Embedded Planet, Inc.
31225 Bainbridge Rd Suite N
Solon, OH 44139
216-245-4180 p

User Manual

epSDK FreeRTOS User Manual

Document Revision History

Date	Description	Author	Version
08/01/2023	First Draft	M. Golob	1.0
03/05/2024	Feature enhancements	M. Golob	1.1
08/26/2025	Adding MC3479 and BLE Multirole sections	M. Golob	1.2
02/01/2026	Adding in latest projects	M. Golob	1.3
02/12/2026	Feature enhancement	M. Golob	1.4

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

TABLE OF CONTENTS

1. ACRONYMS, ABBREVIATIONS, AND DEFINITIONS 5
2. OVERVIEW 6
3. CELLULAR LIBRARY 6
3.1 CONFIGURATION 7
3.2 API CALLS 10
3.2.1 cellInit() 10
3.2.2 runCellular() 12
3.2.3 getHTTPAttributes() 13
3.2.4 queryCellularDiag() 14
3.2.5 appendMsgWithGNSS 15
3.3 Cellular Queue 15
3.4 Cellular Task 16
3.5 Cellular FOTA 16
3.6 FreeRTOS & SDK Resources 17
3.7 Makefile Setup 17
3.8 Dependencies 18
4. QPSI LIBRARY 18
4.1 API Calls 18
4.1.1 qspi_init() 18
4.1.2 qspi_uninit() 18
4.1.3 qspi_erase() 19
4.1.4 qspi_write() 19
4.1.5 qspi_read() 19
4.1.6 Example 20
4.2 FreeRTOS & SDK Resources 20
4.3 Makefile Setup 21
4.4 Dependencies 21
5. TIME LIBRARY 21
5.1 API Calls 21
5.1.1 set_time() 21
5.1.2 get_time_s () 22
5.1.3 get_time_ms() 22
5.1.4 Example 22
5.2 FreeRTOS & SDK Resources 22
5.3 Makefile Setup 23
5.4 Dependencies 23
6. LED LIBRARY 23
6.1 API Calls 23
6.1.1 led_init () 23
6.1.2 led_mode() 23
6.1.3 led_pause () 24
6.1.4 led_resume () 24
6.1.5 Example 25
6.2 FreeRTOS & SDK Resources 26
6.3 Makefile Setup 26
6.4 Dependencies 26
7. UART LIBRARY 26
7.1 API Calls 26
7.1.1 init_uart () 26
7.1.2 uninit_uart () 27
7.1.3 init_swo () 27
7.1.4 tx_enqueue () 27
7.1.5 Configuration 28
7.1.6 Example 29
7.2 FreeRTOS & SDK Resources 29
7.3 Makefile Setup 29
7.4 Dependencies 30

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

8. LORAWAN LIBRARY 30
8.1 CONFIGURATION 30
8.2 API Calls 31
8.2.1 loraConfig 31
8.3 FreeRTOS & SDK Resources 32
8.4 Makefile Setup 33
8.5 Dependencies 33
9. BLE 33
9.1 BLE Peripheral Library 33
9.1.1 CONFIGURATION 33
9.1.2 API Calls 34
9.1.3 FreeRTOS & SDK Resources 35
9.1.4 Makefile Setup 35
9.1.5 Dependencies 36
9.2 BLE Central Library 36
9.2.1 CONFIGURATION 36
9.2.2 API Calls 37
9.2.3 FreeRTOS & SDK Resources 37
9.2.4 Makefile Setup 37
9.2.5 Dependencies 39
10. OBDII & J1939 LIBRARY 39
10.1 CONFIGURATION 39
10.2 API Calls 39
10.2.1 can_init_obdii 39
10.2.2 Create_OBDII_Msg 40
10.2.3 Gather_General_OBDII_Info 40
10.2.4 Read_OBDII_Data 40
10.2.5 Update_PID_Status 41
10.2.6 can_init_j1939 42
10.2.7 Parse_J1939_Data 42
10.2.8 Create_J1939_Msg 43
10.2.9 J1939_DTC_Status_Check 43
10.2.10 J1939_Immediate_Resp 44
10.2.11 Update_SPN_Status 44
10.3 OBDII Task 45
10.4 J1939 Task 47
10.5 FreRTOS & SDK Resources 48
10.6 Makefile Setup 48
10.7 Dependencies 48
11. COMPACT PAYLOAD 48
11.1 CONFIGURATION 49
11.2 API Calls 49
11.2.1 epcp_builder_xxxxx_init 49
11.2.2 epcp_builder_xxxxx_deinit 49
11.2.3 xxxxx_add_data 50
11.2.4 xxxxx_get_packet 51
11.3 FreeRTOS & SDK Resources 51
11.4 Makefile Setup 51
11.5 Dependencies 52
12. EP BSP 52
12.1 CONFIGURATION 52
12.2 API Calls 52
12.2.1 ep_bsp_init () 52
12.2.2 ep_bsp_read_battery_voltage () 52
12.3 FreeRTOS & SDK Resources 53
12.4 Makefile Setup 53
12.5 Dependencies 53
13. OTHER COMM LIBRARIES 53
14. SENSOR LIBRARY 53

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

15. BOOTLOADER LIBRARY 53
16. EXAMPLE PROJECTS 54
16.1 Connected Assest 54
16.2 Connected Vehicle 54
16.3 Connected Equipment 54
16.4 Connected Sensor 54
16.5 Additional Example 54
17. INTEGRATION 54
17.1 GENERAL INFORMATION 54
17.2 LOW POWER OPERATION 54
17.3 NRF SDK 55
17.3.1 Low Power Modification 55
17.4 MAKEFILE 55
17.5 FREERTOS HOOKS 56
17.6 NRF SOFTDEVICE 57
17.7 EXAMPLE FOLDER STRUCTURE 58
18. PERFORMANCE & RESOURCE USAGE 60
19. STATIC ANALYSIS RESULTS 61
APPENDIX A. SUPPORT & RESOURCES 62
APPENDIX B. LICENSES 62

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

1. ACRONYMS, ABBREVIATIONS, AND DEFINITIONS

Term	Description
EPI	Embedded Planet, Inc.
MQTT	Message Queuing Telemetry Transport
HTTP	Hypertext Transfer Protocol
OTA	Over-the-air Update
SSL	Secure Sockets Layer
TLS	Transport Layer Security
BSP	Board Support Package
FOTA	Firmware Over-The-Air Update
LoRa	Long range spread spectrum modulation
CoAP	Constrained Application Protocol
epSDK	EP FreeRTOS SDK
NTN	Non-terrestrial Networks

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

2. OVERVIEW

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

The epSDK intended use is with EP supplied hardware and a FreeRTOS OS running on a Nordic NRF52840 microcontroller. Users can configure the communication interface and drivers through compiler defines that get passed to the libraries. The epSDK is compatible with the NRF52840 and the FreeRTOS kernel. A slightly modified Nordic SDK is provided with the epSDK. The epSDK contains a FreeRTOS build. The Sensors, Communication Interfaces, and Bootloader are contained within the epSDK. Users are responsible for developing the application but are provided with an example to get started, along with EP support.

Layer					
App	User Created				
Comm Interfaces	MQTT over Cellular (CatM1/NBIoT)	HTTP over Cellular (CatM1/NBIoT)	CoAP over Cellular (CatM1/NBIoT)	BLE	LoRa
EP SDK	CAN (OBDII/J1939)	SmartMesh (In Progress)	CoAP over NTN (In Progress)	I2C	(Q)SPI
Sensors	Humidity	Temperature	Pressure	Gas	Button
	GNSS	6-axis IMU	Time of Flight		
General	Watchdog	File System	Compact Payload	Crypto	LED
Bootloader	Custom EP Bootloader				
OS	FreeRTOS Kernel				
Vendor Drivers	nRF5 SDK				
Hardware	NRF52840				

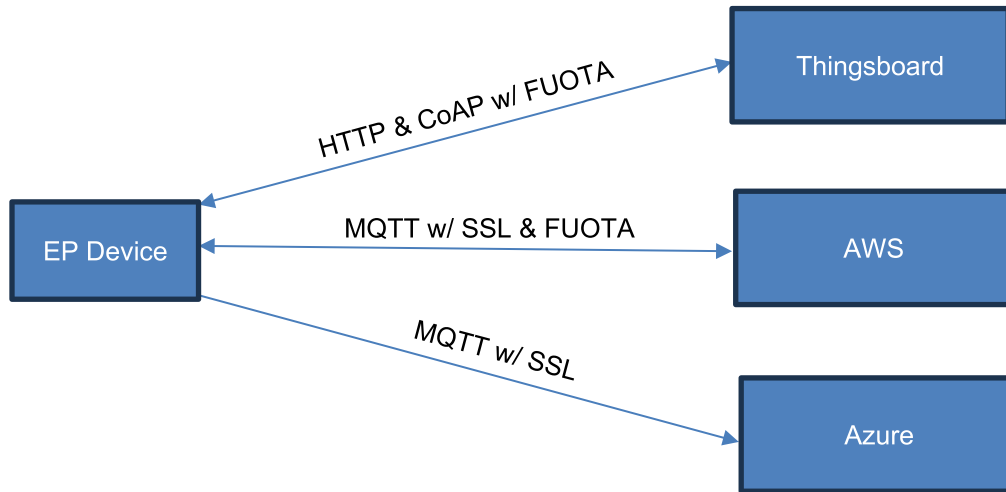
Table 1: EP FreeRTOS Structure

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, AWS IoT Core, and Azure Event Grid. Contact EP for more information or additional endpoints. The current state of the library is shown below. Contact EP for more information and customization.

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.



The open-source libraries include:

- backoffAlgorithm
- CoAP
- coreMQTT
- coreMQTT Agent
- coreHTTP
- coreJSON
- FreeRTOS Cellular Interface
- http-parser
- LoRaWAN
- mbedTLS
- OTA for AWS
- tinyCBOR

The EP proprietary code provides the UART communication interface for the Telit ME310 and 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
- Maintaining MQTT socket
- Maintaining CoAP 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_NUM VERSION_MAJOR VERSION_MINOR VERSION_BUILD	"XX.XX.XX" 0-99 0-99 0-99	Used during OTA process to check if update is present on cloud server

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

Cellular Carrier	CELLULAR_CARRIER	ATT (0) VERIZON (1) ROW (2) AU (3)	Galaxis and Agora52 SKU A will auto select carrier based on SIM, making this define meaningless.
Cellular Technology (Cat-M1 or NB-IoT)	CELLULAR_TECH	CELLULAR_IOT_CATM1 CELLULAR_IOT_NBIOT CELLULAR_IOT_CATM1_PREF CELLULAR_IOT_NBIOT_PREF	Cat-M1 NB-IoT Cat-M1 preferred over NB-IoT NB-IoT preferred over Cat-M1
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.
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 Timeout (in ticks)	CELL_QUEUE_TIMEOUT	Up to 32 bit unsigned number in ticks	Timeout for sending data to cellular queue
Cellular Queue Size	CELL_QUEUE_SIZE	Limited by available memory	Cellular queue elements
Cell queue payload format	CELL_PAYLOAD_FORMAT	JSON_PAYLOAD PLAIN_TEXT_PAYLOAD	Select between sending plain text or json messages to cloud server.
FOTA Checksum Method	otapal_CHECKSUM_METHOD	SUM_ALL CRC32_CHKSM	Sum all fota bytes in 32 bit number or perform crc on all bytes
Server Address and Protocol	THINGSBOARD_HTTP_INTEGRATION THINGSBOARD_COAP_INTEGRATION AWS_MQTT_X509 AZURE_MQTT_X509	Endpoint address and protocol selection	Define one of the items to enable communication over that link. epSDK support HTTP to Thingsboard, CoAP to Thingsboard, MQTT to AWS, and MQTT to Azure. Define one option will enable other configuration settings required in main.h
FOTA External Flash Start Address	otapal_FLASH_START	Up to 32 bit unsigned number	Used by cellular library for start location of fota image in external flash
FOTA External Flash Descriptor Table Addr	otapal_DESCRIPTOR_START	Up to 32 bit unsigned number	Used by cellular library for start location of descriptor table in external flash
FOTA External Flash Size	otapal_BANK_SIZE	Up to 32 bit unsigned number	Used by cellular library for preventing too large of FOTA image
FOTA External Flash Production Table Start	Otapal_PROD_TBL_START	Up to 32 bit unsigned number	Used by cellular library for start location of production table in external flash

Along with the configuration table above, there are additional configurations provided in the cell_helper 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.

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

Protocols with SSL utilize the following:

- mbedTLS V1.2
- x509 certificates and keys
- Options:
    - Max fragment length
    - Session tickets
    - Server name indication
    - Extended master secret
- Cipher suites:
    - MBEDTLS_TLS_PSK_WITH_AES_256_GCM_SHA384
    - MBEDTLS_TLS_PSK_WITH_AES_256_CBC_SHA384
    - MBEDTLS_TLS_PSK_WITH_AES_256_CBC_SHA
    - MBEDTLS_TLS_PSK_WITH_CAMELLIA_256_GCM_SHA384
    - MBEDTLS_TLS_PSK_WITH_CAMELLIA_256_CBC_SHA384
    - MBEDTLS_TLS_PSK_WITH_AES_128_GCM_SHA256
    - MBEDTLS_TLS_PSK_WITH_AES_128_CBC_SHA256
    - MBEDTLS_TLS_PSK_WITH_AES_128_CBC_SHA
    - MBEDTLS_TLS_PSK_WITH_CAMELLIA_128_GCM_SHA256
    - MBEDTLS_TLS_PSK_WITH_CAMELLIA_128_CBC_SHA256
    - MBEDTLS_TLS_PSK_WITH_3DES_EDE_CBC_SHA
    - MBEDTLS_TLS_PSK_WITH_RC4_128_SHA
    - MBEDTLS_TLS_ECDHE_PSK_WITH_AES_256_CBC_SHA384
    - MBEDTLS_TLS_ECDHE_PSK_WITH_AES_256_CBC_SHA
    - MBEDTLS_TLS_ECDHE_PSK_WITH_CAMELLIA_256_CBC_SHA384
    - MBEDTLS_TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA256
    - MBEDTLS_TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA
    - MBEDTLS_TLS_ECDHE_PSK_WITH_CAMELLIA_128_CBC_SHA256
    - MBEDTLS_TLS_ECDHE_PSK_WITH_3DES_EDE_CBC_SHA
    - MBEDTLS_TLS_ECDHE_PSK_WITH_RC4_128_SHA
    - MBEDTLS_TLS_RSA_WITH_AES_256_GCM_SHA384
    - MBEDTLS_TLS_RSA_WITH_AES_256_CBC_SHA256
    - MBEDTLS_TLS_RSA_WITH_AES_256_CBC_SHA
    - MBEDTLS_TLS_RSA_WITH_CAMELLIA_256_GCM_SHA384
    - MBEDTLS_TLS_RSA_WITH_CAMELLIA_256_CBC_SHA256
    - MBEDTLS_TLS_RSA_WITH_CAMELLIA_256_CBC_SHA
    - MBEDTLS_TLS_RSA_WITH_AES_128_GCM_SHA256
    - MBEDTLS_TLS_RSA_WITH_AES_128_CBC_SHA256
    - MBEDTLS_TLS_RSA_WITH_AES_128_CBC_SHA
    - MBEDTLS_TLS_RSA_WITH_CAMELLIA_128_GCM_SHA256
    - MBEDTLS_TLS_RSA_WITH_CAMELLIA_128_CBC_SHA256
    - MBEDTLS_TLS_RSA_WITH_CAMELLIA_128_CBC_SHA
    - MBEDTLS_TLS_RSA_WITH_3DES_EDE_CBC_SHA
    - MBEDTLS_TLS_RSA_WITH_RC4_128_SHA
    - MBEDTLS_TLS_RSA_WITH_RC4_128_MD5
    - MBEDTLS_TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
    - MBEDTLS_TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
    - MBEDTLS_TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
    - MBEDTLS_TLS_ECDHE_RSA_WITH_CAMELLIA_256_GCM_SHA384
    - MBEDTLS_TLS_ECDHE_RSA_WITH_CAMELLIA_256_CBC_SHA384
    - MBEDTLS_TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
    - MBEDTLS_TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
    - MBEDTLS_TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
    - MBEDTLS_TLS_ECDHE_RSA_WITH_CAMELLIA_128_GCM_SHA256
    - MBEDTLS_TLS_ECDHE_RSA_WITH_CAMELLIA_128_CBC_SHA256
    - MBEDTLS_TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
    - MBEDTLS_TLS_ECDHE_RSA_WITH_RC4_128_SHA
    - MBEDTLS_TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
    - MBEDTLS_TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
    - MBEDTLS_TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
    - MBEDTLS_TLS_ECDHE_ECDSA_WITH_CAMELLIA_256_GCM_SHA384
    - MBEDTLS_TLS_ECDHE_ECDSA_WITH_CAMELLIA_256_CBC_SHA384
    - MBEDTLS_TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
    - MBEDTLS_TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
    - MBEDTLS_TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
    - MBEDTLS_TLS_ECDHE_ECDSA_WITH_CAMELLIA_128_GCM_SHA256
    - MBEDTLS_TLS_ECDHE_ECDSA_WITH_CAMELLIA_128_CBC_SHA256
    - MBEDTLS_TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA
    - MBEDTLS_TLS_ECDHE_ECDSA_WITH_RC4_128_SHA
    - MBEDTLS_TLS_ECDH_ECDSA_WITH_RC4_128_SHA
    - MBEDTLS_TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA
    - MBEDTLS_TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA
    - MBEDTLS_TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA
    - MBEDTLS_TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256
    - MBEDTLS_TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384
    - MBEDTLS_TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256
    - MBEDTLS_TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384
    - MBEDTLS_TLS_ECDH_ECDSA_WITH_CAMELLIA_128_CBC_SHA256
    - MBEDTLS_TLS_ECDH_ECDSA_WITH_CAMELLIA_256_CBC_SHA384
    - MBEDTLS_TLS_ECDH_ECDSA_WITH_CAMELLIA_128_GCM_SHA256
    - MBEDTLS_TLS_ECDH_ECDSA_WITH_CAMELLIA_256_GCM_SHA384
    - MBEDTLS_TLS_ECDH_RSA_WITH_RC4_128_SHA
    - MBEDTLS_TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA
    - MBEDTLS_TLS_ECDH_RSA_WITH_AES_128_CBC_SHA
    - MBEDTLS_TLS_ECDH_RSA_WITH_AES_256_CBC_SHA
    - MBEDTLS_TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256
    - MBEDTLS_TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384
    - MBEDTLS_TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256
    - MBEDTLS_TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384
    - MBEDTLS_TLS_ECDH_RSA_WITH_CAMELLIA_128_CBC_SHA256
    - MBEDTLS_TLS_ECDH_RSA_WITH_CAMELLIA_256_CBC_SHA384
    - MBEDTLS_TLS_ECDH_RSA_WITH_CAMELLIA_128_GCM_SHA256
    - MBEDTLS_TLS_ECDH_RSA_WITH_CAMELLIA_256_GCM_SHA384
    - MBEDTLS_TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA
    - MBEDTLS_TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA
    - MBEDTLS_TLS_ECDH_RSA_WITH_AES_128_CBC_SHA
    - MBEDTLS_TLS_ECDH_RSA_WITH_AES_256_CBC_SHA
    - MBEDTLS_TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256
    - MBEDTLS_TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384
    - MBEDTLS_TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256
    - MBEDTLS_TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384
    - MBEDTLS_TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256
    - MBEDTLS_TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384
    - MBEDTLS_TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256
    - MBEDTLS_TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384
    - MBEDTLS_TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
    - MBEDTLS_TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
    - MBEDTLS_TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
    - MBEDTLS_TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
    - MBEDTLS_TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
    - MBEDTLS_TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
    - MBEDTLS_TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
    - MBEDTLS_TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
    - MBEDTLS_TLS_DHE_RSA_WITH_AES_256_CBC_SHA
    - MBEDTLS_TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
    - MBEDTLS_TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
    - MBEDTLS_TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
    - MBEDTLS_TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
    - MBEDTLS_TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
    - MBEDTLS_TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
    - MBEDTLS_TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
    - MBEDTLS_TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
    - MBEDTLS_TLS_DHE_RSA_WITH_AES_128_CBC_SHA
    - MBEDTLS_TLS_DHE_PSK_WITH_AES_256_GCM_SHA384
    - MBEDTLS_TLS_ECDHE_PSK_WITH_AES_256_CBC_SHA384
    - MBEDTLS_TLS_DHE_PSK_WITH_AES_256_CBC_SHA384
    - MBEDTLS_TLS_ECDHE_PSK_WITH_AES_256_CBC_SHA
    - MBEDTLS_TLS_DHE_PSK_WITH_AES_256_CBC_SHA
    - MBEDTLS_TLS_DHE_PSK_WITH_AES_128_GCM_SHA256
    - MBEDTLS_TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA256
    - MBEDTLS_TLS_DHE_PSK_WITH_AES_128_CBC_SHA256
    - MBEDTLS_TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA
    - MBEDTLS_TLS_DHE_PSK_WITH_AES_128_CBC_SHA
    - MBEDTLS_TLS_RSA_WITH_AES_256_GCM_SHA384
    - MBEDTLS_TLS_RSA_WITH_AES_256_CBC_SHA256
    - MBEDTLS_TLS_RSA_WITH_AES_256_CBC_SHA
    - MBEDTLS_TLS_RSA_WITH_AES_128_GCM_SHA256
    - MBEDTLS_TLS_RSA_WITH_AES_128_CBC_SHA256
    - MBEDTLS_TLS_RSA_WITH_AES_128_CBC_SHA
    - MBEDTLS_TLS_RSA_PSK_WITH_AES_256_GCM_SHA384
    - MBEDTLS_TLS_RSA_PSK_WITH_AES_256_CBC_SHA384
    - MBEDTLS_TLS_RSA_PSK_WITH_AES_256_CBC_SHA

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

- MBEDTLS_TLS_RSA_PSK_WITH_AES_128_GCM_SHA256
- MBEDTLS_TLS_RSA_PSK_WITH_AES_128_CBC_SHA256
- MBEDTLS_TLS_RSA_PSK_WITH_AES_128_CBC_SHA
- MBEDTLS_TLS_PSK_WITH_AES_256_GCM_SHA384
- MBEDTLS_TLS_PSK_WITH_AES_256_CBC_SHA384
- MBEDTLS_TLS_PSK_WITH_AES_256_CBC_SHA
- MBEDTLS_TLS_PSK_WITH_AES_128_GCM_SHA256
- MBEDTLS_TLS_PSK_WITH_AES_128_CBC_SHA256
- MBEDTLS_TLS_PSK_WITH_AES_128_CBC_SHA

3.2 API CALLS

3.2.1 cellInit()

Description

Function is responsible for setting all configurable parameters in the cellular library. This function needs to be the first call to the cellular library. This function is recommended to run with a cellular task.

The function is responsible for setting up the Telit modem. The process takes about 30s to complete the required power up sequence.

Within the main application code, the xCellQueue queue and cellularSemaphore semaphore must be created. Refer to the example for reference.

If the user needs the cellular protocol to use SSL/mbedTLS then nrf_crypto_init must be called prior to the FreeRTOS scheduler starting, unless the application uses the SoftDevice. The SoftDevice will handle the initialization of the crypto engine.

Prototype

Bool cellInit(cellParam *params, CellularBLEConfig_t *pdnConfig)

Parameters

pdnConfig: structure holding cell pdp setting with structure elements:

ELEMENT	DESCRIPTION
uint8_t CellularSocketPdnContextId	Context ID for cell from 1-6
CellularPdnContextType_t pdnContextType	1 (IPV4), 2 (IPV6), 3 (IPV4V6)
char apnName[65]	APN for SIM, defaults in example app to blank

params: structure holding configuration settings, with structure elements:

ELEMENT	DESCRIPTION
char appVersion[9];	Application version as defined by user. Identifies new image for FOTA and part of data package to cloud Example: 00.00.10
uint8_t carrier	Cellular carrier ATT or VERIZON
uint8_t technology	Cellular technology NB-IoT or Cat-M1
uint32_t gnssInterval	Interval for dedicated attempt of GNSS fix in seconds
uint32_t gnssTimeout	Timeout in 10s of gnss fix attempt

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

uint8_t payloadFormat	Cell queue payload format. Either Json or plain text
char brokerAddrPost[100]	Broker address for sending data to server Example: http://thingsboard.cloud:80
char brokerAddrgGet[100]	Broker address for receiving data from server Example: http://thingsboard.cloud:80
bool gnssStatus;	GNSS Status True: Enabled, False: Disabled
char serverAddrPrefixPost[50];	Server address path prefix to broker, only used if access token enabled Example: /api/v1
char serverAddrPrefixGet[50];	Server address path prefix from broker, only used if access token enabled Example: /api/v1
char serverDownloadSuffix[50];	Server path for FOTA suffix Example: /firmware?
char serverTelemSuffix[65];	Server path for telemetry suffix Example: /telemetry
char serverAttribSuffix[50];	Server path for attribute suffix Example: /attributes
bool accessTokenInPathPost	Bool for signifying if access token (IMEI) is present in server path when sending data to server
bool accessTokenInPathGet	Bool for signifying if access token (IMEI) is present in server path when receiving data from server
char rootCA1[2000];	SSL rootCA certificate
char cert[2000];	SSL certificate
char pvtKey[2000];	SSL private key
char mqttTopicString[]	Topic string name if MQTT is selected as protocol
uint8_t checksumMethod;	FOTA Checksum Method CRC32 or SUMALL (Sums every byte)
uint32_t fotaFlashStart	FOTA Start Location in External Flash Example: 0x1000
uint32_t fotaDescrTblStart	FOTA Descriptor Table Location in External Flash Example: 0x0000
uint32_t fotaBankSize	FOTA Bank Size Example: 0xAAE60
uint32_t cellOnOffPin	Cell modem ON/OFF pin

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

uint32_t cellPwrEnPin	Cell modem power enable pin
uint32_t cellRTSPin	Cell modem RTS pin

Return Value

True Success

False Failure

Example:

```
/* Initial PDP config */
CellularBLEConfig_t pdnConfig = { ATT_PDN, CELLULAR_PDN_CONTEXT_IPV4, "" };
```

```
// Set cell parameters, must be called prior to scheduler starting if using SSL/TLS due to NRF SDK incompatibilities
strncpy(cellParams.appVersion, updateVerStr, strlen(updateVerStr));
strncpy(cellParams.brokerAddrPost, IOT_BROKER_ADDRESS_POST, strlen(IOT_BROKER_ADDRESS_POST));
strncpy(cellParams.brokerAddrGet, IOT_BROKER_ADDRESS_GET, strlen(IOT_BROKER_ADDRESS_GET));
cellParams.carrier = CELLULAR_CARRIER;
strncpy(cellParams.cert, CERTIFICATE, sizeof(CERTIFICATE));
cellParams.gnssStatus = TELIT_GNSS_STATUS;
cellParams.gnssInterval = GNSS_ATTEMPT_INTERVAL;
cellParams.gnssTimeout = GNSS_TIMEOUT;
cellParams.payloadFormat = CELL_PAYLOAD_FORMAT;
strncpy(cellParams.serverAddrPrefixPost, SERVER_ADDR_PREFIX_POST, strlen(SERVER_ADDR_PREFIX_POST));
strncpy(cellParams.serverAddrPrefixGet, SERVER_ADDR_PREFIX_GET, strlen(SERVER_ADDR_PREFIX_GET));
strncpy(cellParams.serverAttribSuffix, SERVER_ADDR_SUFFIX_ATTR, strlen(SERVER_ADDR_SUFFIX_ATTR));
strncpy(cellParams.serverDownloadSuffix, SERVER_ADDR_SUFFIX_FOTA, strlen(SERVER_ADDR_SUFFIX_FOTA));
strncpy(cellParams.serverTelemSuffix, SERVER_ADDR_SUFFIX_TELE, strlen(SERVER_ADDR_SUFFIX_TELE));
cellParams.accessTokenInPathPost = SERVER_PATH_ACCESS_TOKEN_POST_ENABLED;
cellParams.accessTokenInPathGet = SERVER_PATH_ACCESS_TOKEN_GET_ENABLED;
strncpy(cellParams.pvtKey, PVT_KEY, sizeof(PVT_KEY));
strncpy(cellParams.rootCA1, ROOT_CA1, sizeof(ROOT_CA1));
strncpy(cellParams.mqttTopicString, MQTT_TOPIC, sizeof(MQTT_TOPIC));
cellParams.technology = CELLULAR_TECH;
cellParams.checksumMethod = otapal_CHECKSUM_METHOD;
cellParams.fotaFlashStart = otapal_FLASH_START;
cellParams.fotaDescrTblStart = otapal_DESCRIPTOR_START;
cellParams.fotaBankSize = otapal_BANK_SIZE;
cellParams.cellOnOffPin = CELL_ON_OFF;
cellParams.cellPwrEnPin = CELL_PWR_EN;
cellParams.cellRTSPin = CELL_RTS_PIN_NUMBER;

// Initialize cell
retCellular = cellInit( &cellParams, &pdnConfig );
if( retCellular == false )
{
    DBGI( "Cellular failed to initialize" );
}
else
{
    DBGI( "Cellular successfully initialized" );
}
```

The pdnConfig input can allow the user to have the pdn settings modified over BLE.

3.2.2 runCellular()

Description

Function is responsible for running the cellular library. The function handles initialization of the cellular connection, registering to the network, maintaining the cellular connection, powering on/off the modem, and supporting the socket data transfers. The first pass of the function initializes the modem. A return of “false” represents a failure and the function can be called again to reattempt. This function is recommended to run within a cellular task, along with cellInit. When using TLS protocols, the library uses a backoffAlgorithm to aid in preventing connection collisions at the server.

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

The function returns after attempting to send data for HTTP, and CoAP. When using MQTT, the function will only return when there is an error. RunCellular for MQTT manages and maintains the connection.

On the first pass of runCellular after initialization, the function sends the queue data and returns, providing the end user a quick response time to see data on the server. On the next pass, if GNSS is enabled, the fix will be attempted with the timing provided in cellInit.

Prototype

CellStatus_t runCellular(uint32_t cellTransInt, uint32_t regTimeout)

Parameters

cellTransInt: Cellular data transmission interval. Used for reference to determine if the modem should be powered off. If interval in seconds is less than CELL_PWR_OFF_THRESHOLD in cell_helper, modem will turn off between transmissions.

regTimeout: Time given to register to a cell network in seconds.

Return Value

CellStatus_t cellRet

The structure returns are:

```
/**
 * @ingroup cell_enum_types
 * @brief The cell library return status.
 */
typedef enum CellStatus
{
    CellGeneralFail = 0,    /*!< @brief General Failure. */
    CellSuccess,            /*!< @brief Success, no errors. */
    CellInitFail,           /*!< @brief Initialization Library Failed, or modem Init failed. */
    CellSimFail,            /*!< @brief SIM failure. */
    CellRegisterFail,       /*!< @brief Failure to register to network during initialization. */
    CellSocketFail,         /*!< @brief Failed to configure and connect to socket. */
    CellLostService,        /*!< @brief Cell service lost during runtime. */
    CellFailToSend,         /*!< @brief Failed to send data. */
    CellFailToRec           /*!< @brief Failed to receive data. */
} CellStatus_t;
```

Example

```
for(;;)
{
    /* Enter task to send periodic data */
    runCellular(cellTransInt, regTimeout );

    vTaskDelay( pdMS_TO_TICKS( ONE_DAY ) );
}
```

3.2.3 getHTTPAttributes()

Description

Function is used by the application to receive HTTP attributes (if applicable). The attributes are in the format “app_x”. “x” is the index of attribute starting at 0. At this time, users can have up to 5 attributes. Typically, the first attribute is used for setting the transmission interval.

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

Prototype
void getHTTPAttributes(uint8_t index, char* value)

Parameters
index: Index of HTTP attribute.
value: String stored for attribute.

Return Value
void

Example
The following example takes a single HTTP attribute and overwrite the cell data transmission interval:

```
/* Get HTTP Attributes here for now, Display attributes */
for( uint8_t i = 0; i < num_HTTP_ATTRIBUTES; i++ )
{
    getHTTPAttributes(i,sharedAppAttr);
    /* Check if attribute value is valid */
    if(sharedAppAttr[0] != '\0')
    {
        /* For this app, only 1 attribute used, and its for cell transmission interval, overwrite this value */
        cellTransInt = ( uint32_t ) strtoul( sharedAppAttr, NULL, 10 );
        DBGI("New Cell Transmission Interval: %ds",cellTransInt);
    }
}
```

3.2.4 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
void 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

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

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
uint8_t satellites	Total number of satellites in use for fix
uint16_t restartCounter	Counter for modem failures
char iccid[21]	ICCID of SIM
char imei[16]	IMEI of cellular modem

Return Value

CellStatus_t

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;
}
```

3.2.5 appendMsgWithGNSS

Description

Function is a callback to the application to signify a new GNSS fix is available. The function will be called when there is a message available in the queue to append. If no messages are available, then the messages will be re-checked on the next call of runCellular. The application may use the function to append the cell message in the queue.

Prototype

void appendMsgWithGNSS(void)

Parameters

N/A

Return Value

N/A

3.3 Cellular Queue

The cellular queue is used to send data from the application over cellular. The queue data can be sent in plaint text or json format. The queue handle is “xCellQueue”.

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

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

Takes the plain text payload with a compact payload and sends the data to the cell queue:

```
// Generate completed packet
cell_queue_msg local_sensor_data_msg;
local_sensor_data_msg.size = asset_get_packet(&asset_compact_payload, local_sensor_data_msg.data);

//Forward data to cell queue if there is space
if(xQueueSend(xCellQueue, &local_sensor_data_msg, CELL_QUEUE_TIMEOUT) != pdTRUE){
    DBGE("Queue Timeout!");
}
```

Set wait period between queue updates:

```
    /* Enter task every 60s */
    vTaskDelay(pdMS_TO_TICKS(cellTransInt*1000));
}

vTaskDelete( NULL );
```

3.4 Cellular Task

Users are recommended to create a cellular task to handle the cellInit and runCellular functions. The cellInit function should be called on the initial pass of the task. The runCellular function should be called within an endless loop.

3.5 Cellular FOTA

FOTA over Cellular is self-contained with the Cellular Library over HTTP and MQTT. There is no code needed to support FOTA in the application. The Cellular Library writes the image in external flash and resets the unit. It is recommended to use the EP Bootloader to handle the security checks and writing of a valid image after the cellular library reboots the microcontroller. The FOTA receives a checksum over cellular and keeps track of its own checksum during the download to external flash. If a write to external flash fails or the calculated and expected checksums do not match, the image is not marked as valid and the FOTA process will need to be restarted. The FOTA process uses the descriptor table from the configuration files to track the current image information. The structure is shown below:

```
/**
 * @brief Enum for OTA status flags
 *
 */
typedef enum
{
    otapalIMAGE_FLAG_NEW = 0xFE,           /* If the application image is running for the first time and never executed before. */
    otapalIMAGE_FLAG_VALID = 0xFC,         /* The application image is marked valid and committed. */
    otapalIMAGE_FLAG_INVALID = 0xF8        /* The application image is marked invalid. */
} ImageFlags_t;

/**
 * @brief Structure for storing descriptor tables information
 *
 */
typedef struct
{
    uint8_t usImageFlags;                  /* Image flags. */
    uint32_t ulImageStartAddress;          /* Starting address of the application image. */
    uint32_t ulImageSize;                  /* End address of the application image. */
    uint32_t ulHardwareID;                 /* 32 bit ID that can be generated unique for a particular platform. */
    uint32_t updateVersion;                /* 32 bit ID for version number */
    uint32_t checksum;                     /* 32 bit checksum. */
} ImageDescriptor_t;
```

The code will only write the descriptor table once an image download to external flash is complete and successful. The image flag alerts the bootloader to program the image in external flash if the flag is set to “NEW”. The previous descriptor table is erased at the initialization of the FOTA process.

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

3.6 FreeRTOS & SDK Resources

The cellular library uses the libUARTE0 NRF driver, along with TIMER1 and TIMER2.

For timer1 to be enabled, the following must be defined to 1 in sdk_config.h:

- NRFX_TIMER_ENABLED
- NRFX_TIMER1_ENABLED
- TIMER_ENABLED
- TIMER1_ENABLED

For timer2 to be enabled, the following must be defined to 1 in sdk_config.h:

- NRFX_TIMER_ENABLED
- NRFX_TIMER2_ENABLED
- TIMER_ENABLED
- TIMER2_ENABLED

For libUARTE0 to be enabled, the following must be defined to 1 in sdk_config.h:

- NRF_LIBUARTE_DRV_UARTE0
- NRFX_UARTE_ENABLED
- NRFX_UARTE0_ENABLED

For the crypto engine to support SSL protocols:

- NRF_CRYPTO_ENABLED
- NRF_CRYPTO_RNG_STATIC_MEMORY_BUFFERS_ENABLED
- NRF_CRYPTO_BACKEND_CC310_ENABLED
- NRF_CRYPTO_BACKEND_CC310_RNG_ENABLED
- NRF_CRYPTO_BACKEND_CC310_INTERRUPTS_ENABLED

There are several FreeRTOS tasks associated with the cellular library. The cellular task described in section 3.3 must be created in the application code with a priority of "tskIDLE_PRIORITY+2" and a stack size of at least 5000 bytes.

Within the cellular library there is a task responsible for taking the received UART data from the cell modem and passing it to the cellular API for AT command processing. This task has a stack size of 2048 bytes and a priority of "tskIDLE_PRIORITY+3". The priority of this task must be higher than the cellular task, as it's more important to ensure all UART data is received.

When the cellular protocol is set to HTTP or CoAP, there are no additional FreeRTOS tasks.

When the cellular protocol is set to MQTT, there are 2 additional FreeRTOS tasks. A low priority subscription and publication task is set to priority "tskIDLE_PRIORITY+1" with a stack size of 2000 bytes. This task must be a lower priority than the 2 previous tasks, as it is not time critical. The other task is responsible for handling the OTA process. This task has a priority of "tskIDLE_PRIORITY+2", as it's as critical as the main cellular task once an OTA is started. The task also used 2000 bytes.

3.7 Makefile Setup

The makefile for the cellular library requires the following files and folder for successful compilation. Refer to the example project provided for further clarification.

```
# Library include folders
INC_FOLDERS += \
  ../libFileHeaders/epCellHeaders \
```


Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

```
# Libraries common to all targets
LIB_FILES += \
  ../libFiles/XXXXXXX.a \
```

3.8 Dependencies

The cell library requires several other EP BSP libraries.

- BSP Library: Watchdog initialized in BSP and used to forced restart after successful FOTA
- QSPI Library: QSPI is used during FOTA to write image to external flash
- Time Library: Not required, GNSS and Cell can sync time if used
- UART Library: Not required, used for UART/SWO debug output

4. QPSI LIBRARY

The QSPI library provides several APIs to access the NRF SDK QSPI driver code. The QSPI library is used for programming an application image in external flash to program memory or for data storage in external flash. It is recommended to initialize the QSPI driver prior to the scheduler starting, so the driver is available for FOTA updates.

The EP proprietary code contained within the library specifically:

- Initializes the NRF QSPI driver
- Uninitializes the NRF QSPI driver
- Erases external flash
- Reads external flash
- Writes external flash

4.1 API Calls

4.1.1 qspi_init()

Description

The init function is used to initialize the NRF SDK QSPI driver instance. The function is responsible for configuring the QSPI settings. The function sets the external flash chip to QSPI.

Prototype

nrfx_err_t qspi_init(void)

Parameters

N/A

Return Value

nrfx_err_t: Nordic provided error flags

4.1.2 qspi_uninit()

Description

The unit function is used to uninitialize the NRF SDK QSPI driver instance.

Prototype

nrfx_err_t qspi_uninit()

Parameters

N/A

Return Value

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

nrfx_err_t: Nordic provided error flags

4.1.3 qspi_erase()

Description

The QSPI erase function is used to erase memory on the external flash chip.

Prototype

nrfx_err_t qspi_erase(qspi_erase_len_enum erase_len, uint32_t start_address)

Parameter

erase_len: Size of memory block to erase. Either 4kB, 16kB, or entire chip.
Start_address: Start address of where memory erase should occur.

Return Value

nrfx_err_t: Nordic provided error flags

4.1.4 qspi_write()

Description

The QSPI write function is used to write memory on the external flash chip.

Prototype

nrfx_err_t qspi_write(void const *tx_buffer, size_t length, uint32_t start_address)

Parameter

tx_buffer: Pointer to the start of data to be written to external flash over QSPI.
length: Length of data to write to external flash.
Start_address: Start address of where memory write should occur.

Return Value

nrfx_err_t: Nordic provided error flags

4.1.5 qspi_read()

Description

The QSPI read function reads memory from the external flash chip.

Prototype

nrfx_err_t qspi_read(void const *rx_buffer, size_t length, uint32_t start_address)

Parameter

tx_buffer: Memory location to save data read from external flash.
length: Length of data to read from external flash.
start_address: Start address of where memory write should occur.

Return Value

nrfx_err_t: Nordic provided error

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

4.1.6 Example

The example below initializes the QSPI driver, erases 64KB of external flash, writes a test string to external flash, verifies success of the write by reading external flash, and uninitializes the QSPI driver.

```
//Init the QSPI
qspi_init();

//Fill TX buffer
srand(0);
for (int i = 0; i < QSPI_TEST_DATA_SIZE; ++i)
{
    m_buffer_tx[i] = (uint8_t)rand();
}

//Erase 64KB from QSPI flash, starting at address 0.
//Other options are _4KB or _ALL
DBGI("Erasing 64KB from QSPI...");
err_code = qspi_erase(QSPI_ERASE_LEN_64KB, 0);
if(err_code){
    DBGE("Erase failure! Err: %d", err_code);
}
DBGI("Erase complete");

//Write the TX buffer to QSPI flash, starting at address 0 (multiples of 256)
DBGI("Writing %d bytes to QSPI...", QSPI_TEST_DATA_SIZE);
err_code = qspi_write(m_buffer_tx, QSPI_TEST_DATA_SIZE, 0);
if(err_code){
    DBGE("Write failure! Err: %d", err_code);
}
DBGI("Write complete");

//Read back from QSPI flash into the RX buffer, starting at address 0 (multiples of 256)
DBGI("Reading %d bytes from QSPI...", QSPI_TEST_DATA_SIZE);
err_code = qspi_read(m_buffer_rx, QSPI_TEST_DATA_SIZE, 0);
if(err_code){
    DBGE("Read failure! Err: %d", err_code);
}
DBGI("Read complete");

//Compare what was transmitted to what was received.
DBGI("Comparing TX/RX...");
if (memcmp(m_buffer_tx, m_buffer_rx, QSPI_TEST_DATA_SIZE)){
    DBGE("Failure! Data is not consistent!");

    // Uncomment this to print TX/RX buffer

    for (int i = 0; i < QSPI_TEST_DATA_SIZE; i++)
    {
        printf("TX: %d   |   RX: %d\r\n", m_buffer_tx[i], m_buffer_rx[i]);
    }

    return -1;
}
DBGI("Success! Data is consistent!");

//Uninitialize QSPI when finished
qspi_uninit();
```

4.2 FreeRTOS & SDK Resources

The QSPI library uses the QSPI NRF driver.

For QSPI to be enabled, the following must be defined in sdk_config.h:

- QSPI_ENABLED 1
- QSPI_CONFIG_SCK_DELAY 1
- QSPI_CONFIG_XIP_OFFSET 0
- QSPI_CONFIG_READOC 3
- QSPI_CONFIG_WRITEOC 2
- QSPI_CONFIG_ADDRMODE 0

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

- QSPI_CONFIG_MODE 0
- QSPI_CONFIG_FREQUENCY 0
- QSPI_PIN_SCK BSP_QSPI_SCK_PIN
- QSPI_PIN_CSN BSP_QSPI_CSN_PIN
- QSPI_PIN_IO0 BSP_QSPI_IO0_PIN
- QSPI_PIN_IO1 BSP_QSPI_IO1_PIN
- QSPI_PIN_IO2 BSP_QSPI_IO2_PIN
- QSPI_PIN_IO3 BSP_QSPI_IO3_PIN
- QSPI_CONFIG_IRQ_PRIORITY 6

There are no FreeRTOS resources required to run the QSPI library.

4.3 Makefile Setup

The makefile for the QSPI library requires the following files and folder for successful compilation. Refer to the example project provided for further clarification.

```
# Library include folders
INC_FOLDERS += \
  ../libFileHeaders/epUtilityHeaders \

# Libraries common to all targets
LIB_FILES += \
  ../libFiles/epQspiLibrary.a \
```

4.4 Dependencies

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

5. TIME LIBRARY

The time library provides several APIs to access an RTC. The library configures the external LFCLK crystal and NRF RTC driver for setting real time and receiving real time in seconds or milliseconds. The time library works in coordination with the cell library to synchronize time. 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 timer.

The EP proprietary code contained within the library specifically:

- Initializes the NRF RTC and LFCLK driver
- Sets real time
- Reads real time

5.1 API Calls

5.1.1 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.

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

Return Value

N/A

5.1.2 get_time_s ()

Description

Gets the system time 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

5.1.3 get_time_ms()

Description

Gets the system time 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

5.1.4 Example

The example below shows a FreeRTOS task setting the initial time to 0 and periodically reading the second and millisecond functions for validity.

```
void timeTrackerTask(void *pvParameters){
    DBGI("Setting time to %lus and printing value every %d seconds...", TIME_START, TIME_PRINT);
    set_time(TIME_START);
    while(true){
        DBGI("Time in seconds: %lus | Time in milliseconds: %llums", get_time_s(), get_time_ms());
        vTaskDelay(pdMS_TO_TICKS(TIME_PRINT * 1000));
    }
}
```

5.2 FreeRTOS & SDK Resources

The time library uses the RTC NRF driver.

For QSPI to be enabled, the following must be defined to 1 in sdk_config.h:

- NRFX_RTC_ENABLED
- RTC_ENABLED
- RTC2_ENABLED

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

There are no FreeRTOS resources required to run the time library.

5.3 Makefile Setup

The makefile for the time library requires the following files and folders for successful compilation. Refer to the example project provided for further clarification.

```
# Library include folders
INC_FOLDERS += \
  ../libFileHeaders/epUtilityHeaders \

# Libraries common to all targets
LIB_FILES += \
  ../libFiles/epTimeLibrary.a \
```

5.4 Dependencies

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

6. LED LIBRARY

The LED library provides several APIs to access LED(s). The library provides predefined blink patterns to a background FreeRTOS task that can be used in the application. The defines of either BOARD_AGORA or BOARD_GALAXIS in the makefile provide the LED library with the proper GPIO configurations.

The EP proprietary code contained within the library specifically:

- Initializes the LEDs
- Handles flash pattern control

6.1 API Calls

6.1.1 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 the FreeRTOS task to run in the background.

Prototype

bool led_init(void)

Parameters

N/A

Return Value

Bool: True for success, false for failure.

6.1.2 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:

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

```
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_QUADRUPLE_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 FreeRTOS LED task. The mode will return to the last latched mode on the next pass of the task.

led_mask: Binary mask to set active LEDs, using 2^(n-1), where 'n' is the LED or LEDs of interest. For example:
 * If the device supports a single LED the mask is set to 1. For three LEDs: 1,2, or 4 set the individual LEDs
 * with 3 setting LED 1 and 2, and 5, 6 and 7 setting additional combinations of the LEDs. The number of LEDs and LED list is set in the NRF SDK, in the agora.h board file.

Return Value
N/A

6.1.3 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.

6.1.4 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.

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

6.1.5 Example

The example below shows a use case for the LED library within a FreeRTOS task.

```
//Init led utility
bool init_status = led_init();
if(!init_status){
    DBGE("LED init failure!");
    return 0;
}

while(1){
    //The helper supports up to 8 different LEDs, send 0xFF
    //Will turn off all LEDs (even if less are supported)
    DBGI("LED OFF");
    led_mode(LED_OFF, true, 0xFF);
    vTaskDelay(1000);

    //Setting the latch argument to true forces a repeat of the blink pattern. Both LED_ON
    //and LED_OFF automatically latch, no matter what argument is provided.
    DBGI("LED ON");
    led_mode(LED_ON, true, ACTIVE_LEDS);
    vTaskDelay(10000);

    DBGI("LED SLOW BLINK");
    led_mode(LED_SLOW_BLINK, true, ACTIVE_LEDS);
    vTaskDelay(10000);

    DBGI("LED FAST BLINK");
    led_mode(LED_FAST_BLINK, true, ACTIVE_LEDS);
    vTaskDelay(10000);

    //Or, we can do single-shot blinks by setting the latch argument to false
    DBGI("LED ONE SLOW BLINK");
    led_mode(LED_SLOW_BLINK, false, ACTIVE_LEDS);
    vTaskDelay(5000);

    DBGI("LED ONE FAST BLINK");
    led_mode(LED_FAST_BLINK, false, ACTIVE_LEDS);
    vTaskDelay(5000);
}
```

Another example is flashing the LED based on the cellular status when the modem is powered on. The LED task in main.c is as follows:

```
static void LEDTask( void * pvParameters )
{
    led_init();

    for(;;)
    {
        if( cellStatus == CellSuccess )
        {
            led_mode(LED_CELL_OK, true, 1);
        }
        else if( cellStatus == CellFailToSend || cellStatus == CellFailToRec )
        {
            led_mode(LED_FAIL_TO_SEND, true, 1);
        }
        else
        {
            led_mode(LED_FAIL_TO_REG, true, 1);
        }

        vTaskDelay(pdMS_TO_TICKS(5000));
    }

    vTaskDelete( NULL );
}
```

The cellular task then needs to restart the LED task when sending data and pausing when powering off as follows:

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

```
for(;;)
{
    /* Enter task to send periodic data, either wait for queue entry or set delay */
    runCellular(cellTransInt, regTimeout );

    // Suspend LED task
    vTaskSuspend( ledTaskHandle );
    // Enable debug out for LED
    led_pause();
    // Shutoff any LEDs, turn to input
    nrf_gpio_cfg_default(NRF_GPIO_PIN_MAP(0, 8));

    //Sleep for 2 minutes
    vTaskDelay( pdMS_TO_TICKS( TWO_MINUTES ) );

    // Reenable LED task
    led_resume();
    vTaskResume( ledTaskHandle );
}
```

6.2 FreeRTOS & SDK Resources

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

The LED library uses a FreeRTOS timer and a FreeRTOS task to handle the flash patterns. The task size is 100 bytes with priority 1. The priority can be low, as the flashing of the LED does not need strict timing, as compared to other FreeRTOS tasks.

6.3 Makefile Setup

The makefile for the LED library requires the following files and folder for successful compilation. Refer to the example project provided for further clarification.

```
# Library include folders
INC_FOLDERS += \
  ../libFileHeaders \

# Libraries common to all targets
LIB_FILES += \
  ../libFiles/led_helper.a \
```

6.4 Dependencies

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

7. UART LIBRARY

The UART library provides several APIs to enable debug output. The debug output can be through UART or SWO. The debug output is enabled/disabled through a global flag. The UART library connects to the NRF SDK retarget library to send data through the desired output. Essentially, the library is using fprintf to print the debug output.

The EP proprietary code contained within the library specifically:

- Initialization of debug UART and/or SWO
- Connection to NRF SDK retarget code

7.1 API Calls

7.1.1 init_uart ()

Description

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

The UART initialization function initializes the UART port for application data and/or sending debug information. The function keeps track of the calling task to prevent the UART from being uninitialized when a task is using the debug UART. If the user desires to always keep the debug UART enabled, then call the init_uart with a task number, and never call the associated uart_init.

Prototype

int init_uart(uint8_t task)

Parameters

task: calling task number. The UART helper header file has a list of defines for the main loop along with 4 tasks that may be inputs to init_uart. The cellular library uses TASK_2, while the others are available for the application to use.

Return Value

int: 0 if successful, otherwise error code

7.1.2 uninit_uart ()

Description

The UART uninitialization function is to enable the lowest power operation. If there are no remaining tasks with have called init without uninit, the debug UART is disabled and the communication lines are set to inputs.

Prototype

void uninit_uart(uint8_t task)

Parameters

task: calling task number. The UART helper header file has a list of defines for the main loop along with 4 tasks that may be inputs to init_uart. The cellular library uses TASK_2, while the others are available for the application to use.

Return Value

N/A

7.1.3 init_swo ()

Description

The SWO initialization function initializes the SWO port for application data and/or sending debug information.

Prototype

int init_swo(void)

Parameters

N/A

Return Value

int: 0 if successful, otherwise error code

7.1.4 tx_enqueue ()

Description

Function to enter debug data into tx queue for transmission over UART and/or SWO.

Prototype

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

void tx_enqueue(const char* ansi_color, const char* msg_type, const char* func, int line, const char* format, ...)

Parameters

ansi_color: An ANSI defined color. The recommended options are below:

```
//Color escape definitions for printing
//Normal colors (suffix N)
#define ANSI_COLOR_REDN "\033[0;31m"
#define ANSI_COLOR_GRNN "\033[0;32m"
#define ANSI_COLOR_YELN "\033[0;33m"
#define ANSI_COLOR_BLUN "\033[0;34m"
#define ANSI_COLOR_MAGN "\033[0;35m"
#define ANSI_COLOR_CYNN "\033[0;36m"
//Bold colors (suffix B)
#define ANSI_COLOR_REDB "\033[1;31m"
#define ANSI_COLOR_GRNB "\033[1;32m"
#define ANSI_COLOR_YELB "\033[1;33m"
#define ANSI_COLOR_BLUB "\033[1;34m"
#define ANSI_COLOR_MAGB "\033[1;35m"
#define ANSI_COLOR_CYNB "\033[1;36m"
//Set to normal mode (remove color)
#define ANSI_COLOR_RST "\x1b[0m"
```

msg_type: Message type level. Recommended options are Info, Warning, or Error:

- Info: “[INF]”
- Warning: “[WRN]”
- Error: “[ERR]”

func: Add function to debug header. C define “__func__” macro used to add calling function to header.

line: Add line number to debug header. C defined “__LINE__” macro used to add calling line to header.

format: Add function arguments to message body. C defined “__VA_ARGS__” macro used to add calling arguments to message body.

Return Value

N/A

7.1.5 Configuration

Several defines are in the header to provide an example on a possible method of using the debug output. The Warning and Error defines are enabled by default but can be set to true in the application to ensure the expected operation. The Info needs to be enabled by the application.

#define DBGI: If the application enables the info debug output (uart_helper.dbgi = true), the debug output will print an “INF” message with no color.

#define DBGW: If the application enables the warning debug output(uart_helper.dbgw = true), the debug output will print a “WRN” message with Blue text.

#define DBGE: If the application enables the error debug output(uart_helper.dbge = true), the debug output will print a “ERR” message with Red text.

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

The debug information can contain additional information: function file and line number (full), function and line number (compact),or just the debug text (none) by selecting the header style:

uart_helper.dbg_header_style = DEBUG_HEADER_FULL;

uart_helper.dbg_header_style = DEBUG_HEADER_COMPACT;

uart_helper.dbg_header_style = DEBUG_HEADER_NONE;

7.1.6 Example

The example below shows a use case for the debug library.

```
//Init uart for program output
init_uart();
//Init SWO for program output
init_swo();

DBGI("Testing DBGI (this should not print!)");
DBGW("Testing DBGW");
DBGE("Testing DBGE");
uart_helper.dbgi = true;
DBGI("Testing DBGI");
```

7.2 FreeRTOS & SDK Resources

The UART library uses several NRF SDK resources. The UART and SWO drivers are required, along with the SDK retarget code. For debug UART to be enabled, the following must be defined to 1 in sdk_config.h:

- RETARGET_ENABLED
- At least 1 of the following
    - SEND_LOG_OVER_UART
    - SEND_LOG_OVER_SWO

The library also uses LibUARTE1, along with timers 3 and 4.

For timer3 to be enabled, the following must be defined to 1 in sdk_config.h:

- NRFX_TIMER_ENABLED
- NRFX_TIMER3_ENABLED
- TIMER_ENABLED
- TIMER3_ENABLED

For timer4 to be enabled, the following must be defined to 1 in sdk_config.h:

- NRFX_TIMER_ENABLED
- NRFX_TIMER4_ENABLED
- TIMER_ENABLED
- TIMER4_ENABLED

For libUARTE1 to be enabled, the following must be defined to 1 in sdk_config.h:

- NRF_LIBUARTE_DRV_UARTE1
- NRFX_UARTE_ENABLED
- NRFX_UARTE1_ENABLED

The debug output is sent to retarget using a FreeRTOS queue.

7.3 Makefile Setup

The makefile for the UART library requires the following files and folder for successful compilation. Refer to the example project provided for further clarification.

```
# Library include folders
```


Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

```
INC_FOLDERS += \
  ../libFileHeaders/epUtilityHeaders \

# Libraries common to all targets
LIB_FILES += \
  ../libFiles/epDebugUartLibrary.a \
```

7.4 Dependencies

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

8. LORAWAN LIBRARY

The LoRaWAN library enables users to quickly run a LoRaWAN stack by creating a task and sending data through a queue. The stack is responsible for configuring the Semtech modem, initializing the LoRa library, joining a LoRaWAN network, sending queue data over LoRa, and receiving downlink data over LoRa.

8.1 CONFIGURATION

The modifiable configuration settings available for the LoRa library are below:

Configuration Setting	Define	Values
Application LoRa status flag	LORA_ACTIVE	Bool: true-LoRa active, false-LoRa inactive
LoRaWAN Region	LORAWAN_REGION	LORA_REGION_AS923 LORA_REGION_AU915 LORA_REGION_CN470 LORA_REGION_CN779 LORA_REGION_EU433 LORA_REGION_EU868 LORA_REGION_KR920 LORA_REGION_IN865 LORA_REGION_US915 LORA_REGION_RU864
LoRaWAN downlink port	LORAWAN_APP_PORT	uint16_t port number
Confirmed messages status	LORAWAN_CONFIRMED_SEND	Bool: true-confirmed messages, false-unconfirmed messages
Jitter between transmissions	LORAWAN_APPLICATION_JITTER_MS	int32_t jitter in ms
Max wait time for receive downlink	CLASSA_RECEIVE_WINDOW_DURATION_MS	uint32_t wait time in ms
Max queue timeout	LORA_QUEUE_TIMEOUT	uint32_t queue wait in ms
Max number of queue elements	LORA_QUEUE_SIZE	uint8_t queue elements
Join attempts	lorawanConfigMAX_JOIN_ATTEMPTS	uint8_t join attempts

The provided LoRaWAN header file contains variables and prototypes that can be used by the application but may not be modified. The library currently supports OTAA for devices join a LoRa server.

The example source code provides a recommended path for users to store the LoRa keys in external flash. The structure to store these keys is labeled in the production table as seen below. The table gives space during production to store the unit serial number, hardware identifier, part number, along with the LoRa parameters of sub-band, device EUI, join EUI, and application key.

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

```
typedef struct
{
    uint8_t  serialNumber[6];           /* 8 byte serial number */
    uint32_t ulHardwareID;              /* 32 bit ID that can be generated unique for a particular platform. */
    uint8_t  partNumber[6];             /* 6 byte part number, ie. 800263 for P800000000263. */
    uint8_t  loraSubband;               /* 8 bit lora sub-band */
    uint8_t  loraDevEUI[8];             /* 8 byte app key for lora */
    uint8_t  loraJoinEUI[8];            /* 8 byte app key for lora */
    uint8_t  loraAppKey[16];            /* 16 byte app key for lora */
} ProductionTable_t;
```

The table is read on external flash during power up. If there are no valid values receives, the defaults for the LoRa parameters are used in the main header.

8.2 API Calls

8.2.1 loraConfig

Description

Function is responsible for setting all configurable parameters in the LoRa library. This function needs to be the first call to the LoRa library. This function is recommended to run within the miscellaneous initialization.

The function is responsible for setting up the LoRa modem, by setting the required GPIO and SPI peripherals.

Prototype

Bool loraConfig(loraParam *loraParams)

Parameters

loraParams: structure holding configuration settings, with structure elements:

ELEMENT	DESCRIPTION
uint8_t region	Region for LoRa communication. List provided as part of LORWAN_REGION definition notes.
uint16_t appPort	Application port for downlink data.
bool confSend	Confirmed send of transmit: true Unconfirmed send: false
int32_t jitterMS	Jitter in milliseconds between transmission attempts.
uint32_t rxWindowMS	Receive window timeout in milliseconds.
uint32_t txIntervalSec	Time between transmit interval cycles in seconds.
uint32_t mosi	MOSI pin of modem Example: NRF_GPIO_PIN_MAP(0,11)
uint32_t miso	MISO pin of modem Example: NRF_GPIO_PIN_MAP(0,12)
uint32_t sck	SCK pin of modem Example: NRF_GPIO_PIN_MAP(0,7)

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

Uint8_t subBand	LoRa sub-band
Uint8_t devEUI[8]	Device EUI
Uint8_t joinEUI[8]	Join EUI
Uint8_t appKey[16]	Application key
Uint8_t appSessionKey[16]	Application session key. Not currently used.
Uint8_t nwkSessionKey[16]	Network session key. Not currently used.

Return Value

True Success

False Failure

Example:

```
/* Initialize LoRa parameters */
loraParam loraParams = {0};
// Set lora parameters, must be called prior to scheduler starting if using SSL/TLS due to NRF SDK incompatibilities
loraParams.region = LORAWAN_REGION;
loraParams.appPort = LORAWAN_APP_PORT;
loraParams.confSend = LORAWAN_CONFIRMED_SEND;
loraParams.jitterMS = LORAWAN_APPLICATION_JITTER_MS;
loraParams.rxWindowMS = CLASSA_RECEIVE_WINDOW_DURATION_MS;
loraParams.txIntervalSec = cellTransInt;
loraParams.mosi = SER_CON_SPIS_MOSI_PIN;
loraParams.miso = SER_CON_SPIS_MISO_PIN;
loraParams.sck = SER_CON_SPIS_SCK_PIN;
loraParams.subBand = subbandDefault;
loraParams.maxJoinAttempts = lorawanConfigMAX_JOIN_ATTEMPTS;
loraParams.commInterface = comm_conf;
memcpy(loraParams.devEUI, devEUIDefault, sizeof(devEUI));
memcpy(loraParams.joinEUI, joinEUIDefault, sizeof(joinEUI));
memcpy(loraParams.appKey, appKeyDefault, sizeof(appKey));

// Initialize LoRa parameters
if( loraConfig( loraParams ) == false )
{
    DBGI( "LoRa parameters failed to initialize" );
}
else
{
    DBGI( "LoRa parameters successfully initialized" );
}
```

8.3 FreeRTOS & SDK Resources

The LoRa library uses several NRF SDK resources.

The library uses GPIOTE, for the LoRa modem interrupt line DIO0 and SPI2 for the communication lines.
For GPIOTE to be enabled, the following must be defined to 1 in sdk_config.h:

- GPIOTE_ENABLED
- NRFX_GPIOTE_ENABLED

For SPI2 to be enabled, the following must be defined to 1 in sdk_config.h:

- SPI_ENABLED
- SPI2_ENABLED

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

- SPI2_USE_EASY_DMA_ENABLED

The LoRa library is run by creating the vLorawanClassATask in main.c as seen below:

```
xTaskCreate( vLorawanClassATask, "LoRaWanClassA", LORAWAN_CLASSA_TASK_STACK_SIZE, NULL, LORAWAN_CLASSA_TASK_PRIORITY, NULL );
```

The task handles initializing the Semtech LoRaMAC library, connecting, transmitting, receiving, and disconnecting from the LoRa network. The task provides debug information on the status of the LoRa connection. After every transmit, the library will attempt a downlink. Once the queue is empty, the task waits for the txIntervalSec seconds before checking the queue for message again.

The internal LoRaMAC task runs with a task size of 2048 bytes and a priority of configMAX_PRIORITIES-1. The priority of the LoRaMAC task shall be higher than the vLorawanClassATask.

8.4 Makefile Setup

The makefile for the LoRa library requires the following files and folders for successful compilation. Refer to the example project provided for further clarification.

```
# Library include folders
INC_FOLDERS += \
  ../libFileHeaders/epLoRaHeaders \

# Libraries common to all targets
LIB_FILES += \
  ../libFiles/epXXLibrary.a \
```

The LoRa library requires additional flags in the makefile. The following must be added the C and Assembler flags:

- DREGION_US915
- DLORAWAN_USE_EXTERNAL_TIMERS

8.5 Dependencies

The LoRa library requires several other EP BSP libraries.

- BSP Library: Watchdog initialized in BSP
- QSPI Library: QSPI is recommended to store keys on external flash
- UART Library: Not required, used for UART/SWO debug output

9. BLE

The BLE library consists of two implementations:

- BLE Peripheral: The unit runs as a BLE peripheral device. Advertises data and connectable with shared to GATT server
- BLE Central: The unit runs as a BLE central device. Allows for connections to BLE peripherals.

9.1 BLE Peripheral Library

9.1.1 CONFIGURATION

There are several configuration definitions located in main.h. Generally, if these values need to be modified, reach out to EP for a new static library build.

```
#define BLE_ACTIVE                    1        // 0 = Deactivated, 1 = Activated
#define BLE_TX_POWER_BOOST            8        // TX power
#define DEVICE_NAME                   "EP-DEV" // Name of device. Will be included in the advertising data.
#define BLE_SERVICE_SYSTEM            true     // Enable the system service
#define BLE_SERVICE_ICM20602          true     // Enable the ICM20602 service
#define MANUFACTURER_ID               0xFFFF   // Manufacturer ID
```


Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

```
#define DEVICE_APPEARANCE                  0x0540   // Device appearance to add to adv data. 0x0540 is generic sensor, see BLE
spec for full lis
#define APP_ADV_INTERVAL                   10000    // The advertising interval in units of 0.625 ms
#define APP_ADV_DURATION                   3000     // The advertising duration in units of 10 milliseconds
#define APP_ADV_WAIT                       0        // Ignored, peripheral is always advertising
#define APP_CONT_ADV                       0        // Ignored, peripheral is always advertising
#define MIN_CONN_INTERVAL                  MSEC_TO_UNITS(100, UNIT_1_25_MS)  // Minimum acceptable connection interval
#define MAX_CONN_INTERVAL                  MSEC_TO_UNITS(200, UNIT_1_25_MS)  // Maximum acceptable connection interval
#define SLAVE_LATENCY                      0
#define CONN_SUP_TIMEOUT                   MSEC_TO_UNITS(4000, UNIT_10_MS)   // Connection supervisory timeout (4 seconds).
#define FIRST_CONN_PARAMS_UPDATE_DELAY     5000     // ms from init evt to 1st time sd_ble_gap_conn_param_update is called
#define NEXT_CONN_PARAMS_UPDATE_DELAY      30000    // ms between each call to sd_ble_gap_conn_param_update after the first call
#define MAX_CONN_PARAMS_UPDATE_COUNT       3        // Number of attempts before giving up the connection parameter negotiation
```

9.1.2 API Calls

9.1.2.1 ep_ble_peripheral

Description

The function is responsible for initializing the BLE stack, GAP parameters, GATT module, advertising initialization and starting

Prototype

void ep_ble_peripheral (char *devName, uint8_t devNameSize, char *epSerial)

Parameters

devName: Pointer to device name of BLE peripheral

devNameSize: size of device name

epSerial: Pointer to EP serial number of device

Return Value

N/A

9.1.2.2 ep_ble_beacon_update

Description

This function is responsible for updating the data in the advertising packet

Prototype

void ep_ble_beacon_update(uint8_t data[6], uint8_t ep_serial[7])

Parameters

data: 6 bytes of user defined data to include in advertising packet

ep_serial: EP serial number of device

9.1.2.3 ble_characteristic_update_xxx

Description

This function is responsible for updating data in the corresponding UUID of the GATT server. The “xxx” can be replaced with “system”, “bme680”, or “icm20602

Prototype

void ble_characteristic_update_system(ble_serv_system *system_service, system_service_data *new_data)

void ble_characteristic_update_icm20602(ble_serv_icm20602 *icm20602_service, icm20602_service_data *new_data)

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

void ble_characteristic_update_bme680(ble_serv_bme680 *bme680_service, bme680_service_data *new_data)

Parameters

xxxxxx_service: structure for service

new_data: new data to be pushed to service

9.1.3 FreeRTOS & SDK Resources

The BLE peripheral library uses several NRF SDK resources.

The following must be defined to 1 in sdk_config.h:

- BLE_ADVERTISING_ENABLED
- BLE_DB_DISCOVERY_ENABLED
- NRF_BLE_CONN_PARAMS_ENABLED
- NRF_BLE_GATT_ENABLED
- NRF_BLE_GQ_ENABLED
- NRF_BLE_QWR_ENABLED
- NRF_BLE_SCAN_ENABLED
- NRF_BLE_SCAN_FILTER_ENABLED
- PEER_MANAGER_ENABLED
- PM_CENTRAL_ENABLED

The FreeRTOS Kernel must be started for the BLE peripheral functionality to properly operate.

9.1.4 Makefile Setup

The makefile for the peripheral library requires the following files and folders for successful compilation. Refer to the example project provided for further clarification.

```
# Required Source files
SRC_FILES += \
  $(SDK_ROOT)/components/ble/peer_manager/auth_status_tracker.c \
  $(SDK_ROOT)/components/ble/common/ble_advdata.c \
  $(SDK_ROOT)/components/ble/ble_advertising/ble_advertising.c \
  $(SDK_ROOT)/components/ble/common/ble_conn_params.c \
  $(SDK_ROOT)/components/ble/common/ble_conn_state.c \
  $(SDK_ROOT)/components/ble/common/ble_srv_common.c \
  $(SDK_ROOT)/components/ble/peer_manager/gatt_cache_manager.c \
  $(SDK_ROOT)/components/ble/peer_manager/gatts_cache_manager.c \
  $(SDK_ROOT)/components/ble/peer_manager/id_manager.c \
  $(SDK_ROOT)/components/ble/nrf_ble_gatt/nrf_ble_gatt.c \
  $(SDK_ROOT)/components/ble/nrf_ble_qwr/nrf_ble_qwr.c \
  $(SDK_ROOT)/components/ble/peer_manager/peer_data_storage.c \
  $(SDK_ROOT)/components/ble/peer_manager/peer_database.c \
  $(SDK_ROOT)/components/ble/peer_manager/peer_id.c \
  $(SDK_ROOT)/components/ble/peer_manager/peer_manager.c \
  $(SDK_ROOT)/components/ble/peer_manager/peer_manager_handler.c \
  $(SDK_ROOT)/components/ble/peer_manager/pm_buffer.c \
  $(SDK_ROOT)/components/ble/peer_manager/security_dispatcher.c \
  $(SDK_ROOT)/components/ble/peer_manager/security_manager.c \
  $(SDK_ROOT)/components/softdevice/common/nrf_sdh.c \
  $(SDK_ROOT)/components/softdevice/common/nrf_sdh_ble.c \
  $(SDK_ROOT)/components/softdevice/common/nrf_sdh_freertos.c \
  $(SDK_ROOT)/components/softdevice/common/nrf_sdh_soc.c \


# Library include folders
```


Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

```
SRC_FOLDERS += \
  ../libFileHeaders/epUtilityHeaders \
  $(SDK_ROOT)/components/softdevice/common \
  $(SDK_ROOT)/components/softdevice/s140/headers \
  $(SDK_ROOT)/components/softdevice/s140/headers/nrf52 \
  $(SDK_ROOT)/components/toolchain/cmsis/include \
  $(SDK_ROOT)/components/ble/ble_advertising \
  $(SDK_ROOT)/components/nfc/ndef/connection_handover/ble_oob_advdata \
  $(SDK_ROOT)/components/nfc/ndef/conn_hand_parser/ble_oob_advdata_parser \
  $(SDK_ROOT)/components/ble/ble_services/ble_ancs_c \
  $(SDK_ROOT)/components/ble/ble_services/ble_ias_c \
  $(SDK_ROOT)/components/ble/ble_services/ble_gls \
  $(SDK_ROOT)/components/libraries/bootloader/ble_dfu \
  $(SDK_ROOT)/components/ble/ble_dtm \
  $(SDK_ROOT)/components/ble/peer_manager \
  $(SDK_ROOT)/components/ble/nrf_ble_gatt \
  $(SDK_ROOT)/components/ble/nrf_ble_qwr \
  $(SDK_ROOT)/components/nfc/ndef/connection_handover/ble_pair_lib \
  $(SDK_ROOT)/components/ble/ble_racp \
  $(SDK_ROOT)/components/ble/common \

# Libraries common to all targets
LIB_FILES += \
  ../libFiles/epXXPeripheralLibrary.a \
```

The BLE peripheral library requires additional flags in the makefile. The following must be added the C and Assembler flags:

- DSOFTDEVICE_PRESENT
- DNRF_SD_BLE_API_VERSION=7
- DNRF_SDH_BLE_CENTRAL_LINK_COUNT=0
- DNRF_SDH_BLE_PERIPHERAL_LINK_COUNT=1
- DNRF_SDH_BLE_TOTAL_LINK_COUNT=1

9.1.5 Dependencies

The BLE peripheral library requires several other EP BSP libraries.

- UART Library: Used for UART/SWO debug output

9.2 BLE Central Library

9.2.1 CONFIGURATION

There are several configuration definitions located in main.h. Generally, if these values need to be modified, reach out to EP for a new static library build.

```
#define BLE_ACTIVE                          1        // 0 = Deactivated, 1 = Activated
#define BLE_TX_POWER_BOOST                  0        // BLE TX power boost in dBm
#define BLE_SERVICE_AGORA                   true     // Enable the AGORA service
#define CONNECT_IF_MATCH                    true     // Should be set to true to allow connection with EP BLE Peripheral devices
#define TARGET_PERIPH_NAME_COUNT            1        // Number of possible device types/names that the central can connect to
#define TARGET_PERIPH_ADDR_COUNT            1        // Number of possible device addresses that the central can connect to
/** < BLE peripheral advertising addresses (LSBF) that the central will try to connect to */
#define TARGET_PERIPH_NAMES ((const char*[TARGET_PERIPH_NAME_COUNT])   {"EP_DEV"}
#define AGORA_CONNECTIONS_MAX               3        // Number of Agora peripherals that can be connected
#define AGORA_CHARS_MAX                     160      // Max number of characteristics retrieved from peripheral
#define BLE_DATA_AGGREGATOR                 BLE_TO_EPCP      // Convert periphal data to EP compact payload format
#define BLE_EPCP_QUEUE_SIZE                 3        // Number of items allows in peripheral data queue
#define BLE_EPCP_QUEUE_TIMEOUT              pdMS_TO_TICKS(3000)       // Timeout for placing data in peripheral queue
```


Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

9.2.2 API Calls

9.2.2.1 ep_ble_central_init

Description

The function is responsible for initializing the BLE stack, scanning, GAP parameters, GATT module, connection parameters, database discovery, connection state, services, peer manager, advertising, and the NRF SoftDevice within FreeRTOS.

Prototype

void ep_ble_central_init (bool connect_if_match)

Parameters

connect_if_match: true-allow peripheral connections, false-prevent peripheral connections

Return Value

N/A

9.2.3 FreeRTOS & SDK Resources

The BLE central library uses several NRF SDK resources.

The following must be defined to 1 in sdk_config.h:

- BLE_ADVERTISING_ENABLED
- BLE_DB_DISCOVERY_ENABLED
- NRF_BLE_CONN_PARAMS_ENABLED
- NRF_BLE_GATT_ENABLED
- NRF_BLE_GQ_ENABLED
- NRF_BLE_QWR_ENABLED
- NRF_BLE_SCAN_ENABLED
- NRF_BLE_SCAN_FILTER_ENABLED
- PEER_MANAGER_ENABLED
- PM_CENTRAL_ENABLED

The FreeRTOS Kernel must be started for the BLE central functionality to properly operate.

9.2.4 Makefile Setup

The makefile for the LoRa library requires the following files and folders for successful compilation. Refer to the example project provided for further clarification.

```
# Library include folders
SRC_FOLDERS += \
  $(SDK_ROOT)/components/ble/ble_advertising/ble_advertising.c \
  $(SDK_ROOT)/components/ble/ble_db_discovery/ble_db_discovery.c \
  $(SDK_ROOT)/components/ble/ble_services/ble_lbs_c/ble_lbs_c.c \
  $(SDK_ROOT)/components/ble/common/ble_advdata.c \
  $(SDK_ROOT)/components/ble/common/ble_conn_state.c \
  $(SDK_ROOT)/components/ble/common/ble_srv_common.c \
  $(SDK_ROOT)/components/ble/nrf_ble_gatt/nrf_ble_gatt.c \
  $(SDK_ROOT)/components/ble/nrf_ble_gq/nrf_ble_gq.c \
  $(SDK_ROOT)/components/ble/nrf_ble_scan/nrf_ble_scan.c \
  $(SDK_ROOT)/components/softdevice/common/nrf_sdh.c \
  $(SDK_ROOT)/components/softdevice/common/nrf_sdh_ble.c \
  $(SDK_ROOT)/components/softdevice/common/nrf_sdh_freertos.c \
  $(SDK_ROOT)/components/softdevice/common/nrf_sdh_soc.c \
```


Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

```
# Libraries common to all targets
LIB_FILES += \
  ../libFiles/epXXCentralLibrary.a \

INC_FILES += \
  ../libFileHeaders/epUtilityHeaders \
  $(SDK_ROOT)/components \
  $(SDK_ROOT)/components/ble/ble_advertising \
  $(SDK_ROOT)/components/ble/ble_db_discovery \
  $(SDK_ROOT)/components/ble/ble_dtm \
  $(SDK_ROOT)/components/ble/ble_racp \
  $(SDK_ROOT)/components/ble/ble_services/ble_ancs_c \
  $(SDK_ROOT)/components/ble/ble_services/ble_ans_c \
  $(SDK_ROOT)/components/ble/ble_services/ble_bas \
  $(SDK_ROOT)/components/ble/ble_services/ble_bas_c \
  $(SDK_ROOT)/components/ble/ble_services/ble_cscs \
  $(SDK_ROOT)/components/ble/ble_services/ble_cts_c \
  $(SDK_ROOT)/components/ble/ble_services/ble_dfu \
  $(SDK_ROOT)/components/ble/ble_services/ble_dis \
  $(SDK_ROOT)/components/ble/ble_services/ble_gls \
  $(SDK_ROOT)/components/ble/ble_services/ble_hids \
  $(SDK_ROOT)/components/ble/ble_services/ble_hrs \
  $(SDK_ROOT)/components/ble/ble_services/ble_hrs_c \
  $(SDK_ROOT)/components/ble/ble_services/ble_hts \
  $(SDK_ROOT)/components/ble/ble_services/ble_ias \
  $(SDK_ROOT)/components/ble/ble_services/ble_ias_c \
  $(SDK_ROOT)/components/ble/ble_services/ble_lbs \
  $(SDK_ROOT)/components/ble/ble_services/ble_lbs_c \
  $(SDK_ROOT)/components/ble/ble_services/ble_lls \
  $(SDK_ROOT)/components/ble/ble_services/ble_nus \
  $(SDK_ROOT)/components/ble/ble_services/ble_nus_c \
  $(SDK_ROOT)/components/ble/ble_services/ble_rscs \
  $(SDK_ROOT)/components/ble/ble_services/ble_rscs_c \
  $(SDK_ROOT)/components/ble/ble_services/ble_tps \
  $(SDK_ROOT)/components/ble/common \
  $(SDK_ROOT)/components/ble/nrf_ble_gatt \
  $(SDK_ROOT)/components/ble/nrf_ble_gq \
  $(SDK_ROOT)/components/ble/nrf_ble_qwr \
  $(SDK_ROOT)/components/ble/nrf_ble_scan \
  $(SDK_ROOT)/components/ble/peer_manager \
  $(SDK_ROOT)/components/nfc/ndef/conn_hand_parser \
  $(SDK_ROOT)/components/nfc/ndef/conn_hand_parser/ac_rec_parser \
  $(SDK_ROOT)/components/nfc/ndef/conn_hand_parser/ble_oob_advdata_parser \
  $(SDK_ROOT)/components/nfc/ndef/conn_hand_parser/le_oob_rec_parser \
  $(SDK_ROOT)/components/nfc/ndef/connection_handover/ac_rec \
  $(SDK_ROOT)/components/nfc/ndef/connection_handover/ble_oob_advdata \
  $(SDK_ROOT)/components/nfc/ndef/connection_handover/ble_pair_lib \
  $(SDK_ROOT)/components/nfc/ndef/connection_handover/ble_pair_msg \
  $(SDK_ROOT)/components/nfc/ndef/connection_handover/common \
  $(SDK_ROOT)/components/nfc/ndef/connection_handover/ep_oob_rec \
  $(SDK_ROOT)/components/nfc/ndef/connection_handover/hs_rec \
  $(SDK_ROOT)/components/nfc/ndef/connection_handover/le_oob_rec \
```

The BLE central library requires additional flags in the makefile. The following must be added the C and Assembler flags:

- DSOFTDEVICE_PRESENT
- DNRF_SD_BLE_API_VERSION=7
- DNRF_SDH_BLE_CENTRAL_LINK_COUNT=8
- DNRF_SDH_BLE_PERIPHERAL_LINK_COUNT=0

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

- DNRF_SDH_BLE_TOTAL_LINK_COUNT=8

9.2.5 Dependencies

The BLE central library requires several other EP BSP libraries.

UART Library: Used for UART/SWO debug output

10. OBDII & J1939 LIBRARY

The CAN libraries of OBDII and J1939 provides users protocols stacks for communication over CAN. The libraries initialize the CAN interface, retrieve the data, parse the data, and send the data into the cellular queue. The libraries can be configured to work with a range of J1939 SPNs and OBDII PID's. The OBDII library is compatible with vehicles using OBDII protocols on the ISO15765-4 standard. The library loops through 4 variants to attempt to receive a response from the vehicle:

- 11 bit ID, 500 Kbaud
- 29 bit ID, 500 Kbaud
- 11 bit ID, 250 Kbaud
- 29 bit ID, 250 Kbaud

The J1939 library is compatible with 250 Kbaud and 29 bit IDs.

This library is intended to be run with the Connected Vehicle example file. The example is compatible with the Connected Vehicle hardware, available through EP.

10.1 CONFIGURATION

The main header contains several defines for enabling either J1939 or OBDII. The defines are OBDII_ACTIVE and J1939_ACTIVE.

10.2 API Calls

10.2.1 can_init_obdii

Description

The function initializes the OBDII library and SPI interface used to communicate with the CAN transceiver.

Prototype

bool can_init_obdii(MCP2515 *mcp2515)

Parameters

mcp2515: structure holding configuration settings.

Return Value

True Success

False Failure

Example:

```
if (can_init_obdii( &mcp2515 ))
{
    DBGI("Initialized CAN successfully");
}
else
{
    DBGE("Failed to Initialize CAN");
}
```


Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

10.2.2 Create_OBDII_Msg

Description

The function creates a message to send into the cellular queue, taking the latest cellular diagnostic values, active DTCs, inactive DTCs, and PID values through the EP compact payload.

Prototype

void Create_OBDII_Msg(char *ep_serial, uint8_t updateVerMaj, uint8_t updateVerMin, uint16_t updateVerBui)

Parameters

ep_serial: Serial number of unit

updateVerMaj: Current major version number running on unit

updateVerMin: Current minor version number running on unit

updateVerBui: Current build version number running on unit

Return Value

N/A

Example:

```
/* Create OBDII Json message */
Create_OBDII_Msg(ep_serial, updateVerMaj, updateVerMin, updateVerBui);
```

10.2.3 Gather_General_OBDII_Info

Description

The function loops through the 4 variants of OBDII to attempt a connection with the vehicle. The function is successful when able to read the VIN of the vehicle.

Prototype

bool Gather_General_OBDII_Info(MCP2515 *mcp2515)

Parameters

mcp2515: structure holding configuration settings.

Return Value

True Success

False Failure

Example:

```
// Periodic gathering data, check for engine running (comms with ECU)
while(!Gather_General_OBDII_Info( &mcp2515 ))
```

10.2.4 Read_OBDII_Data

Description

The function reads the desired OBDII service with the data to be stored within the OBDII library.

Prototype

void Read_OBDII_Data(MCP2515 *mcp2515, uint8_t service)

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

Parameters

mcp2515: structure holding configuration settings.

service: OBDII service to read data from

```
#define OBDII_SVC_CURRENT_DATA      0x01
#define OBDII_SVC_FREEZE_DATA       0x02
#define OBDII_SVC_STORED_DTCS       0x03
#define OBDII_SVC_CLEAR_DTCS        0x04
#define OBDII_SVC_TEST_NON_CAN      0x05
#define OBDII_SVC_TEST_CAN          0x06
#define OBDII_SVC_PENDING_DTCS      0x07
#define OBDII_SVC_CNTL_OP           0x08
#define OBDII_SVC_VEHICLE_INFO      0x09
#define OBDII_SVC_PERM_DTCS         0x0A
```

Return Value

N/A

Example:

```
/* Get current obdii data with service 01
Service 2 is data captured when the last trouble code
is captured) */
Read_OBDII_Data( &mcp2515, OBDII_SVC_CURRENT_DATA );

/* Read Service 03 stored DTCs, Mode 7 and 10 are not useful for our application to get DTC */
/* Mode 10 is used by emissions testers to see if someone got the MIL off but did not fix the repair */
Read_OBDII_Data( &mcp2515, OBDII_SVC_STORED_DTCS );
```

10.2.5 Update_PID_Status

Description

Function gives user ability to support or not support the list of PIDs. The supported PIDs will be included in the cellular queue message, while the non-supported PIDs will be omitted.

Prototype

void Update_PID_Status(uint8_t pid, bool status)

Parameters

pid: Pid number to update status for, current supported PIDs are:

0 – PIDs supported [01-20]
4 – Calculated engine load
5 – Engine coolant temperature
12 – Engine speed
13 – Vehicle speed
16 – MAF air flow rate
31 – Run time since engine start
32 – PIDs supported [21-40]
47 – Fuel tank level input

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

64 – PIDs supported [41-60]
66 – Control module voltage
status: true – activate, false – de-activate

Return Value
N/A

Example:

```
/* Remove PID 4 from cloud reporting */
Update_PID_Status( 0x04, false );
```

10.2.6 can_init_j1939

The function initializes the J1939 library and SPI interface used to communicate with the CAN transceiver.

Prototype
bool can_init_j1939(MCP2515 *mcp2515)

Parameters
mcp2515: structure holding configuration settings

Return Value

True	Success
False	Failure

Example:

```
if (can_init_j1939( &mcp2515 ))
{
    DBGI("Initialized CAN successfully");
}
else
{
    DBGE("CAN controller init error");
}
```

10.2.7 Parse_J1939_Data

Description
Function to be called when data received on MCP2515 chip. Recommended to call the function following the interrupt line on the MCP2515 being brought low.

Prototype
void Parse_J1939_Data(MCP2515 *mcp2515)

Parameters
mcp2515: structure holding configuration settings.

Return Value

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

True Success
False Failure

Example:

```
/* Check to see if receive interrupt received */
while(nrf_gpio_pin_read(MCP2515_INTERRUPT) == 0)
{
    /* Parse message */
    Parse_J1939_Data( &mcp2515 );
}
/* Give delay between passess */
vTaskDelay(pdMS_TO_TICKS(3));
```

10.2.8 Create_J1939_Msg

Description

The function creates a message to send into the cellular queue, taking the latest cellular diagnostic values, DTCs, and SPN data values through the EP compact payload.

Prototype

void Create_J1939_Msg(char *ep_serial, uint8_t updateVerMaj, uint8_t updateVerMin, uint16_t updateVerBui)

Parameters

ep_serial: Serial number of unit
updateVerMaj: Current major version number running on unit
updateVerMin: Current minor version number running on unit
updateVerBui: Current build version number running on unit

Return Value

N/A

Example:

```
/* Create J1939 message */
Create_J1939_Msg(ep_serial, updateVerMaj, updateVerMin, updateVerBui);
```

10.2.9 J1939_DTC_Status_Check

Description

The function monitors DTC statuses. When a DTC active is not received for fifteen calls of this function, the DTC is marked as de-active.

Prototype

void J1939_DTC_Status_Check(void)

Parameters

N/A

Return Value

N/A

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

Example:

```
/* Check if its been 30s since receiving active dtc, then de-activate dtc */
J1939_DTC_Status_Check();
```

10.2.10 J1939_Immediate_Resp

Description

Function can be used to send an immediate response when an engine start or stop occurs.

Prototype

bool J1939_Immediate_Resp(void)

Parameters

N/A

Return Value

True	Immediately response should be sent with call to Create_J1939_Msg due to engine start or stop.
False	Engine state change not detected.

Example:

```
/* If engine RPM has fallen above or below 500RPM then immediately report */
if( J1939_Immediate_Resp() )
{
    /* Create J1939 Json message */
    Create_J1939_Msg();
```

10.2.11 Update_SPN_Status

Description

Function gives user ability to support or not support the list of SPNs. The supported SPNs will be included in the cellular queue message, while the non-supported SPNs will be omitted.

Prototype

void Update_SPN_Status(uint32_t spn, bool status)

Parameters

spn: SPN number to update status for, current supported SPNs are:

- SPN92 – Engine load
- SPN96 – Fuel level
- SPN100 – Oil pressure
- SPN110 – Coolant temperature
- SPN167 – Battery voltage
- SPN190 – Engine speed
- SPN247 – Engine hours

status: true – activate, false – de-activate

Return Value

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

N/A

Example:

```
/* Disable engine load from cloud report */
Update_SPN_Status( 92, false );
```

10.3 OBDII Task

The OBDII library requires the use of one task in the application. The task needs a size of 2000 bytes and a priority of two higher than the idle priority. The first step within the task is to enable sensor power enable, the debug UART, and OBDII library through the can_init_obdii function.

```
/* Turn on debug uart */
init_uart(TASK_4);
/* Turn on sensor power enable */
sensorPwrEnConfig(true);

if (can_init_obdii( &mcp2515 ))
{
    DBGI("Initialized CAN successfully");
}
else
{
    DBGE("Failed to Initialize CAN");
}
```

The initialization is followed by the task entering a while loop forever. The connection to the vehicle is attempted through the Gather_General_OBDII_Info function call. If the function returns true, the vehicle is successfully communicating, and data can be collected. However, if the function returns false, then low power mode is entered prior to retrying the connection.

The low power is captured below. Power to the MCP2515 and sensor power enable is removed, and the UART and TWI peripherals are disabled. After 60s, the interfaces and peripherals are reenabled, and vehicle communication is attempted again.

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

```
// Periodic gathering data, check for engine running (comms with ECU)
while(!Gather_General_OBDII_Info( &mcp2515 ))
{
    /* Set flag */
    canFail = true;
    //Shutdown power to the OBDII system
    DBGI("OBDII Failure, System is powered down, no CAN comms...\r\n\r\n");
    /* The following are functions we only want to call once during sleep */
    if( firstPass )
    {
        // Suspend LED task
        vTaskSuspend( ledTaskHandle );
        // Enable debug out for LED
        led_pause();
        firstPass = false;
    }
    /* Shutoff CAN */
    nrf_gpio_pin_clear(CAN_PWR_ENABLE);
    //Set the MCP1515 in reset to prevent the wiring from being a current leakage path
    nrf_gpio_pin_clear(CAN_RESET);
    /* Disable two wire interface */
    twi_uninit();
    /* Shutoff sensor power enable */
    sensorPwrEnConfig(false);
    /* Uninit UART */
    uninit_uart(TASK_4);
    /* Delay, put unit to sleep */
    vTaskDelay(pdMS_TO_TICKS(60000));
    /* Init debug uart */
    init_uart(TASK_4);
    /* Reenable sensor power enable */
    sensorPwrEnConfig(true);
    nrf_delay_ms(1500);
    /* Enable two wire interface */
    twi_init();
    vTaskDelay(pdMS_TO_TICKS(1000));
    //Enable power to the MCP2515
    nrf_gpio_pin_set(CAN_PWR_ENABLE);
    nrf_gpio_pin_set(CAN_RESET);
    vTaskDelay(pdMS_TO_TICKS(1000));
    //Put the MCP registers back in place after power was cut
    if (can_init_obdii(&mcp2515))
    {
        DBGI("Initialized CAN successfully");
        DBGI("System is powering up...\r\n\r\n");
    }
    else
    {
        DBGE("Failed to reinit CAN, retrying");
    }
}
```

Once communication to the vehicle is successful, the current list of support PIDs is retrieved and the stored DTCs.

```
/* Get current obdii data with service 01
Service 2 is data captured when the last trouble code
is captured) */
Read_OBDII_Data( &mcp2515, OBDII_SVC_CURRENT_DATA );

/* Read Service 03 stored DTCs, Mode 7 and 10 are not useful for our application to get DTC */
/* Mode 10 is used by emissions testers to see if someone got the MIL off but did not fix the repair */
Read_OBDII_Data( &mcp2515, OBDII_SVC_STORED_DTCS );
```

Next a cellular queue message is created with the cellular diagnostic data and vehicle data collected above along with the VIN.

```
/* Create OBDII Json message */
Create_OBDII_Msg(ep_serial, updateVerMaj, updateVerMin, updateVerBui);
```


Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

Optionally, the user may collect any http attributes on the server. In the case below, attribute “app_0” is received and overwrite the transmission interval in second.

```
/* Get HTTP Attributes here for now, Display attributes */
for( uint8_t i = 0; i < num_HTTP_ATTRIBUTES; i++ )
{
    getHTTPAttributes(i,sharedAppAttr);
    /* Check if attribute value is valid */
    if(sharedAppAttr[0] != '\0')
    {
        /* For this app, only 1 attribute used, and its for cell transmission interval, overwrite this value */
        cellTransInt = ( uint32_t ) strtoul( sharedAppAttr, NULL, 10 );
        DBGI("New Cell Transmission Interval: %ds",cellTransInt);
    }
}
```

The task then delays for transmission interval period prior to restarting the process again.

10.4 J1939 Task

The J1939 library requires the use of two tasks in the application. The first task is responsible for collecting MPC2515 J1939 data as it is received by the module. The task is allocated 2000 bytes and a priority two levels above the idle task. A major difference between the OBDII and J1939 operation is that J1939 is receiving asynchronous messages, forcing the interrupt pin on the MCP2515 chip to be monitored for data to be ready. The J1939 task is responsible for turning on debug UART, initializing the J1939 library, and SPI communication to the MCP2515.

```
init_uart(TASK_4);

if (can_init_j1939( &mcp2515 ))
{
    DBGI("Initialized CAN successfully");
}
else
{
    DBGE("CAN controller init error");
}
```

Once successfully initialized, a second J1939 task is created for transferring the data. The first J1939 task then stays in an endless while loop waiting for the interrupt pin on the MCP2515 to signify data is ready. When data is ready the Parse_J1939_Data function is called to save the data within the J1939 library. A 3ms delay is needed between receptions to prevent undesired behavior with the MCP2515.

```
while (true)
{
    /* Check to see if receive interrupt received */
    while(nrf_gpio_pin_read(MCP2515_INTERRUPT) == 0)
    {
        /* Parse message */
        Parse_J1939_Data( &mcp2515 );
    }
    /* Give delay between passess */
    vTaskDelay(pdMS_TO_TICKS(3));
}
```

The second J1939 task, used for transferring J1939 data to the cell queue is given 2000 bytes and the same priority as the original J1939 task. The task contains an infinite while loop with a 2s cadence. The J1939_DTC_Status_Check function in called on each pass to mark each DTC as inactive if it has not been received in 15 calls.

```
/* Check if its been 30s since receiving active dtc, then de-activate dtc */
J1939_DTC_Status_Check();
```

The task calls the Create_J1939_Msg function when the number of passes is greater than the cell transmission interval variable. The function is responsible for collecting the cellular diagnostic data, SPN data, and DTC data, before sending the compact payload into the cellular queue.

```
/* Create J1939 message */
Create_J1939_Msg(ep_serial, updateVerMaj, updateVerMin, updateVerBui);
```

Optionally, the user may collect any http attributes on the server. In the case below, attribute “app_0” is received and overwrite the transmission interval in second.

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

```
/* Get HTTP Attributes here for now, Display attributes */
for( uint8_t i = 0; i < num_HTTP_ATTRIBUTES; i++ )
{
    getHTTPAttributes(i,sharedAppAttr);
    /* Check if attribute value is valid */
    if(sharedAppAttr[0] != '\0')
    {
        /* For this app, only 1 attribute used, and its for cell transmission interval, overwrite this value */
        cellTransInt = ( uint32_t ) strtoul( sharedAppAttr, NULL, 10 );
        DBGI("New Cell Transmission Interval: %ds",cellTransInt);
    }
}
```

At this time, the J1939 library does not operate in low power mode. The unit shall be installed with a battery, line powered, or renewable energy source.

10.5 FreRTOS & SDK Resources

The OBDII and J1939 libraries use several NRF SDK resources.

The library uses SPI1. To enable the resource, the following must be defined to 1 in sdk_config.h:

- SPI_ENABLED
- SPI1_ENABLED
- SPI1_USE_EASY_DMA_ENABLED
- NRFX_SPI_ENABLED
- NRFX_SPI1_ENABLED

The CAN libraries use a FreeRTOS semaphore to internally protect the data when creating a message for the cell queue.

10.6 Makefile Setup

The makefile for the OBDII and J1939 libraries require the following files and folder for successful compilation. Refer to the example project provided for further clarification.

```
# Library include folders
INC_FOLDERS += \
  ../libFileHeaders/epUtilityHeaders \

# Libraries common to all targets
LIB_FILES += \
  ../libFiles/ epCVLibrary.a \
```

10.7 Dependencies

The CAN libraries of OBDII and J1939 require several other EP BSP libraries.

- BSP Library: Watchdog initialized in BSP
- Cell Library: Messages are added to the cellular queue and diagnostic data is pulled from the library
- UART Library: Not required, used for UART/SWO debug output

The OBDII/J1939 library also uses the MC33397, and the chip may not be used in the application when using the library.

11. 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. The library may be used on Galaxis and Agora52 hardware. The OBDII and J1939 libraries have internal calls to the compact payload to initialize these variable length data sets.

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

11.1 CONFIGURATION

The compact payload requires a configuration header in the config folder, compact_payload_config.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.

11.2 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. There are three compact payloads:

- Agora: Compatible with Agora52 and Connected Sensor
- Asset: Compatible with Connected Asset
- CE: Compatible with Connected Equipment

11.2.1 epcp_builder_xxxxx_init

Description

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

Prototype

void epcp_builder_agora_init(epcp_builder_agora *agora_builder)

void epcp_builder_asset_init(epcp_builder_asset *asset_builder)

void epcp_builder_ce_init(epcp_builder_ce *ce_builder)

Parameters

agora_builder: structure holding sensor data and flags for activation of each sensor on Agora52/Connected Sensor

asset_builder: structure holding sensor data and flags for activation of each sensor on Connected Asset

ce_builder: structure holding sensor data and flags for activation of each sensor on Connected Equipment

Return Value

N/A

Example:

```
//Set up compact payload
epcp_builder_agora agora_compact_payload;
epcp_builder_agora_init(&agora_compact_payload);
```

11.2.2 epcp_builder_xxxxx_deinit

Description

Function is responsible for freeing memory that has been allocated through xxxxx_get_packet.

Prototype

void epcp_builder_agora_deinit(epcp_builder_agora *agora_builder)

void epcp_builder_asset_deinit(epcp_builder_asset *asset_builder)

void epcp_builder_ce_deinit(epcp_builder_ce *ce_builder)

Parameters

agora_builder: structure holding sensor data and flags for activation of each sensor.

asset_builder: structure holding sensor data and flags for activation of each sensor.

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

ce_builder: structure holding sensor data and flags for activation of each sensor.

Return Value

N/A

Example:

```
//Free message
epcp_builder_agora_deinit(&agora_compact_payload);
```

11.2.3 xxxxx_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_builder_agora *agora_builder, EPCP_SENSOR_ID sensor_id , EPCP_SENSOR_SLOT_TYPE sensor_slot, void * sensor_data)

EPCP_ERROR_CODE asset_add_data(epcp_builder_asset *asset_builder, EPCP_SENSOR_ID sensor_id , EPCP_SENSOR_SLOT_TYPE sensor_slot, void * sensor_data)

EPCP_ERROR_CODE ce_add_data(epcp_builder_ce *ce_builder, EPCP_SENSOR_ID sensor_id , EPCP_SENSOR_SLOT_TYPE sensor_slot, void * sensor_data)

Parameters

xxxxx_builder: structure holding sensor data and flags for activation of each sensor.

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 is entering sensor data of htu21d/si7021 temperature and humidity sensor data into the compact payload. The data is manipulated to the proper format prior to placing in agora_add_data.

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

```c
#if HTU21D_ACTIVE
int16_t temperature = (int16_t)(si7021_temp * 100);
// Send as big endian
int8_t tempArray[2];
tempArray[0] = ((temperature >> 8) & 0xFF);
tempArray[1] = temperature & 0xFF;
agora_add_data(&agora_compact_payload, EPCP_SI7021, EPCP_TEMP, &tempArray);
uint16_t humidity = (uint16_t)(si7021_hum * 100);
// Send as big endian
int8_t humidArray[2];
humidArray[0] = ((humidity >> 8) & 0xFF);
humidArray[1] = humidity & 0xFF;
agora_add_data(&agora_compact_payload, EPCP_SI7021, EPCP_HUM, &humidArray);
#endif
```

11.2.4 xxxxx_get_packet

Description

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

Prototype

uint16_t agora_get_packet(epcp_builder_agora *agora_builder, char* ret_packet)

uint16_t asset_get_packet(epcp_builder_asset *asset_builder, char* ret_packet)

uint16_t ce_get_packet(epcp_builder_cess *ce_builder, char* ret_packet)

Parameters

xxxxx_builder: structure holding sensor data and flags for activation of each sensor.

ret_packet: packet to be returned

Return Value

uint16_t: size of compact payload message.

Example:

```c
//Generate completed packet
uint16_t packet_size = 0;
uint8_t *buff = agora_get_packet(&agora_compact_payload, &packet_size);
```

11.3 FreeRTOS & SDK Resources

The compact payload uses FreeRTOS heap to store the message. The user shall call the epcp_builder_xxxxx_deinit message to free the data.

11.4 Makefile Setup

The makefile for the compact payload library requires the following files and folder for successful compilation. Refer to the example project provided for further clarification.

```
# Library include folders
INC_FOLDERS += \
  ../libFileHeaders/epUtilityHeaders \

# Libraries common to all targets
LIB_FILES += \
  ../libFiles/epXXLibrary.a \
```


Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

11.5 Dependencies

The compact payload does not have any dependencies.

12. EP BSP

The EP BSP handles configuration of the Agora52 board. The library initializes the watchdog, board bring-up, and battery voltage readings. The FreeRTOS idle task is responsible for feeding the watchdog.

12.1 CONFIGURATION

The configuration settings available for the BSP are below. The configuration settings are inputs to the later to be discussed ep_bsp_init function. It is recommended to keep these defines in main.h.

Configuration Setting	Define	Values
Watchdog reload value	WATCHDOG_RELOAD	uint32_t value in seconds
Watchdog reload rate	WATCHDOG_RELOAD_RATE	uint32_t value in seconds

12.2 API Calls

12.2.1 ep_bsp_init ()

Description

The init function initializes the NRF52 Power system along with the watchdog. 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. A couple other pins are left in their current settings to prevent the code from going into an undefined state. These include the external crystal and debug UART pins.

Prototype

void ep_bsp_init(uint32_t reloadValue, uint32_t reloadRate)

Parameters

reloadValue: Watchdog reload value, example: 60 (represents 60s)

reloadRate: Watchdog reload rate, example: 30 (represents reloading the watchdog every 30s)

Return Value

N/A

12.2.2 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, unitializes the SAADC, and disables the battery monitor GPIO pin.

Prototype

float ep_bsp_read_battery_voltage(void)

Parameters

N/A

Return Value

Float: represents the battery voltage.

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

12.3 FreeRTOS & SDK Resources

The BSP requires the watchdog and ADC to be enabled in the SDK. For the watchdog and ADC to be enabled, the following must be defined to 1 in sdk_config.h:

- NRFX_WDT_ENABLED
- WDT_ENABLED
- NRFX_SAADC_ENABLED

The EP BSP does not require any FreeRTOS resources.

12.4 Makefile Setup

The makefile for the EP BSP library requires the following files and folder for successful compilation. Refer to the example project provided for further clarification.

```
# Library include folders
INC_FOLDERS += \
  ../libFileHeaders/epBSPHeaders \

# Libraries common to all targets
LIB_FILES += \
  ../libFiles/epXXLibrary.a \
```

12.5 Dependencies

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

13. OTHER COMM LIBRARIES

The other communication interface libraries described in Section 2 are not released.

14. SENSOR LIBRARY

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

- ADS124S08
- BME680
- HTU21D/SI7021
- ICM20602
- MC3479
- MC33397
- MCP2515
- MCP23008
- SC16IS750
- VL5310X

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

15. BOOTLOADER LIBRARY

The bootloader is provided to users as a hex file. The bootloader takes a valid image in external flash that has yet to be programmed and programs the image to application memory on the microcontroller. A separate bootloader is provided for Galaxis and Agora52, however the same settings hex is used for both.

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

16. 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.

16.1 Connected Assest

The product brief and user guide for Connected Asset can be found here: Connected Asset Tracking with IoT Integration | Embedded Planet

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

The project has a minimum sleep current of 3uA with TLS disabled and 23uA with TLS enabled.

16.2 Connected Vehicle

The product brief and user guide for Connected Vehicle can be found here: Connected Vehicle IoT Connectivity Module | Embedded Planet

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

16.3 Connected Equipment

The product brief and user guide for Connected Equipment can be found here: Connected Industrial Equipment IoT Module | Embedded Planet

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

16.4 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.

16.5 Additional Example

Contact EP for additional examples or questions.

17. INTEGRATION

17.1 GENERAL INFORMATION

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

- **Compiler:** GCC V11.3 release 1
- **IDE:** Visual Studio Code V1.809.1 or greater
- **Debugger:** Segger J-Link

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, a makefile, and an example application.

17.2 LOW POWER OPERATION

In the provided example application, there are several lines that enable the lowest power operation of the Agora and Galaxis modules. The Galaxis has been able to reduce sleep current to 3uA. Within the application, all peripherals that are enabled must be disabled when not in use. Users must include the sensorPwrEnConfig function throughout the tasks. When a task needs sensor power enable, the function must be called with the true parameter. When a function does not

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

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 task that has called the function has called a corresponding false.

17.3 NRF SDK

The BSP uses NRF SDK version 17.1. A FreeRTOS build is provided within the SDK and is utilized by the libraries. There have been minor EP modifications to the SDK to support the BSP.

The FreeRTOS build within the SDK is configured to use RTC1. The following must be configured in the sdk configuration file:

- NRFX_RTC_ENABLED
- RTC_ENABLED

17.3.1 Low Power Modification

The SDK requires a FreeRTOSConfig header file in the app for configuring the OS. There is a slight change made to the file to allow the microcontroller to reach its lowest power potential. The following additions were made to ensure the UARTE using the DMA kept the micro awake until they were disabled:

```
/* Tickless idle/low power functionality. */
#define configUSE_LOW_POWER
/* Need to prevent sleep for uart interrupts otherwise they will be missed and system will crash */
#define configCHECK_PERIPHERALS()                                              \
    if( NRF_UARTE0->ENABLE != 0 || NRF_UARTE1->ENABLE != 0 )                   \
    {                                                                          \
        xExpectedIdleTime = 0;                                                 \
    }                                                                          \
```

UART1 is used for the debug UART and UARTE0 is used for the cellular library. Without this change, the microcontroller would go into SYSTEM ON sleep, the DMA would fire, and the unit would crash.

17.4 MAKEFILE

A makefile is provided in the example project for a guide. The makefile provides a list of NRF SDK source code files and includes folders that are needed. The libraries discussed above are added as library files. The example makefile provides a list of the recommended C, Assembler, and C++ flags. The recommended flags are below:

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

```
# C flags common to all targets
CFLAGS += $(OPT)
CFLAGS += -DAPP_TIMER_V2
CFLAGS += -DAPP_TIMER_V2_RTC1_ENABLED
CFLAGS += -DCONFIG_GPIO_AS_PINRESET
CFLAGS += -DFLOAT_ABI_HARD
CFLAGS += -DFREERTOS
CFLAGS += -DBOARD_AGORA
CFLAGS += -DNDEBUG
CFLAGS += -DNRF52840_XXAA
CFLAGS += -DNRF_SD_BLE_API_VERSION=7
CFLAGS += -DS140
CFLAGS += -DSOFTDEVICE_PRESENT
CFLAGS += -mcpu=cortex-m4
CFLAGS += -mthumb -mabi=aapcs
CFLAGS += -mfloat-abi=hard -mfpu=fpv4-sp-d16
# keep every function in a separate section, this allows linker to discard unused ones
CFLAGS += -ffunction-sections -fdata-sections -fno-strict-aliasing
CFLAGS += -fno-builtin -fshort-enums
# C++ flags common to all targets
CXXFLAGS += $(OPT)
# Assembler flags common to all targets
ASMFLAGS += -DAPP_TIMER_V2
ASMFLAGS += -DAPP_TIMER_V2_RTC1_ENABLED
ASMFLAGS += -DCONFIG_GPIO_AS_PINRESET
ASMFLAGS += -DFLOAT_ABI_HARD
ASMFLAGS += -DFREERTOS
ASMFLAGS += -DBOARD_AGORA
ASMFLAGS += -DNDEBUG
ASMFLAGS += -DNRF52840_XXAA
ASMFLAGS += -DNRF_SD_BLE_API_VERSION=7
ASMFLAGS += -DS140
ASMFLAGS += -DSOFTDEVICE_PRESENT
ASMFLAGS += -mcpu=cortex-m4
ASMFLAGS += -mthumb -mabi=aapcs
ASMFLAGS += -mfloat-abi=hard -mfpu=fpv4-sp-d16
ASMFLAGS += -g3
```

17.5 FREERTOS HOOKS

Several FreeRTOS hooks are utilized and recommended within the FreeRTOSConfig header. The following are defined to 1. A tick hook is not used in the example, so its define remains 0.

- configUSE_IDLE_HOOK
- configCHECK_FOR_STACK_OVERFLOW (set to 0 for low power operation)
- configUSE_MALLOC_FAILED_HOOK (set to 0 for low power operation)

The function definitions for these hooks are located in the freertos_hooks source code. An example use case is below. A stack and malloc error results in an error message on the debug output. The idle hook is used for reloading the watchdog at the desired rate. The watchdog is reloaded in the idle hook, as this hook is the lowest priority function running in FreeRTOS.

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

```c
#if configUSE_IDLE_HOOK
void vApplicationIdleHook( void ){
    static TickType_t ticks_last = 0;
    TickType_t ticks_now = xTaskGetTickCount();

    //Feed the watchdog every WATCHDOG_FEED_RATE % of WATCHDOG_RELOAD in seconds.
    if(((ticks_now - ticks_last) / 1024) >= WATCHDOG_RELOAD * (WATCHDOG_RELOAD_RATE / 100) ){
        nrf_drv_wdt_channel_feed(m_channel_id);
        ticks_last = ticks_now;
    }
}
#endif

#if configUSE_TICK_HOOK
void vApplicationTickHook( void ){

}
#endif

#if configCHECK_FOR_STACK_OVERFLOW
void vApplicationStackOverflowHook( TaskHandle_t xTask, signed char *pcTaskName ){
    DBGE("stack overflow!");
}
#endif

#if configUSE_MALLOC_FAILED_HOOK
void vApplicationMallocFailedHook( void ){
    DBGE("malloc failed!");
}
#endif
```

17.6 NRF SOFTDEVICE

The example application includes the S140 soft device from Nordic. The linker offsets the application memory to provide the needed space and the makefile includes the S140 and SOFTDEVICE_PRESENT flags. The softdevice can be used for BLE operation. The S140 hex is located within the SDK repository. The softdevice uses the following resources:

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

Instance	Access SoftDevice enabled	Access SoftDevice disabled
CLOCK	Restricted	Open
POWER	Restricted	Open
RADIO	Blocked[1]	Open
TIMER0	Blocked[1]	Open
RTC0	Blocked	Open
TEMP	Restricted	Open
RNG	Restricted	Open
ECB	Restricted	Open
CCM	Blocked[2]	Open
AAR	Blocked[2]	Open
EGU1/SWI1/Radio Notification	Restricted[3]	Open
EGU5/SWI5	Blocked	Open
ACL	Restricted	Open
NVMC	Restricted	Open
MWU	Restricted[4]	Open
FICR	Blocked	Blocked
UICR	Restricted	Open
NVIC	Restricted[5]	Open

17.7 EXAMPLE FOLDER STRUCTURE

The example provides a recommended folder structure:

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

```
AGORA
  _build
  bootloader
    bootloader_v0.0.2_AGORA.hex
    settings_v0.0.1.hex
  ble_app_multilink_central_gcc_nrf52.ld
  Makefile
config
  cell_helper.h
  FreeRTOSConfig.h
  sdk_config.h
Documentation
GALAXIS
  _build
  bootloader
    bootloader_v0.0.2_GALAXIS.hex
    settings_v0.0.1.hex
  ble_app_multilink_central_gcc_nrf52.ld
  Makefile
INTERNAL_USE_ONLY
libFileHeaders
  epBSPHeaders
    ep_bsp.h
  epCellHeaders
    cell_helper.h
  epDriverHeaders
    ble_service_bme680
    bme68x_defs.h
    bme68x.h
    bme680.h
    bsec_datatypes.h
    bsec_interface.h
    bsec_serialized_configurations_iaq.h
    htu21d.h
    icm20602.h
    mc3479.h
    mc33397.h
    mcp2515.h
    mcp23008.h
    sc16is750.h
  epUtilityHeaders
    led_helper.h
    qspi_helper.h
    time_helper.h
    uart_helper.h
libFiles
  epBSPLibrary.a
  epCellLibrary.a
  epDebugUartLibrary.a
  epLedLibrary.a
  epQspiLibrary.a
  epSensorLibrary.a
  epTimeLibrary.a
source
  freertos_hooks.c
  main.c
  main.h
```

The example project contains a makefile and linker script. It is recommended to use this makefile as a starting point for faster integration. The linker script leaves space for a bootloader, SoftDevice, and BLE characteristics, as defined in the NRF SDK. The source folder contains a source and header file. These files are the example application and should be used as a starting point for further development. The libFiles folder is the location for placing purchased EP library files. The config folder contains configuration information for the project. The sdk_config header can be freely modified as required for SDK operation. The FreeRTOSConfig header can be freely modified for any desired FreeRTOS settings. The libFileHeaders folder cannot be modified and are shared between the application and libraries.

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

18. PERFORMANCE & RESOURCE USAGE

The nrf52840 contains 1024kB of flash memory and 256kB of RAM. FreeRTOS uses dynamic memory allocation that can cause RAM to overflow if overlooked. The settings can be modified in FreeRTOSConfig header file and should be validated in the generated .map file. The following function is called within all example projects to track the FreeRTOS Heap usage:

```
// Monitor FreeRTOS stack usage
DBGI("FreeRTOS Heap Remaining: %d Bytes", xPortGetFreeHeapSize());
```

The memory map and organization for the nrf52840 is below.

The recommended memory usage is:

	Usage	Memory Range
	Bootloader settings	0x000FF000-0x00100000 (4kB)
	MBR parameter storage	0x000FE000-0x000FF000 (4kB)
FLASH	Bootloader	0x000F8000-0x000FE000 (24kB)
	Application area	0x00027000-0x000F8000 (840kB)
	SoftDevice	0x00001000-0x00026000 (148kB)
	Master Boot Record (MBR)	0x00000000-0x00001000 (4kB)

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

RAM	Application area	0x200043b8-0x20040000 (244kB)

19. STATIC ANALYSIS RESULTS

Contact EP to discuss.

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.

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 Agora and Galaxis documentation can be found at:
https://www.embeddedplanet.com/agora
https://www.embeddedplanet.com/galaxis

APPENDIX B. LICENSES

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

Name	Version	Manufacturer	License	Project Location
Nordic SDK	17.1.0	Nordic	Nordic Semiconductor ASA	Nrf_sdk_17_1/license.txt
Nordic SoftDevice S140	7.2.0	Nordic	Nordic Semiconductor ASA	Nrf_sdk_17_1/nrf_softdev/s140nrf52720/s140_nrf52_7.2.0_license-agreement
FreeRTOS Kernel	10.0.0	FreeRTOS	MIT	Nrf_sdk_17_1/external/freertos/license/license.txt
backoffAlgorithm	1.3.0	FreeRTOS	MIT	Contained within Cellular Library
CoAP	2.0.0	1NCE GmbH	MIT	Contained within Cellular Library
coreHTTP	3.0.0	FreeRTOS	MIT	Contained within Cellular Library
coreJSON	3.2.0	FreeRTOS	MIT	Contained within Cellular Library
coreMQTT	2.1.0	FreeRTOS	MIT	Contained within Cellular Library
coreMQTT-Agent	1.2.0	FreeRTOS	MIT	Contained within Cellular Library
FreeRTOS-Cellular-Interface	1.0.0	FreeRTOS	MIT	Contained within Cellular Library
FreeRTOS-LoRaWAN	N/A	FreeRTOS	MIT	Contained within LoRa Library
http-parser	N/A	NodeJS	Apache 2.0	Contained within Cellular Library, heavily modified
mbedTLS	2.28.0	Open Source	Apache 2.0	Contained within Cellular Library, minor modifications
Node-20.5.1	20.5.1	Node.JS	Node.js	Contained within Cellular Library
Ota-for-aws	3.4.0	FreeRTOS	MIT	Contained within Cellular Library
tinyCBOR	0.5.4	FreeRTOS	MIT	Contained within Cellular Library

Embedded Planet Confidential – Do not copy or reproduce without expressed written consent of Embedded Planet, Inc.