Index

| Introduction | 2 |
|------------------------------|----|
| Boards / Kits | 2 |
| IDE / SDK | 2 |
| Useful links | 2 |
| Project import procedure | 3 |
| Code structure | 6 |
| Peripherals resources | 7 |
| Pin Connections | 7 |
| SPI | 8 |
| Timer | 10 |
| LoRaWAN stack overview | 11 |
| Basic use | 11 |
| Network events and callbacks | 13 |
| Application callbacks | 13 |
| Link check response handler | 14 |
| Battery level handler | 14 |
| Error codes | 15 |
| Modifying LoRaWAN stack | 16 |
| LoRaWAN Version | 18 |
| LoRaWAN Phy Region | 19 |
| LoRaWAN Activation Mode | 21 |
| LoRaWAN ID / Keys | 22 |
| LoRaWAN Class Mode | 23 |
| LoRaWAN Channels | 24 |
| Modifying LoRaWAN Radio | 26 |
| Running Demo Applications | 28 |
| LoRaWAN + Pearl Gecko | 28 |
| LoRaWAN + Blue Gecko | 28 |

Introduction

This document describes the use of the open source LoRaWAN end node stack from ARMmbed for the Silicon Labs EFM32 Pearl Gecko and the EFR32 Blue Gecko MCUs connected to a SPI based Semtech LoRa Transceivers.

Boards / Kits

These were the items used for the tests:

- Starter Kit SLSTK3401A EFM32PG1B200F256GM48 MCU
- Starter Kit SLWSTK6020B SLWRB4104A EFR32BG13P632F512GM48 2.4 GHz radio board
- Semtech SX1262MB2xAS Mbed Shield, SX1262, 915 MHz for North America
- Semtech SX1272MB2xAS Mbed Shield, SX1272, 915 MHz for North America
- Vermont Adapter Board Adapter to connect the shields on the IO expansion header

IDE / SDK

The tests were performed with the following tools. Newer versions should also work.

- Simplicity Studio Version: SV4.1.11.2
- Gecko SDK Suite v2.6.1: Bluetooth 2.12.1.0, Flex 2.6.1.0, MCU 5.8.1.0
- GNU ARM v7.2.1

Useful links

The source codes of the examples are located here:

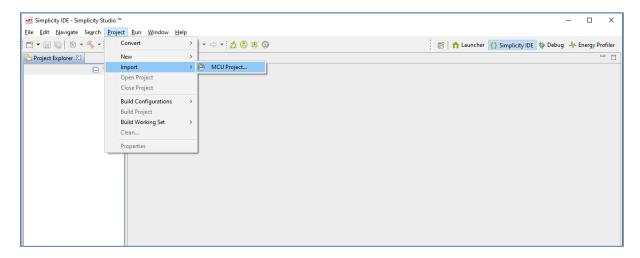
- https://github.com/udev-br/Silabs-Pearl-LoRaWan-Mbed
- https://github.com/udev-br/Silabs-Blue-LoRaWan-Mbed

And make sure to take a look at the following material to better understand how the stack and its configuration work:

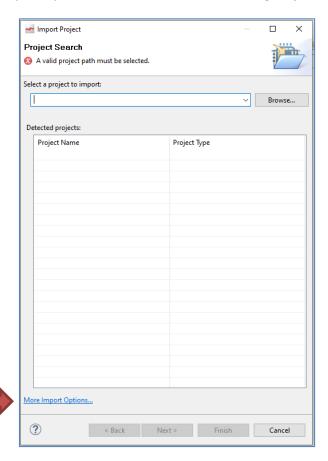
- https://os.mbed.com/docs/mbed-os/v5.13/reference/lora-tech.html
- https://os.mbed.com/docs/mbed-os/v5.13/reference/lorawan-configuration.html
- https://os.mbed.com/docs/mbed-os/v5.13/tutorials/LoRa-tutorial.html
- https://os.mbed.com/docs/mbed-os/v5.13/apis/lorawan.html
- https://os.mbed.com/docs/mbed-os/v5.13/apis/lorawan-api.html
- https://os.mbed.com/blog/entry/Introducing-LoRaWAN-11-support/

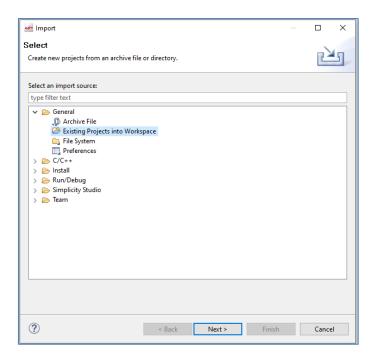
Project import procedure

- 1. Open Simplicity Studio
- 2. Change to Simplicity IDE Perspective
- 3. Click on Project / Import / MCU Project

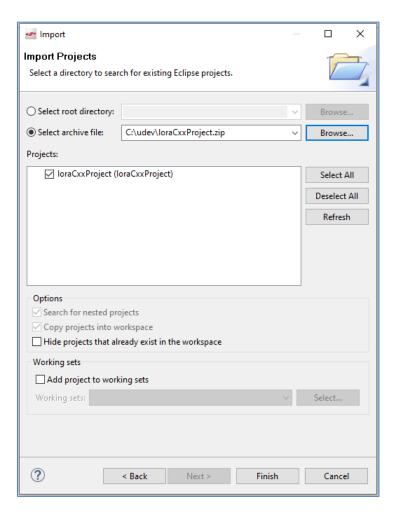


4. Click on More Import Options and choose General / Existing Projects into Workspace

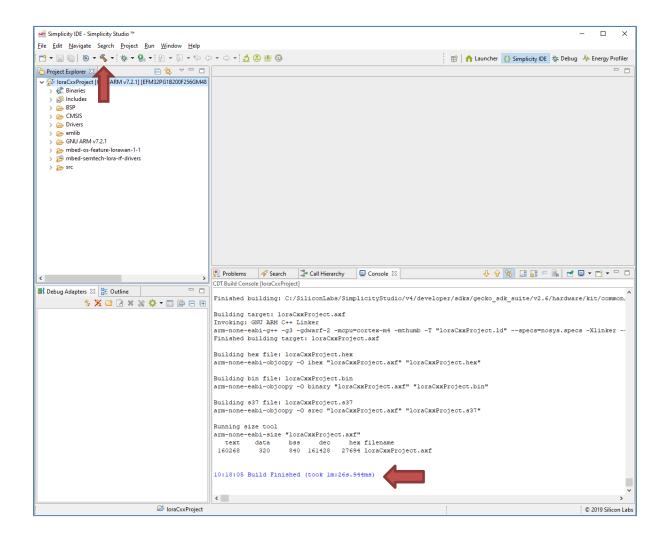




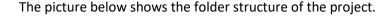
5. Browse to the zip file of the project and then click on Finish

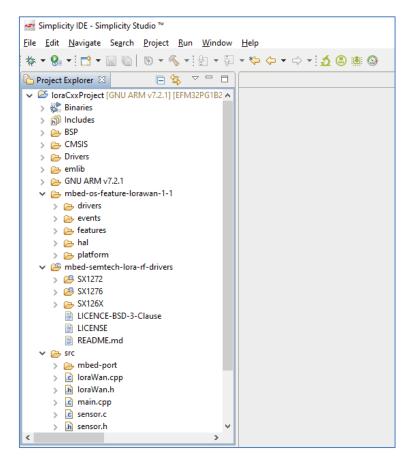


6. Click on the build project icon to check if everything is ok.



Code structure





The Semtech RF drivers are located on "mbed-semtech-lora-rf-drivers" folder. The drivers are based on this repository: https://github.com/ARMmbed/mbed-semtech-lora-rf-drivers/tree/master

The LoraWan stack is located at "mbed-os-feature-lorawan-1-1" folder. The stack is based on this repository: https://github.com/ARMmbed/mbed-os/tree/feature-lorawan-1-1

About the LoRaWAN stack, all unnecessary files from mbed-os framework were removed, keeping only the dependencies that are used on the stack itself, such as the event control feature.

Stub and target functions are located in "src/mbed-port" folder. It is user responsibility to implement those functions. The examples already cover the peripherals port for MCUs EFR32BG13P and EFR32BG13P.

The basic demo application of the LoRaWAN stack resides on the "src" folder.

And finally, there are the support folders from Silabs, including files for BSP, Drivers and emlib.

Peripherals resources

The ARM MBED LoRaWAN stack uses few resources from the MCU:

- (n) GPIO / INTERRUTPS
- SPI / DMA
- TIMER

Following are the instructions on how to configure them.

Pin Connections

Open the file "src/mbed-port/target/PinNames.h" to change the radio connection.

NC stands for Not Connected.

```
In PinNames.h ≅
   ⊖typedef enum {
            EFM32_STANDARD_PIN_DEFINITIONS,
            /* Starter Kit says LEDO and LED1,
           LED0 = PF4,
LED1 = PF5,
LED2 = LED0,
            LED4 = LED1,
            /* Push Buttons */
           SW0 = PF6,

SW1 = PF7,
            BTN0 = SW0,
BTN1 = SW1,
            // Standardized button names
BUTTON1 = BTNO,
BUTTON2 = BTN1,
            SX127X_SPI_MOSI
                                                   = PC7,
= PC8,
            SX127X_SPI_MISO
            SX127X_SPI_CLK
SX127X_SPI_CS
SX127X_RESET
                                                    = PC9,
                                                   = PD11,
           SX127X_RESET

SX127X_SPI_DIOO

SX127X_SPI_DIO1

SX127X_SPI_DIO2

SX127X_SPI_DIO3

SX127X_SPI_DIO4

SX127X_SPI_DIO5
                                                  = PA4,
= PA3,
= PA2,
                                                   = PAO,
                                                   = NC,
= NC,
            SX126X_SPI_MOSI
           SX126X_SPI_MISO
SX126X_SPI_CLK
SX126X_SPI_CS
SX126X_RESET
                                                   = PC7,
= PC8,
                                                   = PD10,
            SX126X_DIO1
SX126X_BUSY
SX126X_FREQ_SELECT
                                                   = PB13,
                                                   = PA4,
                                                   = NC,
            SX126X DEVICE SELECT
            SX126X_CRYSTAL_SELECT = NC,
SX126X_ANT_SWITCH = PD11,
```

To select the SPI peripheral, open the file "src/mbed-port/target/PinNames.h" and modify the return of "spi get peripheral name" function. In this example we are using the SPI 1.

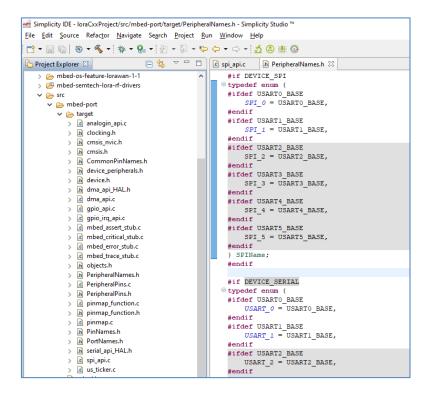
```
Simplicity IDE - IoraCxxProject/src/mbed-port/target/spi_api.c - Simplicity Studio ™
<u>File Edit Source Refactor Navigate Search Project Run Window Help</u>
 E spi_api.c 
☐ PeripheralNames.h
 Project Explorer 🛭
       GSPIName spi_get_peripheral_name(PinName mosi, PinName miso, PinName mclk) {
                                                                                                               return SPI 1;
       🗸 🗁 src

→ b target

→ c target

→
                      > c analogin_api.c
                                                                                                           ⊖void spi_init(spi_t *obj, PinName mosi, PinName miso, PinName clk, PinName cs)
                       > h clocking.h
                       > h cmsis_nvic.h
                                                                                                                       CMU ClockEnable (cmuClock HFPER, true);
                       > h cmsis.h
                                                                                                                        spi_preinit(obj, mosi, miso, clk, cs);
                       > h CommonPinNames.h
                                                                                                                       CMU_ClockEnable(spi_get_clock_tree(obj), true);
usart_init(obj, 100000, usartDatabits8, true, usartClockMode0);
                       > h device_peripherals.h
                       > h device.h
                       > h dma_api_HAL.h
                                                                                                                        spi_enable_pins(obj, true, mosi, miso, clk, cs);
                       > 🔝 dma_api.c
                                                                                                                        spi enable(obj, true);
                       > @ gpio_api.c
                       > i gpio_irq_api.c
> i mbed_assert_stub.c
                                                                                                          @void spi enable_event(spi_t *obj, uint32_t event, uint8_t enable)
                        > 🖻 mbed_critical_stub.c
                                                                                                                         if(enable) obj->spi.event |= event;
                       > lc mbed error stub.c
                        > @ mbed_trace_stub.c
                                                                                                                        else obj->spi.event &= ~event;
                       > h objects.h
                       > h PeripheralNames.h
                       > R PeripheralPins.c
                                                                                                                  void spi_enable_interrupt(spi_t *obj, uint32_t handler, uint8_t enable)
                       > h PeripheralPins.h
                       > le pinmap function.c
                                                                                                               ^{\star} This will enable the interrupt in NVIC for the associated USART RX channel
                       > h pinmap_function.h
                       > i pinmap.c
> i PinNames.h
                                                                                                                      * obj: pointer to spi object
                                                                                                                             handler: pointer to interrupt handler for this channel
                       > h PortNames.h
                                                                                                                        * enable: Whether to enable (true) or disable (false) the interrupt
                       > In serial_api_HAL.h
                        > 🖻 spi_api.c
                        > c us_ticker.c
                                                                                                           ⊕void spi_enable_interrupt(spi_t *obj, uint32_t handler, uint8_t enable)
                       h mbed.h
             > c loraWan.cpp
                                                                                                                        IRQn Type IRQvector;
                                                                                                                        switch ((uint32 t)obj->spi.spi) {
```

These are valid values for the provided examples:

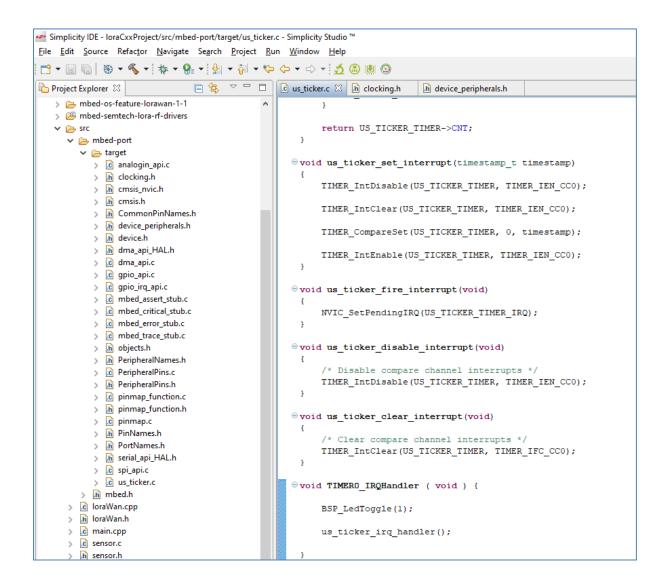


Make sure to check the valid pin function on the file "src/mbed-port/target/PeripheralPins.c". For example the MOSI pin:

```
🖟 PeripheralPins.c 🖂
     const PinMap PinMap_SPI_MOSI[] = {
     #ifdef USARTO_BASE
/* USARTO */
           {PAO, SPI_0, 0},
{PA1, SPI_0, 1},
{PA2, SPI_0, 2},
            {PA3, SPI_0, 3},
            {PA4, SPI_0,
           {PA5, SPI_0,
{PB11, SPI_0,
{PB12, SPI_0,
{PB13, SPI_0,
                                     5},
                                     6},
                                     7},
            { PB14, SPI_0,
           {PB15, SPI_0, 10},
{PD9, SPI_0, 17},
{PD10, SPI_0, 18},
{PD11, SPI_0, 19},
            {PD12, SPI_0, 20},
{PD13, SPI_0, 21},
{PD14, SPI_0, 22},
            {PD15, SPI_0, 23},
     #endif
     #ifdef USART1 BASE
            /* USART1 */
            {PC6, SPI_1, 11},
           {PC7, SPI_1, 12},
{PC8, SPI_1, 13},
{PC9, SPI_1, 14},
{PC10, SPI_1, 15},
            {PC11, SPI_1, 16},
            {PF0, SPI_1, 24},
            {PF1, SPI_1, 25},
            {PF2, SPI_1, 26},
{PF3, SPI_1, 27},
{PF4, SPI_1, 28},
            {PF5, SPI_1, 29}, 
{PF6, SPI_1, 30},
            {PF7, SPI_1, 31},
      #endif
            {NC , NC
```

Timer

The "system tick" required by the event control feature (queue module) is configured on the file "src/mbed-port/target/us_ticker.c" which currently configures the TIMERO peripheral to interrupt every 10ms. To change the timer and oscillator used, user must modify files "src/mbed-port/target/clocking.h" and "src/mbed-port/target/device_peripherals.h"



LoRaWAN stack overview

The steps below can be found here: https://os.mbed.com/docs/mbed-os/v5.13/apis/lorawan.html

Due to the fact that most of the LoRaWAN devices are simple telemetry devices, the stack and its operation need to be as simple as possible. That's why the Mbed LoRaWAN stack is event driven.

Basic use

To bring up the Mbed LoRaWAN stack, consider the following progression.

- 1. Construct an event queue
- 2. Construct a LoRadio object
- 3. Instantiate LoRaWANInterface, and pass LoRaRadio object
- 4. Initialize mac layer and pass EventQueue object

```
Simplicity IDE - IoraCxxProject/src/main.cpp - Simplicity Studio ™
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer 🛭 🕒 🔄 🔻 🔻 🗖 🖟 main.cpp 🖾 💪 IoraWan.cpp

▼ ForaCxxProject [GNU ARM v7.2.1] [EFM32PG1B200F2

    int main(void) {
   > 🚜 Binaries
   > 🔊 Includes
   > 🗁 BSP
                                                 /* Chip errata */
   > 🗁 CMSIS
                                                  CHIP_Init();
   > 🗁 Drivers
                                                  BoardSetup();
   > 📂 emlib
   > 👝 GNU ARM v7.2.1
                                                   sensorInit():
   > 🚌 mbed-os-feature-lorawan-1-1
   > 🔑 mbed-semtech-lora-rf-drivers
                                                  SX1272_LoRaRadio radio ( SX127X_SPI_MOSI,

✓ 

Æ src

                                                                             SX127X SPI MISO,
     > 🚌 mbed-port
                                                                            SX127X SPI_CLK,
SX127X SPI_CS,
     > 🖟 loraWan.cpp
     > h loraWan.h
                                                                             SX127X_RESET,
     > c main.cpp
                                                                             SX127X_SPI_DIOO,
     > c sensor.c
                                                                             SX127X_SPI_DIO1,
SX127X_SPI_DIO2,
     > h sensor.h
                                                                             SX127X_SPI_DIO3,
                                                                             SX127X_SPI_DIO4,
                                                                             SX127X_SPI_DIO5 );
                                                   events::EventQueue ev_queue;
                                                   LoRaWANInterface lorawanInterface( radio );
                                                   lorawanInterface.initialize(&ev_queue);
```

- 5. Set up the event callback
- 6. Add network credentials (security keys) and any configurations
- 7. Connect.

```
Simplicity IDE - IoraCxxProject/src/IoraWan.cpp - Simplicity Studio ™
File Edit Source Refactor Navigate Search Project Run Window Help
😭 🔒 h Launcher 🚯 Simplicity IDE 💠 Debug 🦫 Ener
                                Project Explorer 🛭

▼ SoloraCxxProject [GNU ARM v7.2.1] [EFM32PG1B200F2
                                                             #include "Event.h"
#include "LoRaWANInterface.h"
#include "LoRaRadio.h"
      🔊 Includes
    > 🗁 BSP
> 🗁 CMSIS
      Drivers
                                                             uint8_t NwkSKey[16] = MBED_CONF_LORA_NWKSKEY;
uint8_t AppSKey[16] = MBED_CONF_LORA_APPSKEY;
uint8_t NwkKey[16] = MBED_CONF_LORA_NETWORK_KEY;
uint8_t AppKey[16] = MBED_CONF_LORA_APPLICATION_KEY;
      ⇒ emlib
⇒ GNU ARM ∨7.2.1
       mbed-os-feature-lorawan-1-1
mbed-semtech-lora-rf-drivers

✓ 

æ
src

         mbed-port
loraWan.cpp
                                                            ⊖loraWan::loraWan( events::EventQueue & queue, LoRaRadio & radio, LoRaWANInterface & lorawanInterface ):
                                                                         m_eventQueue( queue ),
m_radio( radio ),
m_lorawanInterface ( lorawanInterface )
        > h loraWan.h
       > c sensor.c
       > lh sensor.h
                                                                   m_lorawanInterface.add_app_callbacks( &m_lorawanAppCallbacks );
                                                                    m_connection.connect_type = LORAWAN_CONNECTION_ABP;
                                                                   m_connection.connect_cype = LORAWAN_CONNECTION_ABP;
m_connection.connection_u.abp.nwk_id= 1;
m_connection.connection_u.abp.adv_addr = MBED_CONF_LORA_DEVICE_ADDRESS;
m_connection.connection_u.abp.app_skey = AppSKey;
m_connection.connection_u.abp.nwk_skey = NwkSKey;
m_connection.connection_u.abp.nwk_senckey = NwkSKey;
m_connection.connection_u.abp.snwk_sintkey = NwkSKey;
                                                                   m_lorawanInterface.disable_adaptive_datarate();
                                                                   m_lorawanInterface.set_datarate( DR_2 );
                                                                   m_lorawanInterface.set_device_class( CLASS_A );
                                                                    m lorawanInterface.connect( m connection );
```

8. To send a message call lorawanInterface send function.

```
Simplicity IDE - IoraCxxProject/src/IoraWan.cpp - Simplicity Studio ™
<u>F</u>ile <u>E</u>dit <u>Source Refactor <u>N</u>avigate Se<u>arch Project Run <u>W</u>indow <u>H</u>elp</u></u>
🖹 🥞 ▽ 🗆 🗖 🖟 *main.cpp 🖟 loraWan.cpp 🛭
Project Explorer 🛭

▼ SoraCxxProject [GNU ARM v7.2.1] [EFM32PG1B200F.
   > 🐉 Binaries
                                          ⊖intl6 t
    ncludes
                                            loraWan::send ( uint8_t port, const uint8_t *data, uint16_t length, int flags )
    BSP
    CMSIS
   > 🗁 Drivers
                                                return m lorawanInterface.send( port, data, length, flags );
   > 降 emlib
   > 👝 GNU ARM v7.2.1
```

Network events and callbacks

Here is the list of possible events that you can post from the stack to the application:

| Event | Description |
|---------------------|--|
| CONNECTED | When the connection is complete |
| DISCONNECTED | When the protocol is shut down in response to disconnect() |
| TX_DONE | When a packet is transmitted |
| TX_TIMEOUT | When the stack is unable to send packet in TX window |
| TX_ERROR | A general TX error |
| TX_CRYPTO_ERROR | If MIC fails, or any other crypto related error |
| TX_SCHEDULING_ERROR | When the stack is unable to schedule a packet |
| TX_TIMEOUT | When the stack is unable to send a packet in TX window |
| RX_DONE | When a packet is received |
| RX_ERROR | A general RX error |

The application must attach an event handler to the stack. The LoRaWANInterface provides an API to attach various callbacks to the stacks. One such callback is the event handler callback.

Application callbacks

The Mbed LoRaWAN stack currently maps 3 different callbacks:

| Callback type | Description |
|------------------------------|---|
| Event callback | Mandatory, Direction: from stack to application |
| Link check response callback | Optional, Direction: from stack to application |
| Battery level callabck | Optional, Direction: from application to stack |

An example of attaching your event handler to the stack:

```
Simplicity IDE - IoraCxxProject/src/IoraWan.cpp - Simplicity Studio ™
<u>F</u>ile <u>E</u>dit <u>S</u>ource Refac<u>t</u>or <u>N</u>avigate Se<u>a</u>rch <u>P</u>roject <u>R</u>un <u>W</u>indow <u>H</u>elp
☐ Launcher {} Simpli
Project Explorer 🛭 🗀 🥞 🔻 🗀 🖟 main.cpp

▼ SoraCxxProject [GNU ARM v7.2.1] [EFM32PG1B200F2
    > 🐉 Binaries
                                                loraWan::loraWanEventsCallback ( lorawan_event_t event )
     ncludes 🗿
   > 🗁 BSP
   > 🗁 CMSIS
                                                    // Treat Events Callback here
     Drivers
     🗁 emlib
                                                    switch (event) {
   > 👝 GNU ARM v7.2.1
                                                        case CONNECTED:
                                                           break:
   > 😝 mbed-os-feature-lorawan-1-1
                                                        case DISCONNECTED:
     pmbed-semtech-lora-rf-drivers
   🗸 🔑 src
                                                        case TX DONE:
      > 🔑 mbed-port
> 🖟 loraWan.cpp
                                                            break;
                                                        case TX_TIMEOUT:
      > h loraWan.h
                                                            break;
      > 🖟 main.cpp
                                                        case TX_ERROR:
      > c sensor.c
                                                            break;
      > h sensor.h
                                                        case CRYPTO_ERROR:
                                                        break;
case TX_SCHEDULING_ERROR:
                                                        case RX DONE:
                                                            break;
                                                        case RX_TIMEOUT:
                                                            break;
                                                        case RX_ERROR:
                                                            break:
                                                        case JOIN_FAILURE:
                                                            break;
                                                        case UPLINK_REQUIRED:
                                                            break;
                                                        case AUTOMATIC UPLINK ERROR:
                                                            break;
                                                        case CLASS_CHANGED: //only in Lorawan 1.1 (ch 18.1)
                                                            break;
                                                        case SERVER_ACCEPTED_CLASS_IN_USE: //only in Lorawan 1.1 (ch 18.1)
                                                            break:
                                                        case SERVER_DOES_NOT_SUPPORT_CLASS_IN_USE: //only in Lorawan 1.1 (ch 18.1)
                                                            break:
                                                        case DEVICE TIME SYNCHED: // only in LoRaWAN v1.0.3 and v1.1.x
                                                            break;
                                                        default:
                                                            break;
```

Link check response handler

Link check request is a MAC command defined by the LoRaWAN specification. To receive the response of this MAC command, set the link check resp callback.

Battery level handler

The battery level callback is different from others. The direction of this callback is from the application to the stack. In other words, it provides information to the stack. The application is reponsible for letting the stack know about the current battery level.

Error codes

All operations on LoRaWANInterface return an error code $lorawan_status_t$ that reflects success or failure of the operation.

Below is the list of error codes and their description.

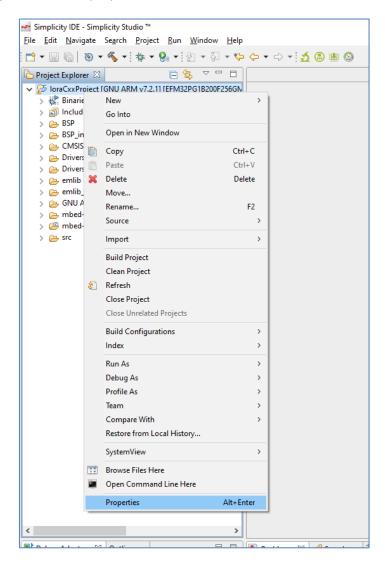
| Error code | Value | Description |
|---------------------------------------|-------|--|
| LORAWAN_STATUS_OK | 0 | Service done successfully |
| LORAWAN_STATUS_BUSY | -1000 | Stack busy |
| LORAWAN_STATUS_WOULD_BLOCK | -1001 | Stack cannot send at the moment or have nothing to read |
| LORAWAN_STATUS_SERVICE_UNKNOWN | -1002 | Unknown service request |
| LORAWAN_STATUS_PARAMETER_INVALID | -1003 | Invalid parameter |
| LORAWAN_STATUS_FREQUENCY_INVALID | -1004 | Invalid frequency |
| LORAWAN_STATUS_DATARATE_INVALID | -1005 | Invalid frequency and datarate |
| LORAWAN_STATUS_FREQ_AND_DR_INVALID | -1006 | When stack was unable to send packet in TX window |
| LORAWAN_STATUS_NO_NETWORK_JOINED | -1009 | Device is not part of a network yet (Applicable only for OTAA) |
| LORAWAN_STATUS_LENGTH_ERROR | -1010 | Payload length error |
| LORAWAN_STATUS_DEVICE_OFF | -1011 | The device is off, in other words, disconnected state |
| LORAWAN_STATUS_NOT_INITIALIZED | -1012 | Stack not initialized |
| LORAWAN_STATUS_UNSUPPORTED | -1013 | Unsupported service |
| LORAWAN_STATUS_CRYPTO_FAIL | -1014 | Crypto failure |
| LORAWAN_STATUS_PORT_INVALID | -1015 | Invalid port |
| LORAWAN_STATUS_CONNECT_IN_PROGRESS | -1016 | Connection in progress (application should wait for CONNECT event) |
| LORAWAN_STATUS_NO_ACTIVE_SESSIONS | -1017 | No active session in progress |
| LORAWAN_STATUS_IDLE | -1018 | Stack idle at the moment |
| LORAWAN_STATUS_DUTYCYCLE_RESTRICTED | -1020 | Transmission will be delayed because of duty cycling |
| LORAWAN_STATUS_NO_CHANNEL_FOUND | -1021 | No channel is enabled at the moment |
| LORAWAN_STATUS_NO_FREE_CHANNEL_FOUND | -1022 | All channels marked used, cannot find a free channel at the moment |
| LORAWAN_STATUS_METADATA_NOT_AVAILABLE | -1023 | Metadata is stale, cannot be made available as its not relevant |

Modifying LoRaWAN stack

Various parameters for Mbed LoRaWAN stack can be configured via either C++ APIs or by using the Mbed configuration system, editing the mbed_app.json file in the root of their application. This is explained here: https://os.mbed.com/docs/mbed-os/v5.13/reference/lorawan-configuration.html

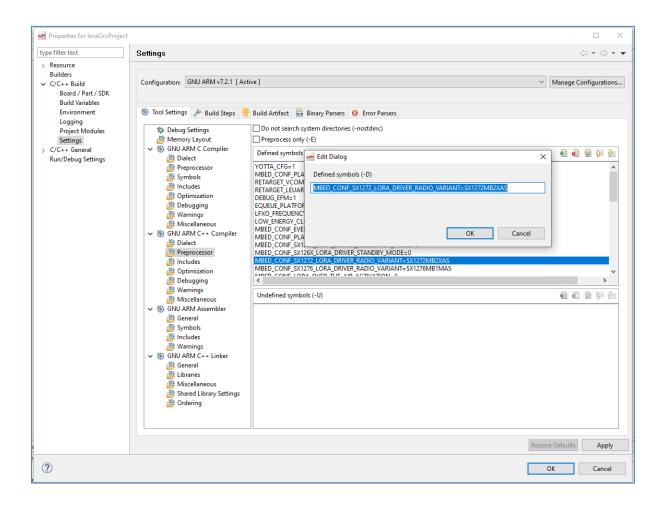
All those configurations were mapped to the preprocessor symbols. Also some definitions found inside the stack must be configured that were not in the mbed_app.json. It is user responsibility to modify those values.

Right click the project and select properties:



And then browse to C/C++ Build / Settings / CNU ARM C++ Compiler / Preprocessor.

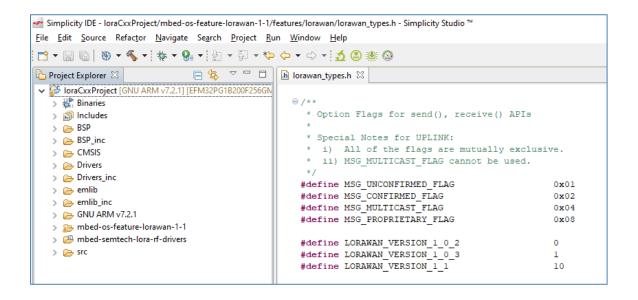
To change any parameter, just double click the desired setting:

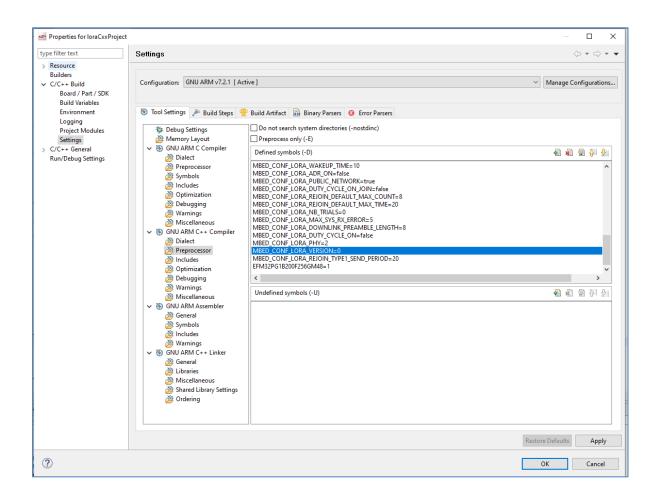


Following are the most relevant configuration items.

LoRaWAN Version

Locate the directive MBED_CONF_LORA_VERSION and change to one of the valid values (0, 1 and 10). In this example we are using the 1.0.2 specification:

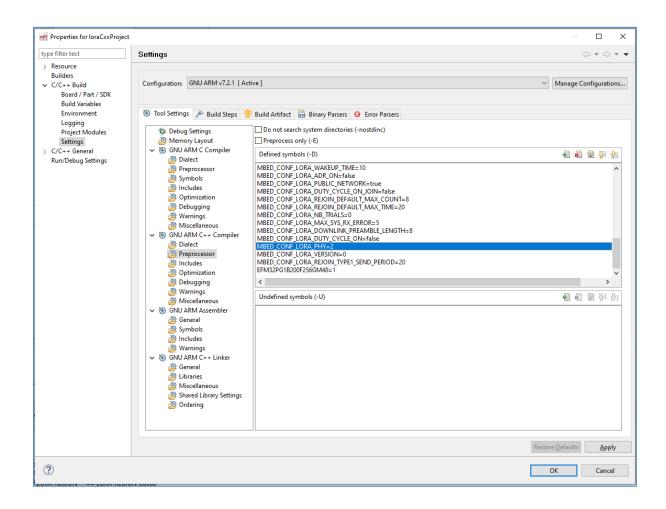




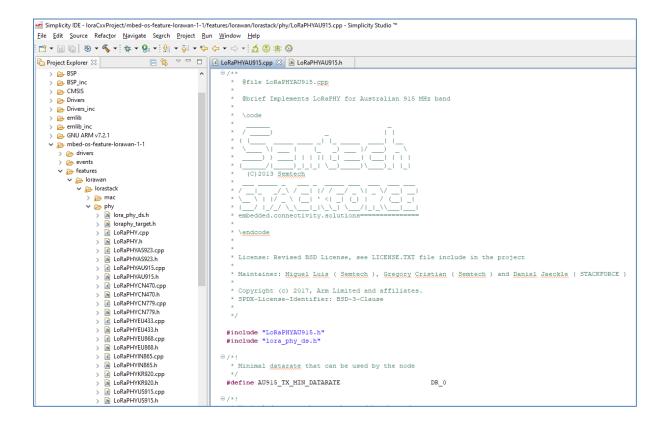
LoRaWAN Phy Region

Locate the directive MBED_CONF_LORA_PHY and change to one of the valid values (0 till 8). In this example we are using the AU915 specification:

```
LORA REGION EU868
                        0
LORA REGION AS923
                        1
LORA REGION AU915
                        2
LORA REGION CN470
                        3
LORA REGION CN779
                        4
LORA REGION EU433
                        5
LORA REGION IN865
                        6
                        7
LORA REGION KR920
LORA REGION US915
```

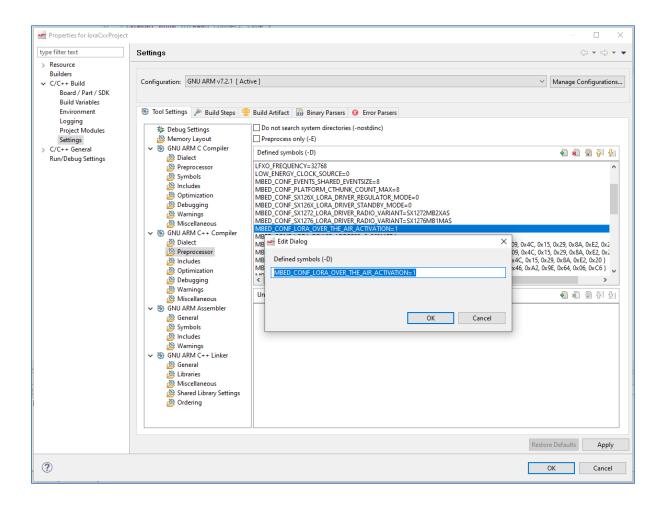


To change specific behaviours for your project, locate and modify the correspondent parameter for your phy region at "mbed-os-feature-lorawan-1-1\features\lorawan\lorastack\phy" folder.



LoRaWAN Activation Mode

To select the activation mode, modify the directive MBED_CONF_LORA_OVER_THE_AIR_ACTIVATION to one of the valid values (0 for ABP and 1 for OTAA).



LoRaWAN ID / Keys

To edit the network credentials locate and edit these parameters:

MBED_CONF_LORA_DEVICE_ADDRESS

ABP activation method

MBED_CONF_LORA_NWKSENCKEY

MBED_CONF_LORA_SNWKSINTKEY

MBED_CONF_LORA_NWKSKEY

MBED CONF LORA APPSKEY

OTAA activation method

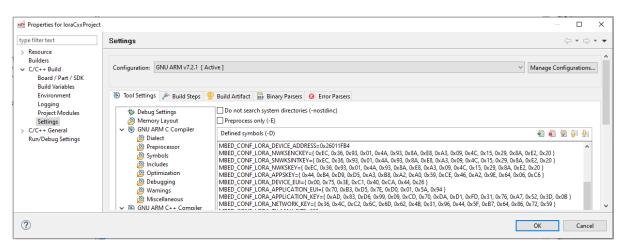
MBED_CONF_LORA_DEVICE_EUI

MBED_CONF_LORA_APPLICATION_EUI

MBED_CONF_LORA_APPLICATION_KEY

MBED_CONF_LORA_NETWORK_KEY

Example:



High volume production requires a minor modification on the source code to insert automated values.

LoRaWAN Class Mode

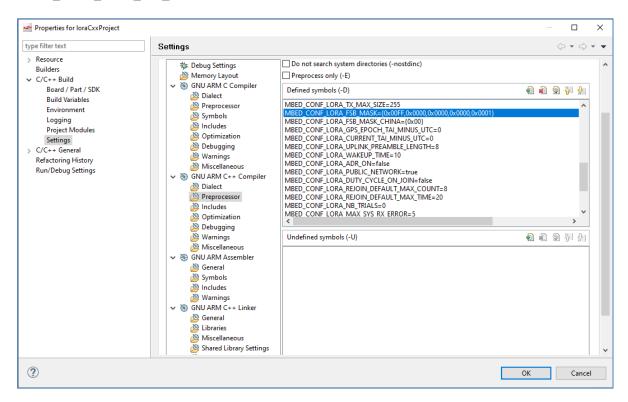
To select the class mode, when configuring the LoRaWANInterface instance call the set_device_class function.

Valid values are CLASS_A or CLASS_C:

```
IoraWan.cpp ☒
       m_lorawanAppCallbacks.events
                                             = mbed::callback( loraWanEventsCallback );
       m_lorawanAppCallbacks.link_check_resp = mbed::callback( loraWanCheckResponseCallback );
       m_lorawanAppCallbacks.battery_level = NULL;
       m_lorawanInterface.add_app_callbacks( &m_lorawanAppCallbacks );
       m connection.connect type = LORAWAN CONNECTION ABP;
       m_connection.connection_u.abp.nwk_id = 1;
       m_connection_u.abp.dev_addr = MBED_CONF_LORA_DEVICE_ADDRESS;
       m_connection.connection_u.abp.app_skey = AppSKey;
m_connection.connection_u.abp.nwk_skey = NwkSKey;
       m_connection.connection_u.abp.nwk_senckey = NwkSKey;
       m_connection_u.abp.snwk_sintkey = NwkSKey;
       m lorawanInterface.disable adaptive datarate();
       m_lorawanInterface.set_datarate( DR_2 );
       m_lorawanInterface.set_device_class( CLASS_A );
       m lorawanInterface.connect( m connection );
```

LoRaWAN Channels

The RF channels used by the stack are configured modifying the preprocessor symbol MBED_CONF_LORA_FSB_MASK.



Make sure to look at mbed-os-feature-lorawan-1-1\features\lorawan\FSB_Usage.txt for important details on how to configure the channels to talk to the LoraWan gateway.

The file content is below:

Frequency sub-bands in US915/AU915:

 ${\tt US915/AU915}$ PHYs define channel structures that can support up to 72 channels for upstream.

The first 64 channels (0-63), occupy 125 kHz and the last 8 channels (64-71) occupy 500 kHz.

However, most of the base stations available in the market support $8\ \mathrm{or}\ 16$ channels.

Network acquisition can become costly if the device has no prior knowledge of the active channel plan and it enables all 72 channels to begin with.

The LoRaWAN 1.0.2 Regional parameters specification refers to a strategy of probing a set of nine channels (8+1) for the joining process. According to that strategy, the device alternatively selects a channel from a set of 8 125 kHz channels and a 500 kHz channel.

For example, send a join request alternatively on a randomly selected channel from a set of 0-7 channels and channel 64, which is the first 500 kHz channel.

Once the device has joined the network (in case of OTAA) or has sent the first uplink (in the case of ABP), the network may send a LinkAdrReq MAC command to set the channel mask to be used. Please note that these PHY layers do not support CFList, so LinkAdrReq is the way the network tells you what channel plan to use.

That means all channels are active. In other words, 64 125 kHz channels and 8 500 kHz channels are active. If you wish to use a custom FSB, you need to set an appropriate mask as the value of "fsb-mask". For example, if you wish to use the first FSB, in other words, the first 8 125 kHz channels (0-7) and the first 500 kHz channel:

```
"fsb-mask" = "\{0x00FF, 0x0000, 0x0000, 0x0000, 0x0001\}"
```

Similarly, if you wish to use the second FSB, in other words, the second set of 8 125 kHz channels (8-15) and the 2nd 500 kHz channel:

```
"fsb-mask" = "\{0xFF00, 0x0000, 0x0000, 0x0000, 0x0002\}"
```

You can also combine FSBs if your base station supports more than 8 channels. For example:

```
"fsb-mask" = "\{0x00FF, 0x0000, 0x0000, 0xFF00, 0x0081\}"
```

means use channels 0-7(125 kHz) + channel 64 (500 KHz) and channels 56-63 (125 kHz) + channel 71 (500 kHz).

Please note that for Certification requirements, you need to alternate between $125~\mathrm{kHz}$ and $500~\mathrm{kHz}$ channels, so before joining, do not set a mask that enables only $500~\mathrm{kHz}$ or only $125~\mathrm{kHz}$ channels.

Frequency sub-bands in CN470 PHY:

The LoRaPHYCN470 class defines 96 channels in total, as per the LoRaWAN Regional Specification. These 96 channels are 125 kHz wide each and can be subdivided into 6 sub-bands containing 16 channels each.

"fsb-mask-china" is the parameter that lorawan/mbed_lib.json defines. It can be used to enforce an FSB. It is defined as a C-style array, and the first element of the array corresponds to first 8 channels (0-7) and so on. By default, all 96 channels are enabled, but there may be base stations that do not support all 96 channels. Therefore, network acquisition can become cumbersome if the device hops on random channels. The probability of finding a correct channel for a base station that supports 8 channels would be 1/12.

For example, if your base station supports 16 channels (channels 0-15), set the "fsb-mask-china" as:

```
"fsb-mask-china" = "{0xffff, 0x0000, 0x0000, 0x0000, 0x0000}"
```

Similarly, if your base station supports 8 channels (channels 0-7), set the "fsb-mask-china" as:

```
"fsb-mask-china" = "\{0x00FF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000\}"
```

Modifying LoRaWAN Radio

When instantiating the LoRadio object, call the appropriate constructor. For example:

```
SX1272_LoRaRadio radio ( SX127X_SPI_MOSI,
                         SX127X_SPI_MISO,
                         SX127X SPI CLK.
                         SX127X SPI CS,
                         SX127X RESET,
                         SX127X_SPI_DIOO,
                         SX127X_SPI_DIO1,
                         SX127X_SPI_DIO2,
                         SX127X_SPI_DIO3,
                         SX127X_SPI_DIO4,
                         SX127X_SPI_DIO5 );
   or
 SX126X LoRaRadio radio( SX126X SPI MOSI,
                         SX126X SPI MISO,
                         SX126X_SPI_CLK,
                         SX126X SPI CS,
                         SX126X RESET,
                         SX126X DIO1,
                         SX126X BUSY,
                         SX126X FREQ SELECT,
                         SX126X DEVICE SELECT,
                         SX126X CRYSTAL SELECT,
                         SX126X ANT SWITCH );
```

And depending on the type of radio, configure the correspondent parameters with the values on the following screenshots.

MBED_CONF_SX126X_LORA_DRIVER_REGULATOR_MODE=0 MBED_CONF_SX126X_LORA_DRIVER_STANDBY_MODE=0

```
Simplicity IDE - IoraCxxProject/mbed-semtech-Iora-rf-drivers/SX126X/sx126x_ds.h - Simplicity Studio ™
<u>F</u>ile <u>E</u>dit <u>S</u>ource Refac<u>t</u>or <u>N</u>avigate Se<u>a</u>rch <u>P</u>roject <u>R</u>un <u>W</u>indow <u>H</u>elp
                                                                                                                       ☐ Launcher {} Simplicity ID
☐ 🧐 ▽ □ □ 🖟 main.cpp 🖟 SX126X_LoRaRadio.cpp 🖟 sx126x_ds.h 🛭
Project Explorer 🖂

    ForaCxxProject [GNU ARM v7.2.1] [EFM32PG1B200F.

   > 👯 Binaries
                                                  \ensuremath{^{\star}} \brief Declares the oscillator in use while in standby mode
   > 🛍 Includes
   > 👝 BSP
   > E CMSIS
                                                  * Using the STDBY_RC standby mode allow to reduce the energy consumption
                                                  * STDBY_XOSC should be used for time critical applications
   > 👝 Drivers
     ≽ emlib
                                               tvpedef enum {
   > GNU ARM v7.2.1

    mbed-os-feature-lorawan-1-1

                                                                                                 = 0x01,
                                                     STDBY XOSC

▼ 

™ 

mbed-semtech-lora-rf-drivers

                                                } radio_standby_mode_t;
      > # SX1272
      > 5 SX1276
      * \brief Declares the power regulation used to power the device
        > [h] sx126x_ds.h
        > & SX126X_LoRaRadio.cpp
                                                  * This command allows the user to specify if DC-DC or LDO is used for power regulation.
        > h SX126X_LoRaRadio.h
                                                  * Using only LDO implies that the Rx or Tx current is doubled
          mbed_lib.json
           SleepMode.txt
                                               etypedef enum {
                                                     USE_LDO
USE_DCDC
        LICENCE-BSD-3-Clause
                                                                                                 = 0x00, // default
                                                                                                  = 0x01,
        ■ LICENSE
        README.md
                                                } radio_regulator_mode_t;
```

MBED_CONF_SX1272_LORA_DRIVER_RADIO_VARIANT=SX1272MB2XAS

```
Simplicity IDE - IoraCxxProject/mbed-semtech-Iora-rf-drivers/SX1272/SX1272_LoRaRadio.cpp - Simplicity Studio ™
File Edit Source Refactor Navigate Search Project Run Window Help
🖹 🤄 ▽ 🗆 🗖 🖟 main.cpp 🖟 SX1272_LoRaRadio.cpp 🛭
跲 Project Explorer 🛭

▼ SoloraCxxProject [GNU ARM v7.2.1] [EFM32PG1B200F2
                                           #define XTAL FREQ
                                                                                               32000000
                                           #define FREQ STEP
  > 🔐 Binaries
                                                                                               61.03515625
  > 🔊 Includes
  > 🗁 BSP
  > 🗁 CMSIS
                                          enum RadioVariant {
  > 📂 Drivers
                                              SX1272UNDEFINED = 0,
  > 📂 emlib
                                               SX1272MB2XAS,
  > 👝 GNU ARM v7.2.1
                                               SX1272MB1DCS
  > 🚌 mbed-os-feature-lorawan-1-1
  ⊕ /*1
       > 📂 registers
                                            * FSK bandwidth definition
       > 🖟 SX1272_LoRaRadio.cpp
       > In SX1272_LoRaRadio.h
                                          ⊕ typedef struct
         mbed_lib.json
         README.md
                                              uint32 t bandwidth;
     > 🕮 SX1276
                                              uint8_t register_value;
     > 🚌 SX126X
                                           } fsk_bw_t;
       LICENCE-BSD-3-Clause
```

MBED_CONF_SX1276_LORA_DRIVER_RADIO_VARIANT=SX1276MB1MAS

```
Simplicity IDE - IoraCxxProject/mbed-semtech-lora-rf-drivers/SX1276/SX1276_LoRaRadio.cpp - Simplicity Studio ™
<u>F</u>ile <u>E</u>dit <u>S</u>ource Refactor <u>N</u>avigate Se<u>a</u>rch <u>P</u>roject <u>R</u>un <u>W</u>indow <u>H</u>elp
🔛 👔 Launcher
Project Explorer 🛭 🕒 🤄 🤝 🗀 🗋 📵 main.cpp 🖟 SX1272_LoRaRadio.cpp 🖟 SX1276_LoRaRadio.cpp 🗵

▼ SoraCxxProject [GNU ARM v7.2.1] [EFM32PG1B200F2
                                                #define SIG_DIO4
                                                                    0x10
                                                #define SIG_DIO5
   > 👯 Binaries
                                                                     0x20
                                                #define SIG TIMOUT 0x40
   > 🛍 Includes
   > 🗁 BSP
   > 👝 CMSIS
                                              ⊕ /**
                                                * Radio hardware registers initialization
   > 👝 Drivers
   > 降 emlib
                                                static const radio_registers_t radio_reg_init[] = RADIO_INIT_REGISTERS_VALUE;
   > 👝 GNU ARM v7.2.1
   > 📻 mbed-os-feature-lorawan-1-1
                                              enum RadioVariant {

        mbed-semtech-lora-rf-drivers

                                                    SX1276UNDEFINED = 0,
      SX1276MB1LAS,
        > 🗁 registers
                                                    SX1276MB1MAS
        > C SX1272_LoRaRadio.cpp
        > In SX1272_LoRaRadio.h
          mbed_lib.json
                                                #ifdef MBED_SX1276_LORA_RADIO_SPI_FREQUENCY
           README.md
                                                #define SPI_FREQUENCY MBED_SX1276_LORA_RADIO_SPI_FREQUENCY
     #else
                                                #define SPI_FREQUENCY
                                                                          8000000
        > 📂 registers
        > & SX1276_LoRaRadio.cpp
        > h SX1276_LoRaRadio.h
           mhed lih ison
                                                using namespace mbed;
```

Running Demo Applications

There are two examples projects to demo demonstrate the stack usage.

LoRaWAN + Pearl Gecko

A simple app to test connectivity has been made, also to demonstrate how to use the stack in C++. It connects via OTAA or ABP and transmits temperature and humidity every 20 seconds.

Using the starter kit SLSTK3401A (EFM32PG1B200F256GM48) and SX1272MB2xAS the code size is about 156K and 100K with optimization set to -O1.



LoRaWAN + Blue Gecko

To demonstrate the concurrent BLE/LoraWan operation on the EFR32BG device, we based on the thermometer example and also demonstrate how to use the stack mixing C/C++. After the mobile connects with the module, it begins transmitting data every 20 seconds.

Using the starter kit SLWSTK6020B (EFR32BG13P632F512GM48) and SX1272MB2xAS the code size is about 320K and 267K with optimization set to -O1.

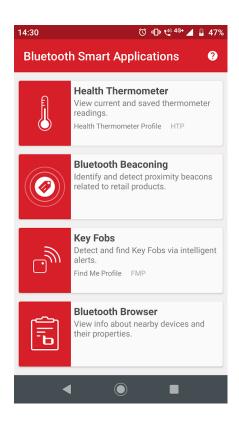


Follow the steps below:

1. Download the Blue Gecko android app here:

https://play.google.com/store/apps/details?id=com.siliconlabs.bledemo

2. Enable the Bluetooth, open the app and click on "Health Thermometer".



3. You should see the device "Silabs BLE LoraWan #1". Click on it to connect. This string is located on the gatt_db.c file, on array bg_gattdb_data_attribute_field_10_data, which should be automatically generated when configuring the .isc file of the project.



4. When connected, the kit will turn on LED1 and toogle LED0, and it begins transmitting data every 20 seconds.

