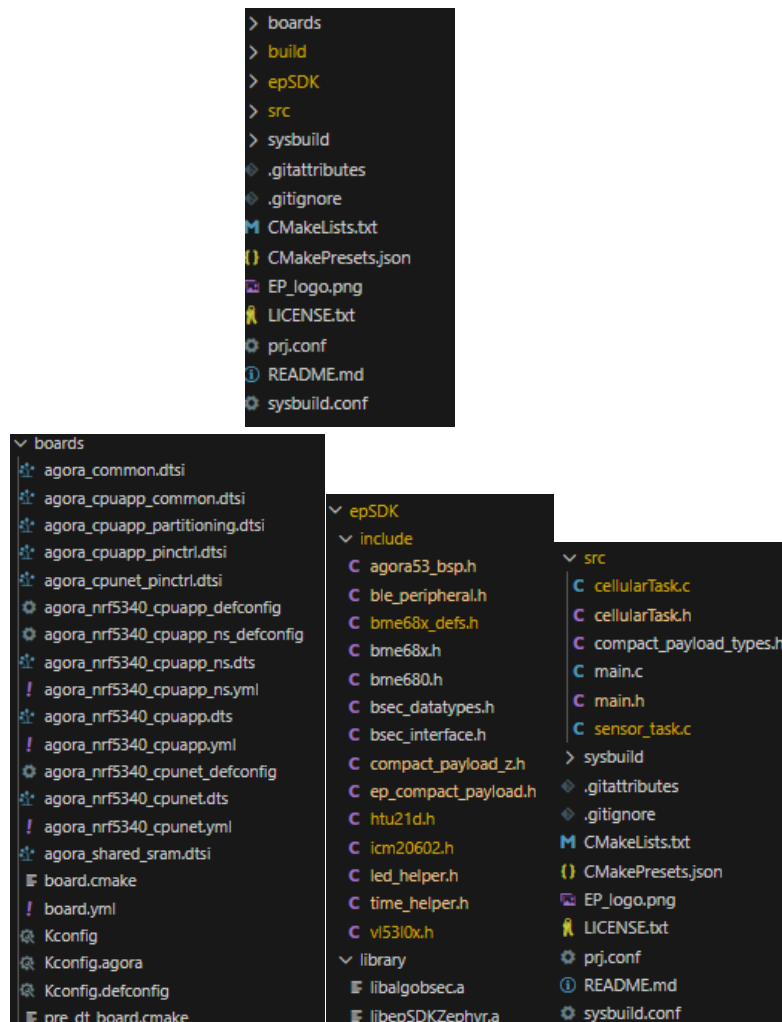
11.3.1 Zephyr Power Management

The proj.conf configuration file activates the power management utility provided by Zephyr. The following defines are used:

- CONFIG_PM=y
- CONFIG_PM_DEVICE=y
- CONFIG_PM_DEVICE_RUNTIME=y

11.4 EXAMPLE FOLDER STRUCTURE

The example provides a recommended folder structure:



The example project contains all the files necessary to build the project. It is recommended to use this project as a starting point for faster integration. The board folders contains the Agora53 peripheral configurations and memory layout. The epSDK folders contains the static library files and associated header files. The src folder contains source and header files for the application.

12. PERFORMANCE & RESOURCE USAGE

The nrf5340 contains 1MB of Flash and 512kB of RAM on the application core, along with 256kB of Flash and 64kB of RAM on the network core. The Agora53 has an additional 4MB of external flash accessible over QPSI.

The recommended memory usage is:

	Usage	Memory Range
App Core FLASH	mcuBoot (Upgradeable bootloader)	0x00000000-0x0000BFFF (48kB)
	image-0 (Application core firmware)	0x0000C000-0x000ED000 (900kB)
Net Core Flash	b0n (immutable secure bootloader)	0x00000000-0x00007FFF (32kB)
	b1 (second stage bootloader)	0x00008000-0x000087FF (2kB)
	s0 (Network core firmware)	0x00008800-0x0003FFFF (222kB)
External Flash	Application storage	0x00000000-0x002CBFF (2864kB)
	image-1 (Appcore staging area)	0x002CC000-0x003BFFFF (900kB)
	image-3 (Network core staging area)	0x003C0000-0x003FFFFFF (256kB)

13. STATIC ANALYSIS RESULTS

Contact EP to discuss.

APPENDIX A. SUPPORT & RESOURCES

For general contact, technical assistance, technical questions contact Embedded Planet at:

- Sales: sales@embeddedplanet.com
- Information Requests: info@embeddedplanet.com
- Technical Support: support@embeddedplanet.com

Additional Agora53 documentation can be found at:

<https://www.embeddedplanet.com/agora>

APPENDIX B. LICENSES

There are several licenses used with open-source libraries. The license manager table is below.

Name	Version	Manufacturer	License	Project Location
nRF Connect SDK	3.4.0	Nordic	LicenseRef-Nordic-5-Clause	C:\ncs\v3.4.0\nrf
Zephyr	4.4	Open-Source	Apache 2.0	C:\ncs\v3.4.0\zephyr\LICENSES
http-parser	N/A	NodeJS	Apache 2.0	Contained within Cellular Library, heavily modified