
# OMx02

User guide of LoRaWAN module

Specification Version 1.7

# Catalog

# 1. Introduction

## 1.1 Overview

Based on a low-power consumption MCU (MKL17x with cortext-M0+) and SX1276/SX1278, OMx02 module opens hardware resources to facilitate developers' secondary development with ManThink-developed real-time operating system named Man-Pregnante Operating System(MPOS). The development becomes convenient by strong support of our library file "MPOS_LWSx02" according to LoRaWAN protocol and the interfaces of SPI , IIC, UART or GPIO. Product based on OMx02 don't need another MCU which can down the cost of the product and the period of the R&D.

MPOS_LWSx02 is a LoRaWAN protocol stack base on the hardware of OMx02, which is enable the function of Class A, Class B and Class C and integarated with a lightweight real-time operating system(MPOS). It provides functions to LoRaWAN underlying operations and periodic task for application development.

Below is the features of MPOS&LWS:

- Easy task based on precise time
- Support semaphore
- Support once event
- Support interrupt
- Supply precise UTC time
- Supply the driver of MCU

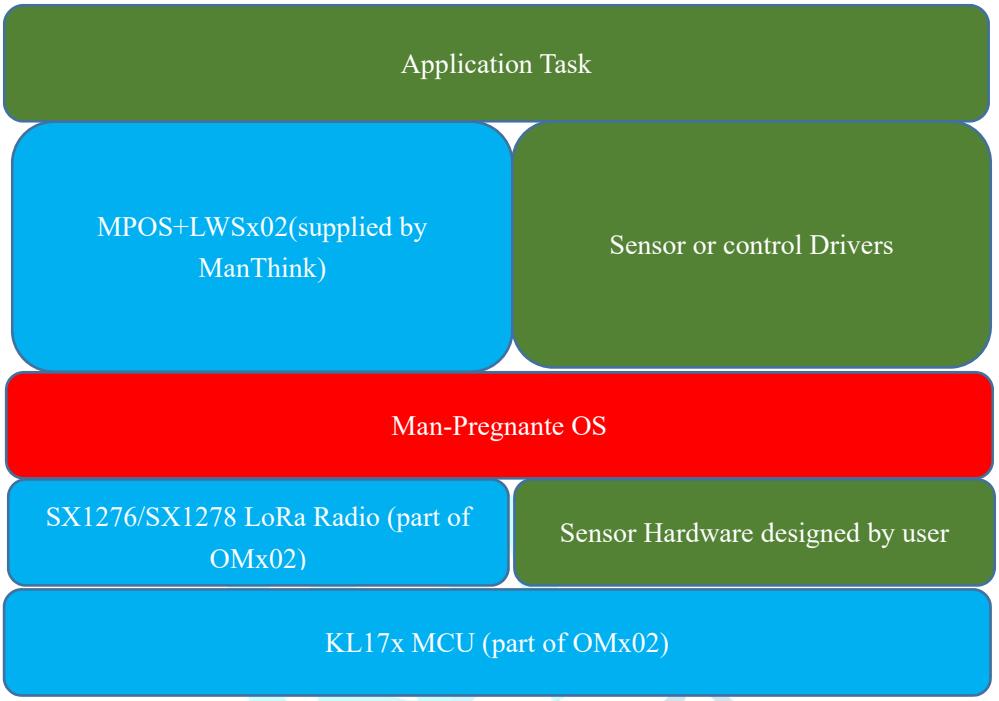
- Support re-writing the MCU's driver
- Support multi-bin
- LWS support ClassA，ClassB and ClassC of LoRaWAN by API
- Support SW mode
- Support FUOTA

Some underlying driver and hardware control are also necessary for the LoRaWAN application development except for the MPOS_LWSx02 supplied by ManThink. All the other drivers and codes would not be included in this document and supported by ManThink.

## 1.2 Featrue

- support Class A, Class B,Class C （LoRaWAN1.0.2, version lite don't supprt ClassB）
- Real-time OS integrated(Man-Pregnante Operating System)
- Easy to build a period-running task
- Abundant interface (SPI，IIC，UART，GPIO)

## 1.3 Architecture



# 2. Overview of SDK

SDK of OMx02 is constructed by library(MPOS_LWSx02),header files and example project based on EWARM. SDK supplies API function to realize the interface to operate the function of system and LoRaWAN.

MPOS uses a event-based programing model to notify developer the event of LoRaWAN and has a real-time OS to manage the event of LoRaWAN and task of developer.

## 2.1 Forbidden resource

Some resource of OMx02 have already been used by the stack which is forbidden for developer. Using the forbidden resource which are listed below will cause unknown issues.

- PA1 , PA2 , PA4
- PB0
- PC5,PC6
- PE0,PE1,PE16,PE17,PE18,PE19
- SPI0 , PIT , LPTIM

## 2.2 Information of different SDK version

| | version | Supported hardware | Function | |
|---|---|---|---|---|
| 1 | MPOS_LWS402lite | OM402,OM402S | MPOS,ClassA,ClassC,<br>SW mode<br>410-510MHz | |
| 2 | MPOS_LWS802lite | OM802,OM802S | MPOS,ClassA,ClassC,<br>SW mode<br>860-1020MHz | |
| 3 | MPOS_LWS402 | OM402S | MPOS,ClassA,ClassB, ClassC<br>SW mode, broadcast, FUOTA,<br>multi-bin<br>,410-510MHz | |
| 4 | MPOS_LWS802 | OM802S | MPOS,ClassA,ClassB,ClassC,<br>SW mode, broadcast, FUOTA,<br>multi-bin<br>860-1020MHz | |
| 5 | MPOS_LWS411 | OM411 | MPOS,ClassA,ClassB, ClassC<br>SW mode, broadcast, FUOTA,<br>multi-bin<br>,410-510MHz | |
| 6 | MPOS_LWS811 | OM811 | MPOS,ClassA,ClassB,ClassC,<br>SW mode, broadcast, FUOTA,<br>multi-bin<br>860-1020MHz | |

# 3. Man-Pregnant Operating System

Man-Pregnant OS ( MP OS) is quite likely to fit well with the lower power consumption and less hardware resources, which could run smoothly on cortex-M0 with the kernel codes less than 5kB. MPOS is a high precision time sheduling OS, the periodic task,interrupt task and one-off event supplied by it can make the IoT application development quickly and easily. The One-off event could implement the hardware trigger event, while the periodic task could realize the codes executed periodically.

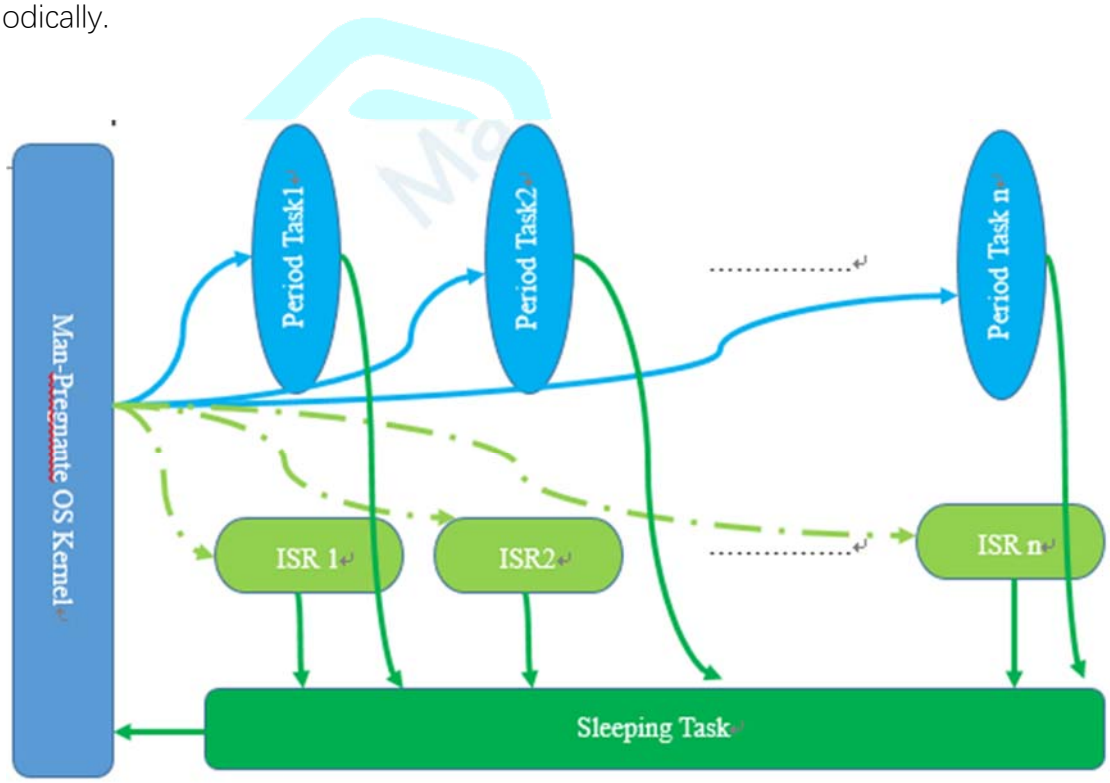



Image 2: Architecture of MPOS

MPOS will make MCU sleeping when there is no task to be excuted automaticlly. Developer don't need to concern the strategy of low power mode,MPOS manage status of all tasks and will awake at the precise time to excute the task.MPOS will make the MCU sleeping after completed the excuting task.

The current of the sleeping is less than 3uA and the time from awake to sleeping is less than 1milliseconds.

## 3.1 One-off Task

One-off task will be excuted once and linked to a function once it is enabled. There are three one-off events in MPOS available to developers, which will be excuted in 5ms if there is no currently running periodic task.

## 3.2 Periodic Task

The periodic Task would be excuted periodically with the time precision of 5 mS± 50ppm per second. The developer should set the interval in milliseconds and link the function to the task.

Periodic task can be configued to a special cycling times, and the times to be excuted can be modified dynamiclly to suit different senario.

The cycling counts of periodic task can be modified when the task is excuting, the modified parameters will be valid after the excuting.

## 3.3 Interrupt Task

The developer could write code through the interface of hardware interrupt function provided by MP OS. There are three one-off events in MP OS, users can enable one of them and connect it to their own fuction. In addition, users can write code directly in the interrupt function to realize the highest-priority response.

## 3.4 Semaphore

The inter-task communications are implemented by MPOS using global varibles, so there is no mutual exclusion. As all of the task's priorities are time-based sequency, there is no task can preempt the MCU from a current running one.

MPOS supply Semaphore to protect the hardware recourse and global variable to avoid the collison between different tasks.

## 3.5 Priority

There are three different priorities in MPOS. The highest priority task can be implemented in the interrupt function as developers write simple codes in ISR(Interrupt Service Routines). The one-off event is recommended to be realized in the interrupt function because the complex codes will reduce the OS efficiency.

The second priority is the periodic task. As all of the task's priorities are time-based sequency, the earliest task will be excuted firstly and there is no task can preempt the MCU from a current running one.

The next priority task is one-off event. Several hook functions are defined by MPOS, which will be excuted only once as it is enabled when there is no interrupt and periodic tasks. The hook function would be disabled until it is linked to a new function and then enable to be executed again.

# 4. API of MPOS

## 4.1 Function bits of MPOS

Different function supplied by MPOS enabled by different function bits, Developer can modify the function bits in the global variable <StackFunction>.

### 4.1.1 StackFunction.watchdog

The watch-dog would be enabled if the value is set to 1, application codes should kick the watch dog in 8 seconds. The value can be set to 0 to disable watch dog when debuging.

### 4.1.2 StackFunction.nosleep

MPOS would go into a virtual sleeping mode to simulate the normal fu nction when degbuging if the value of the bit is set to 1.

MPOS would go into sleeping if no task to be excuted automatically if t he value of this bit is set to 0 to realize the low-power application. The tim e to go into sleeping after the task excuting is less than 1 milliseconds.

The bit should be set to 0 in released product.

### 4.1.3 StackFunction.uart1

MPOS would intialize the UART1 internally if the bit is set to 1. Develop er can use the API of UART1 to realize the application.

### 4.1.4 StackFunction.lorawan

MPOS would enable the function of LoRaWAN if the bit is set 1.

### 4.1.5 StackFunction.MTprotocol

UART1 can receive datas when MCU is asleep if this bit is set to 1 and

the communication protocol of UART1 is the same with ManThink's. The MPOS can parse the data frame automaticlly for the developer's further anal ysis. If the bit is set to 0, the UART1 can't receive any data as MCU is aslee p , otherwise, MPOS will forward the frame to users as MCU keeps awake.

## 4.2 API of system

API function is designed for system hardware configuration.

### 4.2.1 mpos_driver.Clock_Init

Initialize the parameters of MCU and set the frequency of main clock to 48MHz. then all of the outside clock are base on this frequency.

void **mpos_driver.Clock_Init ()。**

### 4.2.2 mpos_driver.mcureset

To reset MCU by this API function.

**void mpos_driver.mcureset ()。**

### 4.2.4 mpos_driver.kickdog

The developer can kick dog based on this API. The task created by the developer must include a kicking dog operation when the watch dog is enabled. Meantime, the application programs have to kick dog by this API periodically. As the time of watch dog is set for 8 seconds, the task built by users should include a kicking dog operation. The MPOS will implement it automatically.

void **mpos_driver.kickdog ()。**

### 4.2.5 mpos_osfun.SysParaInit

The function will read all the parameters stored in flash then initialize the MPOS.

void **mpos_osfun.SysParaInit ()。**

### 4.2.6 mpos_osfun. DelayUs

The delay function of system, unit is microsecond.

void mpos_osfun. DelayUs (unsigned long nCount)。

nCount: time to be delayed。

4.2.7 mpos_osfun. enableInterrupts

Enable the global interrupt.

void mpos_osfun. enableInterrupts ().

4.2.8 mpos_osfun. disableInterrupts

Disable the global interrupt.

void mpos_osfun. disableInterrupts ().

## 4.3 API of utility

### 4.3.1 mpos_osfun. memcpy1

Copy data from source address to the destanation.

void mpos_osfun. memcpy1 (uint8_t *dst, uint8_t *src, uint16_t size)

dst: address of destanation

src: address of source

size: data size to be copied.

### 4.3.2 mpos_osfun. os_rlsbf2

Convert two bytes to a 16bits-data of little-endian.

u2_t mpos_osfun. os_rlsbf2 (u1_t *buf )

buf: address of the bytes

return value : 16bits-data of little-endian

### 4.3.3 mpos_osfun. os_rlsbf4

Convert four bytes to a 32bits-data of little-endian.

u4_t mpos_osfun. os_rlsbf4 (u1_t *buf )

buf: address of the bytes

return value : 32bits-data of little-endian.

### 4.3.4 mpos_osfun. os_rmsbf4

Convert four bytes to a 32bits-data of big-endian.

u4_t mpos_osfun. os_rlsbf4 (u1_t *buf )

buf: address of the bytes

return value : 32bits-data of big-endian.

#### 4.3.5 mpos_osfun. os_wlsbf2

Convert a 16bits-data of little-endian to 2 bytes.

void mpos_osfun. os_wlsbf2 (u1_t *buf , u2_t v)

buf: address of the bytes

v : 16bits-data of little-endian.

#### 4.3.6 mpos_osfun. os_wlsbf3

Convert a 24bits-data of little-endian to 3 bytes.

void mpos_osfun. os_wlsbf2 (u1_t *buf , u4_t v)

buf:address of bytes.

v : 32bits-data of little-endian

#### 4.3.7 mpos_osfun. os_wlsbf4

Convert a 32bits-data of little-endian to 4 bytes.

void mpos_osfun. os_wlsbf2 (u1_t *buf , u4_t v)

buf:adress of bytes

v : 32bits-data of little-endian.

#### 4.3.8 mpos_osfun. os_wlsbf8

Convert a 64bits-data of little-endian to 8 bytes.

void mpos_osfun. os_wlsbf2 (u1_t *buf , u8_t v)

buf: address of bytes

v : 64bits-data of little-endian.

#### 4.3.9 mpos_osfun. os_wmsbf3

Convert a 24bits-data of big-endian to 3 bytes.

void mpos_osfun. os_wmsbf3 (u1_t *buf , u4_t v)

buf: address of bytes

v : 32bits-data of big-endian.

4.3.10 mpos_osfun. os_wmsbf4

Convert a 32bits-data of big-endian to 4 bytes.

void mpos_osfun. os_wmsbf4 (u1_t *buf , u4_t v)

buf:address of bytes

v : 32bits-data of big-endian.

4.3.12 mpos_osfun. mp_modeACRC

Compute the crc of bytes.

u2_t mpos_osfun. mp_modeACRC (unsigned char *buf,unsigned int lenth,u2_t poly)

buf: address of bytes to be computed

v : size of the byte

poly: parameters of CRC

return value : result of the CRC

4.3.13 mpos_osfun. crc8_ccit

Compute the crc of bytes.

u1_t mpos_osfun. crc8_ccit (const uint8_t * data, unsigned size)

data: address of bytes to be computed

size: size of the byte

return value : result of the CRC

## 4.4 API of periodic task

This interface supplied by MP OS for developer to start a periodic task, which can satisfy some basic functions of IoT devices such as read sensor, send data and control GPIO. After creating periodic task, the task would be excuted periodicly, the cycling counts can be set by developer and can be modified dynamically.

The periodic task can be defined by S_periodTask structure, in which one task will be managed by four variables.

- Task

  This callback function will be excuted if time out.

- Interval

  The interval of the task is millisecond with the pricision of 5mS+50ppm/s.

- Cycles

  Counts of the task would be excuted, the value would be descreased by 1 after excuting once automatically. The task would stop to be excuted if the value is 0.

  The task would be excuted infinitely if the value is set to 0xFFFFFFFF.

### 4.4.1 mpos_osfun.Task_Setup

The task would be installed into the system after excute the function.

void mpos_osfun.Task_Setup (S_periodTask * task)

task : Indicator of the task

#### 4.4.2 mpos_osfun.Task_SetPeriod

To set the period of the task.

void mpos_osfun. Task_SetPeriod (S_periodTask * task,clock_time_t period)

task: Indicator of the task

Period: period to be set . unit is millisecond.

#### 4.4.3 mpos_osfun.Task_Restart

To restart the task.

void mpos_osfun.Task_Restart (S_periodTask * task)

task: Indicator of the task

#### 4.4.4 mpos_osfun.Task_Sleep

To modify the period of the task , the function should be used in the task.

void mpos_osfun. Task_Sleep (clock_time_t period)

period: period to be set . unit is millisecond.

#### 4.4.5 mpos_osfun.Task_ExcuteNow

Task would be excuted immediately.

void mpos_osfun. Task_ExcuteNow (S_periodTask * task)

task:Indicator of the task.

#### 4.4.6 mpos_osfun.Task_Stop

Task would be stoped.

void mpos_osfun.Task_Stop (S_periodTask * task)

task: Indicator of the task.

### 4.4.7 mpos_osfun. Task_Remove

To remove the task from the system.

void mpos_osfun. Task_Remove (S_periodTask * task)

task: Indicator of the task.

## 4.5 Semaphore

Semaphore is designed to protect the resource of system. For exsample, there is a senario to use the UART1 resource in different task, the schedule d task should protect the resource of UART1 by semaphore.

Semphore is defined by SEMPHORE .

### 4.5.1 Task_sem_init

To intialize the semaphore to a count. The semahore must be a value w hich is bigger than 0 before used.

void mpos_osfun.Task_sem_init(SEMPHORE * mysem, int counts)

mysem : Indicator of semaphore

counts: counts of semaphore

### 4.5.2 Task_sem_release

Release the semaphore after using the resource.

void mpos_osfun.Task_sem_release(SEMPHORE * mysem)

mysem : Indicator of semaphore

### 4.5.3 Task_sem_waitOne

To Check the valid of semaphore, the resource is available if the return value is true, otherwise the task would be excuted again after the specified t ime to check the resource until the semaphore is released.

The waiting time can be set to small if the task priority is higher.

bool mpos_osfun.Task_sem_waitOne(S_periodTask* task,SEMPHORE* mysem,s8_t waitTime)

task: Indicator of task

mysem : Indicator of semaphore

waitTime: the waiting time to check the semphore nexttime. unit: millisecond。

## 4.6 API of One-off task

There are three one-off events in MP OS, which will be excuted once if they are enabled.

### 4.6.1 Enable bit of one-off task

The following three bits in LPInfo.wakingBit.Bits to enable/disable the single-events.

- LPInfo.wakingBit.Bits.OnceEvent1
- LPInfo.wakingBit.Bits.OnceEvent2
- LPInfo.wakingBit.Bits.OnceEvent3

The following three hook functions could lead users' function to the one-off event.

- void (* HookExcuteOnce1)()
- void (* HookExcuteOnce2)()
- void (* HookExcuteOnce3)()

### 4.6.2 excution process of one-off task

There are two steps to enable the one-off task. Firstly, setting its bit and then link the developers' function to the specific hook function.

Example:

```
LPInfo.wakingBit.Bits.OnceEvent1=1;

HookExcuteOnce1=OnceEventTest1;
```

## 4.7 Interrupt

Five interrupt events are started by MP OS base on KL17, whose interrupt function can be implemeted by five hook functions.

### 4.7.1 HOOK_USART1_IRQHandler

This hook function is for defining UART1. It is set to be a internal function of MPOS to implement UART1 interface and can be reset to users' function if the UART1 function need to be reprogrammed.

### 4.7.2 HOOK_USART2_IRQHandler

This hook function is for the interrupt of UART2. It is set to be a internal function of MPOS to implement UART2 interface and can be reset to users' function if the UART2 function need to be reprogrammed.

### 4.7.3 HOOK_RTC_IRQHandler

This function defines the RTC interrupt hook function. The developers could realize the RTC interrupt function by setting it .

### 4.7.4 HOOK_EXTIPORTBCDE_WKIRQHandler

This hook function defines the external interruption and wakes the sleeping MCU up.

### 4.7.5 HOOK_EXTIPORTBCDE_GNIRQHandler

This hook function defines the external interrupt function but cannot wake up the sleeping MCU. Whereas it defineds several interrupt sources in SysStatus.Bits.WkIntBits.

- Bit0 means PB0 trigger an interrupt

- Bit1 means PC1 trigger an interrupt
- Bit4 means PC5 trigger an interrupt
- Bit5 means PC6 trigger an interrupt
- Bit6 means PD4 trigger an interrupt
- Bit7 means PD6 trigger an interrupt

## 4.8 wake-up bit

MCU will go to sleep deeply with no task running, so MP can reserve some wake-up bits for developers to define wake-up events. Normal wake-up bit and gentle wake-up bit are two kinds of setting modes. The former one will put MCU in a deep sleep and the latter one will put it into a light sleep.

### 4.8.1 LPInfo.wakingBit.UserEvent & LPInfo.wakingBit.RFU

- Size of LPInfo.wakingBit:

  Size of UserEvent is 1, size of LPInfo.wakingBit.RFU is five, MCU would not go into sleep if one of the bits is not-zero.

RFU is reserved for developer, if developer want to make MCU awake according to their application, developer should make some bit of the RFU to be 1. If developer want the MCU sleeping, please confirm that all of RFU is 0.

### 4.8.2 LPInfo. lightWakingBit.UserEvent & LPInfo.lightWakingBit.RFU

- Size of LPInfo.lightWakingBit:

  Size of UserEvent is 1 and the size of LPInfo.lightWakingBit.RFU is 9, If one of the bits is not-zero, MCU would go into light sleeping to down the current of the MCU.

# 5. API of MCU peripheral driver

Drivers provided by us for MKL17 to operate the sensors connecting to OMx02 include Flash, GPIO and UART. More types drivers will be provided by us in the future. Besides, the developer can write your own application drivers or contact with us to add your necessary if it is not in the library.

## 5.1 Flash

MPOS reserved two flash block for developer:APP and database.

APP block reserved 200 bytes for developer which can be used by API directly. (6.1.12 and 6.1.13 )。

DataBase reserved 1022 bytes, the offset address is: 2-1023. Realized read and write operation by mpos_driver.Database_Read and mpos_driver.Database_Write .

### 5.1.1mpos_driver.Database_Read

To read the data from DataBase Block.

unsigned char mpos_driver.Database_Read(unsigned int Addr,unsigned char *RxBuffer,unsigned char Length)

Addr: Offset address :2-1023.

RxBuffer: to get the data of read back.

Length : size of the data to be read

### 5.1.2 mpos_driver.Database_Write

To write data to the database block.

unsigned char mpos_driver.Database_Write (unsigned int Addr,unsigned char *RxBuffer,unsigned c

har Lenth)

Addr: Offset address :2-1023.

RxBuffer: data to be written to the database blcok

Length : size of data to be written

## 5.2 GPIO

### 5.2.1 mpos_driver.GPIO_Config

For configuring the working mode of GPIO.

void mpos_driver.GPIO_Config (GPIO_TypeDef* gpioport, unsigned char pin, unsigned short gpio cfg)

gpioport : The value of target port to be operated is GPIOA or GPIOB

pin : the value of target pin to be operated is 0-7

gpiocfg : the mode of GPIO based on the file "MT_KL17_GPIO.h"

### 5.2.2 mpos_driver.GPIO_PinSet

Output GPIO pin signal.

void mpos_driver.GPIO_PinSet (GPIO_TypeDef* ***gpioport,*** unsigned char ***pin,*** unsigned char ***state***)

***gpioport*** : The value of target port to be operated is GPIOA or GPIOB

***pin*** : the value of target pin to be operated is 0-7

***state*** : the high and low levels ouput from pin can be indicated with 1 or 0.

### 5.2.3 mpos_driver.GPIO_PinGet

Retrieve the status of GPIO pin.

unsigned char mpos_driver.GPIO_PinGet (GPIO_TypeDef* ***gpioport***, unsigned char ***pin***)

***gpioport*** : The value of target port to be operated is GPIOA or GPIOB

*pin* : the value of target pin to be operated is 0-7

*return value:* Retrieve the status of GPIO pin, 0 and 1 indicate to low and high signal level respectively.

5.2.4 mpos_driver.GPIO_IT_Config

For the configuration of GPIO interrupt.

void mpos_driver.GPIO_PinGet (GPIO_TypeDef* gpioport , unsigned char pin, GPIO_ITConfig_Type gpiocfg)

gpioport : The value of target port to be operated is GPIOA or GPIOB

pin : the value of target pin to be operated is 0-7

gpiocfg: the interrupt mode, please refer "MT_KL17_GPIO.h"

5.2.5 mpos_driver.GPIO_IT_Diable

To disable the interrupt of GPIO

void mpos_driver.GPIO_IT_Diable (GPIO_TypeDef* gpioport , unsigned char pin)

gpioport : The value of target port to be operated is GPIOA or GPIOB

pin : the value of target pin to be operated is 0-7

## 5.3 UART1

5.3.1 mpos_driver.UART1_Init

For UART1 intialization

void mpos_driver.UART1_Init(uint32_t baudrate)

*baudrate :* defines the baudrate of the uart.

**Example:** UART1_Init(9600);

### 5.3.2 mpos_driver.UART1_SendBuffer

UART1 sends a buffer.

unsigned char mpos_driver.UART1_SendBuffer(unsigned char* data,unsigned short len)

*data:* defines the start address of the buffer sent by UART1

*len:* defines the size of the buffer

## 5.4 UART2

### 5.4.1 mpos_driver.UART2_Init

For UART2 intialization

void mpos_driver.UART2_Init(uint32_t baudrate)

*baudrate :* defines the baudrate of the uart.

**Example:** UART2_Init(9600);

### 5.4.2 mpos_driver.UART2_SendBuffer

UART2 sends a buffer.

unsigned char mpos_driver.UART2_SendBuffer(unsigned char* data,unsigned short len)

*data:* defines the start address of the buffer sent by UART2

*len:* defines the size of the buffer

## 5.5 SPI

### 5.5.1 mpos_driver.Spi_Init

For SPI intialization.

void mpos_driver.Spi_Init (SPI_Type* SPIx)

SPIx : Select the specific SPI interface

Example : mpos_driver.Spi_Init (SPI1);

### 5.5.2 mpos_driver. SPI_Cmd

To enable or disable SPI.

void mpos_driver. SPI_Cmd(SPI_Type* SPIx,FunctionalState NewState)

SPIx : Select the specific SPI interface

NewState: ENABLE or DISABLE

Example: SPI_Cmd(ENABLE);

### 5.5.3 mpos_driver. SpiInOut

Sending or reading data by SPI.

unsigned char mpos_driver.SpiInOut(SPI_Type* SPIx,unsigned char outData)

SPIx : Select the specific SPI interface

outData: data to be sent

Return value: data read from the slave device

Example: unsigned char getdata= mpos_driver.SpiInOut(SPI1,0xFF);

## 5.6 ADC

### 5.6.1 mpos_driver.ADC_GetTempValue

Get the temperature of the chipset.

uint8_t mpos_driver.ADC_GetTempValue (void)

Return value: temperatrue+100, unit: ℃。

## 5.7 RTC

### 5.7.1 mpos_driver.RTC_ConvertSecondsToDatetime

Convert seconds of UTC to the format of year+month+day+hour+minitu e+second.

void mpos_driver.RTC_ConvertSecondsToDatetime (uint32_t seconds, T_RTC_DATETIME *datetime)

seconds: UTC time, unit:second

datetime: format of year+month+day, refer"MyInline.h"

### 5.7.2 mpos_driver.RTC_ConvertDatetimeToSeconds

Convert the format of year+month+day+hour+minitue+second to UTC time.

uint32_t mpos_driver.RTC_ConvertDatetimeToSeconds (T_RTC_DATETIME *datetime)

datetime: format of year+month+day, refer"MyInline.h"

Return value: UTC time, unit:second

### 5.7.3 mpos_driver.RTC_SetDatetime

Set the time of RTC.

void mpos_driver.RTC_SetDatetime (T_RTC_DATETIME *datetime)

datetime: format of year+month+day, refer"MyInline.h"

### 5.7.4 mpos_driver.RTC_GetDatetime

Get the time of UTC

void mpos_driver.RTC_GetDatetime (T_RTC_DATETIME *datetime)

datetime: format of year+month+day, refer"MyInline.h"

### 5.7.5 mpos_driver.RTC_SetSencond

Set the time of RTC.

void mpos_driver.RTC_Set Sencond (uint32_t sencond)

sencond: UTC time, unit: second

### 5.7.6 mpos_driver.RTC_GetSencond

Get the time of RTC

uint32_t mpos_driver.RTC_Get Sencond (void)

Return value: UTC time, unit: second

# 6. API of LoRa/LoRaWAN

LoRa&LoRaWAN stack is provided by ManThink based on the library named MPOS_LWSx02,which can support ClassA,ClassB and ClassC according to LoRaWAN protocol.The application layer of the stack can support broadcast, FUOTA,SW mode(please refer different SDK version). Developer can transmmite and receive LoRa packet by API function.

The stack can support PDA and relay working mode to suit different sen ario which can be enable or disable by function bits.

LWS is a LoRa+LoRaWAN stack based on MPOS, the operation of RF is realized by API. The data received from RF and the event of LoRaWAN is notified by event.

## 6.1 API of LWS

### 6.1.1 mpos_lws.LWS_init

Used to initialize the stack, developers should run it at the application starts.

void mpos_lws.LWS_init (void)

### 6.1.2 mpos_lws. LoRaWANInit

Used to initialize the parameters of LoRaWAN, the API must be excuted if developer want to use the function of LoRaWAN.

void mpos_lws.LoRaWANInit (void)

### 6.1.3 mpos_lws.LWS_SetDR

To Set the air datarate of LoRaWAN.

void mpos_lws.LWS_SetDR (uint8_t DR)

DR: Please refer the LoRaWAN spec.

Example: mpos_lws.LWS_SetDR(0)

### 6.1.4 mpos_lws.LWS_SetFDD

To select the working mode of LoRaWAN. Different mode supported by differrent standard.Please refer the spec of LoRaWAN.

void mpos_lws. LWS_SetFDD (unsigned char status)

status: 0: TDD, 1: FDD。

Example: mpos_lws. LWS_SetFDD (0)

### 6.1.5 mpos_lws.LWS_SetADR

To enable or disable the ADR function.

void mpos_lws.LWS_SetADR (unsigned char status);

status: status= 0: disable the ADR function, 1: enable ADR function

Example: mpos_lws.LWS_SetADR (1), enable ADR function

### 6.1.6 mpos_lws.LWS_SetServerADR

To enable or disable the ADR function of LoRaWAN server. If enable the function of Server ADR, the datarate of device will be decided by Server.

void mpos_lws.LWS_SetServerADR (unsigned char status);

status: status= 0: enable ADR function by device,

1: enable ADR function by server.

Example: mpos_lws.LWS_SetServerADR (1), Enable the function of Server

### 6.1.7 mpos_lws. LWS_SetLinkCheck

Enable or disable the function of linkcheck.

void mpos_lws. LWS_SetLinkCheck (unsigned char status);

status: status= 0: disable the function of linkcheck, 1: enable the function

of linkcheck.

Example: mpos_lws. LWS_SetLinkCheck (1), enable function of LinkCheck.

6.1.8 mpos_lws. paraFWSetAPPEUI

Set the APPEUI of LoRaWAN.

void mpos_lws. paraFWSetAPPEUI (u1_t * para,u8_t appeui);

para: the internal parameters

Appeui: Appeui to be set

Example: mpos_lws.paraFWSetAPPEUI(paraFwReg.SFwRegister.AppEui,0x4001 01000B000301);

6.1.9 mpos_lws. paraRDSetFreq

To set the frequency parameters into the module.

void mpos_lws. paraRDSetFreq (u1_t *para, u4_t freq,u1_t band);

para: the internal parameters

freq: frequency to be set

band: band to be set

Example: mpos_lws.paraRDSetFreq(paraRdReg.SRdRegister.Freq[0].SFreq,4703 000000,0);

6.1.10 mpos_lws. paraRDSetChannelMap

To set the channelmap parameter to the module

void mpos_lws. paraRDSetChannelMap (u1_t *para, u2_t map);

para: the internal parameter

map: parameter to be set

Example: mpos_lws. paraRDSetChannelMap (paraRdReg.SRdRegister.channel Map,0x00FF);

6.1.11 mpos_lws. paraFWSetDevKey

Used to set APPKey into the module.

void mpos_lws. paraFWSetDevKey (u1_t *para, u1_t *appkey);

para: the internal parameters

appkey: appkey to be set

Example: mpos_lws. paraFWSetDevKey (paraFwReg.SFwRegister.DevKey,DevKey)

6.1.12 mpos_lws.paraAPPGet

Get the APP parameters from the module

void mpos_lws. paraAPPGet (u1_t *para ,u1_t len);

para: APP parameters of the module

len: size of the parameters

Example: mpos_lws. paraAPPGet (u1_t * para,u1_t len)

6.1.13 mpos_lws.paraAPPSave

Wirte the APP parameters into the module

void mpos_lws. paraAPPSave (u1_t *para ,u1_t len);

para: parameters to be written.

len: size of the parameters to be written

Example: mpos_lws. paraAPPSave (u1_t * para,u1_t len)

6.1.14 mpos_lws.LWS_SetClassMode

To set the working mode of LoRaWAN

void mpos_lws.LWS_SetClassMode (uint8_t calssMode)

classMode: ClassA、ClassB and ClassC

Example: mpos_lws.LWS_SetClassMode (Class_A);

### 6.1.15 mpos_lws.JoinReset

Join the LoRaWAN network

LWERRO_t mpos_lws.JoinReset (LWOP_tmode)

mode: =LWOP_REJOIN

Return value: the result of operation

Example: mpos_lws.JoinReset (LWOP_REJOIN);

### 6.1.16 mpos_lws.LWS_GetUPseq

Get the uplink seq number of LoRaWAN.

u4_t mpos_lws.LWS_GetUPseq();

### 6.1.17 mpos_lws.LWS_GetUTCTime

Get the UTC time .

u4_t mpos_lws.LWS_GetUTCTime();

### 6.1.18 mpos_lws.LWS_SetDeviceTimeReq

If the API function is excuted, the stack will retrieve the UTC time fr om server.

void mpos_lws.LWS_SetDeviceTimeReq();

## 6.2 Register of FW

the firmware parameters of LWS stack can be defined by FW register, read and modified by the API of [mpos_lws.ParaFWGet] & [mpos_lws.ParaFWSave]. It is suggested to rerun the stack to make sure the parameters setting correctly.

### 6.2.1 FW registers table

| Register | Length(octet) | type | Description |
|---|---|---|---|
| Module Type | 1 | Unsigned char | Read-only |
| Hardware version | 1 | Unsigned char | Read-only |
| Firmware version | 1 | Unsigned char | Read-only |
| DevEUI | 8 | Unsigned char array | Read-only |
| APPEUI | 8 | Unsigned char array | Read-only |
| DevAddr | 4 | Unsigned char array | Refer LoRaWAN spec |
| NetID | 4 | Unsigned char array | Refer LoRaWAN spec |
| NwkSKey | 16 | Unsigned char array | Refer LoRaWAN spec |
| AppSKey | 16 | Unsigned char array | Refer LoRaWAN spec |
| APPKey | 16 | Unsigned char array | Refer LoRaWAN spec |
| RxWinDelay1 | 1 | Unsigned char | the delay time of the first receiving window |
| RxWinDelay2 | 1 | Unsigned char | the delay time of the second receiving windo |
| JoinDelay1 | 1 | Unsigned char | The delay time of the first window of confirmd packet. |
| JoinDelay2 | 1 | Unsigned char | the delay time of the second window of confmed packet |
| ReTx | 1 | Unsigned char | The retransmission number of the uplink package |
| globalDutyRate | 1 | Unsigned char | Global duty ratio |
| DutyBand[4] | 8 | Unsigned char array | Duty cycle of 4 bands |
| DevTimeReqDuty | 1 | Unsigned char | Get UTC time from server after the counts of uplink packets |
| ShadowBits | 1 | Unsigned char | Reserved |
| RFU | 19 | Unsigned char | Reserved |

### 6.2.2 Structure of FW registers

```
typedef struct t_CRO_FW
{
   unsigned char MType;
   unsigned char HwVersion;
   unsigned char FmVersion;
   unsigned char DevEui[8];
   unsigned char AppEui[8];
   unsigned char DevAddr[4];
   unsigned char NetID[4];
   unsigned char NwkSKey[16];
   unsigned char APPSKey[16];
   unsigned char DevKey[16];
   unsigned char RxWinDelay1;
   unsigned char RxWinDelay2;
   unsigned char JoinDelay1;
   unsigned char JoinDelay2;
   unsigned char ReTx;
   unsigned char globalDutyRate;
   unsigned char DutyBand[4][2];
   unsigned char DevTimeReqDuty;
   U_FW_SHDOW    shadowBits;
   unsigned char RFU[19];
} PACKED T_CRO_FW;
```

### 6.2.3 mpos_lws. paraFWGet

Getting the firmware parameters

void mpos_lws. paraFWGet (U_CROFW *para);

para: **get the the address of parameter variable from LWS stack**

Example: mpos_lws. paraFWGet (&paraFwReg)

### 6.2.4 mpos_lws. paraFWSave

Save the parameters of firmware to the Flash.

void mpos_lws. paraFWSave (U_CROFW *para);

para: the address of varibles has been stored in LWS stack.

Example: mpos_lws. paraFWSave (&paraFwReg)

## 6.3 Radio registers of LWS

The RF parameters are defined by Radio registers of the LWS stack, which can be modified by developers with the API of [mpos_lws.ParaRDGet] and [mpos_lws.ParaRDSave]. It is suggested to restart the stack to make sure the parameters setting correctly.

### 6.3.1 Table of Radio registers

| Register | Length(octet) | type | Description |
|---|---|---|---|
| dn2Feq | 4 | Unsigned char arra | Define the frequency of Rx Window2. Little Endian with the unit in hertz, i.e. 43350000Hz<br>Value=0x60 0xAF 0xD6 0x19 |
| dlsetting | 1 | Unsigned char | Define the data rate of Rx Window 2 |
| ChannelMap | 2 | Unsigned char arra | The status of active channel(each bit d fines whether use the channel) |
| Power | 1 | Unsigned char | RF transmit power is from -5 to 20 wi h the unit in dBm (Power)<br>Actual RF transmit power = Power- 2d m |
| Freq[n] | 4 | Unsigned char arra | Define the frequency of each channel. Little Endian with the unit in hertz, i.e. 43350000Hz<br>Value=0x60 0xAF 0xD6 0x19; n=0~16 |
| DrRange[n] | 1 | Unsigned char arra | Allowed frequency in Freq[n], n=0~16; bit0~bit3 and bit4~bit7 define the min, max DR in Freq[n].<br>Please refer to the LoRaWAN spec |
| channelMapcntl | 2 | Unsigned char arra | Reserved |

### 6.3.2 Structrue of Radio registers

```
typedef struct t_CRO_RD
{
   unsigned char     dn2Freq[4];
   U_DLSETIING dlsetting;
   unsigned char     channelMap[2];
   unsigned char     Power;
   S_FREQDR    Freq[16];
   u2_t aliCHcntl;
}PACKED T_CRO_RD;
typedef union u_CRO_RD
{
   T_CRO_RD SRdRegister;
   unsigned char Bytes[88];
}PACKED U_CRORD;
```

### 6.3.3 mpos_lws.paraRDGet

Get the RF parameters written in the Flash.

void mpos_lws.paraRDGet (U_CRORD *para);

para: get the the address of parameter variable from LWS stack.

Example: mpos_lws.paraRDGet (&paraRdReg)

### 6.3.4 mpos_lws.paraRDSave

Store RF parameters to the Flash.

void mpos_lws.paraRDSave (U_CRORD *para);

para: the address of varible need to be stored in LWS stack.

Example: mpos_lws.paraRDSave (&paraRdReg)

## 6.4 CF registers of LWS

The parameters of LoRaWAN function can be defined by CF register and modified by developers to change the system working mode. Besides, it can be obtained and set by [mpos_lws.paraCFGet] and [mpos_lws.paraCFSave] respectively.

### 6.4.1 Table of CFregisters

| Register | Size | Type | Description |
|---|---|---|---|
| exCNF | 2 | 2 bytes | |
| | | | Bit0=1: keep the value as 1<br>Bit1=1: keep the value as 1<br>Bit2=1: enable multicast<br>Bit3=1: enable SW mode<br>OM402:<br>Bit4~Bit7=0: CN470/EU433<br>Bit4~Bit7=1: LinkWAN<br>OM802:<br>Bit4~Bit7=0: EU868<br>Bit4~Bit7=1: US902<br>Bit4~Bit7=2: KR923<br>Bit4~Bit7=3: AS920<br>Bit4~Bit7=4: AU915<br>Bit4~Bit7=5: CN920<br>Bit8=1: reserved<br>Bit9=1: enable RSSI output<br>Bit10=1: enable SNR output<br>Bit11=1: enable relay function<br>Bit12=1: DwellTime<br>Bit13=1: reserved for linkwan<br>Bit14=1: keep rxing when SW mode<br>Bit15=1: normal operation when SW<br>Bitx=0: disable the function |
| MulDevAddr | 4 | Unsigned char array | For multicast |
| MulNSessionKey | 16 | Unsigned char array | For multicast |
| MulASessionKey | 16 | Unsigned char array | For multicast |

| | | | |
|---|---|---|---|
| BeaconFreq | 4 | Unsigned char array | Frequency of Beacon |
| BeaconDR | 1 | Unsigned char | DR of Beacon |
| BeaconPeriod | 1 | Unsigned char | Period of Beacon= (BeaconPeriod*128)<br>unit: second |
| BeaconAirTime | 4 | Unsigned char array | Air time of Beacon<br>Example: SF=9, 144580 ms<br>SF=10, 305150 ms |
| pingFreq | 4 | Unsigned char array | Frequency of Pingslot |
| pingDR | 1 | Unsigned char | DR of Pingslot |
| pingIntvExp | 1 | Unsigned char | Counts of Pingslot=2^ pingIntvExp |
| MulPingFreq | 4 | Unsigned char array | Frequency of multicast ping |
| MulPingDR | 1 | Unsigned char | DR of multicast Pingslot |
| MulPingIntvExp | 1 | Unsigned char | Counts of multicast Pingslot =2^ pingIntvExp |
| SWSF | 1 | Unsigned char | SF of SW mode |
| SWBW | 1 | Unsigned char | BW of SW mode<br>7: BW=125kHz<br>8: BW=250kHz<br>9: BW=500kHz |
| SWFreq | 4 | Unsigned char array | Frequency of SW |
| SWPeriod | 1 | Unsigned char | Period of SW=50ms* SWPeriod |
| uprepeat | 1 | Unsigned char | Counts of retransmitted for unconfirm packets |
| JoinCHMap | 2 | Unsigned char array | For linkwan |
| DRRange | 1 | Unsigned char | Bit0~Bit3: the min DR<br>Bit4~Bit7: the max DR |
| RelayWin2 | 1 | Unsigned char | Time of the second window in relay mode<br>unit: second |
| RelayFreqMap | 2 | Unsigned char array | Channel in relay mode (defined by bit: on or off) |
| RelayDR | 1 | Unsigned char | DR of relay mode |
| RelayPeriod | 1 | Unsigned char | Wake up time in relay mode<br>= RelayPeriod*50 unit: ms |
| RelayDelayTx | 1 | Unsigned char | RFU |
| MaxDwellTime | 1 | Unsigned char | Bit0~Bit3: unit:dBm<br>Bit4: enable the uplink DwellTime |

| | | | |
|---|---|---|---|
| | | | Bit5: enable the downlink DwellTime<br>Bit6~Bit7: RFU |

### 6.4.2 Structure of CF registers

```
typedef union
{
   u2_t ByteS;
   struct
   {
      u2_t window1Enable :1;
      u2_t window2Enable :1;
      u2_t MulPingEnable :1;
      u2_t SleepWakeEnable:1;
      u2_t Standard:4;
      u2_t Tmode:1;
      u2_t RssiEnable:1;
      u2_t SNREnable:1;
      u2_t RelayEnable:1;
      u2_t DwellTime:1;
      u2_t aliLowDR:1;
      u2_t SWRXON   :1;
      u2_t SWTXF    :1;
   } PACKED Bits;
}  U_EXSTACKCNF;

typedef union
{
   u1_t ByteS;
   struct
   {
      u1_t MaxEIRP :4;
      u1_t UplinkDwellTime :1;
      u1_t DownlinkDwellTime :1;
      u1_t RFU:2;
   } PACKED Bits;
}  U_MAXDWELLTIME;

typedef struct t_CRO_CF
{
      U_EXSTACKCNF      exStackFunction;
      unsigned char     MulDevAddr[4];
      unsigned char     MulNSessionKey[16];
      unsigned char     MulASessionKey[16];
      unsigned char     BeaconFreq[4];
      unsigned char     BeaconDR;
      unsigned char      BeaconPeriod;
```

```
    unsigned char    BeaconAirTime[4];
    unsigned char    pingFreq[4];
    unsigned char    pingDR;
    unsigned char    pingIntvExp;
    unsigned char    MulPingFreq[4];
    unsigned char    MulPingDR;
    unsigned char    MulPingIntvExp;
    unsigned char    SWSF;
    unsigned char    SWBW;
    unsigned char    SWFreq[4];
    unsigned char    SWPeriod;
    unsigned char    uprepeat;
    unsigned char    JoinCHMap[2];
    unsigned char    DRRange;
    unsigned char    RelayWin2;
    unsigned char    RelayFreqMap[2];
    unsigned char     RelayDR;
    unsigned char    RelayPeriod;
    unsigned char    RelayDelayTx;
    U_MAXDWELLTIME    MaxDwellTime;
}PACKED  T_CRO_CF;

typedef union U_CRO_CF
{
    T_CRO_CF  CFRegister;
    unsigned char  Bytes[82];
}PACKED  U_CROCF;
```

### 6.4.3 mpos_lws.paraCFGet

Get the parameters of CF register from the Flash.

void mpos_lws.paraCFGet (U_CROCF *para);

para: get the the address of parameter variable from LWS stack.

Example: mpos_lws. paraCFGet (&paraCFReg)

### 6.4.4 mpos_lws.paraCFSave

Save the parameters of CF registers.

void mpos_lws.paraCSave (U_CROCF *para);

para: the the address of parameter variable is stored in LWS stack.

Example: mpos_lws.paraCFSave (&paraCFReg)

### 6.4.5 mpos_lws.paraCFmodify

Update the modified parameters to the stack.

void mpos_lws. paraCFmodify (U_CROFW *para);

para: the the address of parameter variable is stored in LWS stack.

Example: mpos_lws. paraCFmodify (&paraCFReg)

## 6.5 Transmitte or receive RF data

### 6.5.1 mpos_lws.LW_TxData

Transmit data to gateway according to LoRaWAN protocol.

LWERRO_t mpos_lws. LW_TxData (uint8_t * txBuffer,u1_t lenth,u1_t port,LWOP_t mode)

txBuffer: send the start address of buffer

lenth: the byte length of transmitting data

port: the port number according to LoRaWAN protocol

mode:

LWOP_LTC mean the confirmed packet

LWOP_LTU mean the unconfirmed packet

LWOP_SWTmean the SW packet. Under the SW mode, port must be 223 or <200.

Return value: result of the operation

Example: mpos_lws. LW_TxData ((unsigned char*)(Sensor.Bytes),INFOR_LEN, SENSOR_PORT, LWOP_LTC);

## 6.6 Event of LWS

Many events will be triggered based on LoRaWAN protocol and notified to users by Hook_Event.function (mt_ev_t ev ,u1_t port,u1_t * Buffer, u2_t len). The developer need to write a hook function and link it to Hook_Event.function, then the system will inform users once the following events are triggered.

### 6.6.1 list of event

| Event | Description |
|---|---|
| MT_EV_TXDONE | RF data transmission complete |
| MT_EV_JOINED | join network successfully |
| MT_EV_JOIN_FAILED | join network failed |
| MT_EV_REJOIN_FAILED | rejoin network failed |
| MT_EV_TXOVER_NOPORT | the transmmission completed with no downlink data |
| MT_EV_TXOVER_NACK | No necessary ack for the completed transmmission |
| MT_EV_TXOVER_DNW1 | get downlink data in the window1 |
| MT_EV_TXOVER_DNW2 | get downlink data in the window 2 |
| MT_EV_TXOVER_PING | get downlink data at a ping slot under class B |
| MT_EV_BEACON_TRACKED | The beacon is tracked by device |
| MT_EV_BEACON_MISSED | The beacon is lost by device |
| MT_EV_LOST_TSYNC | the time sync is lost by device |
| MT_EV_LINK_ALIVE | the connection between device and server is working |
| MT_EV_LINK_DEAD | The device and server are disconnected |
| MT_EV_MICWRONG | device get a data but its mic is erro |
| MT_EV_NTRX | RF data transmission complete |
| MT_EV_SWRXDONE | join network successfully |

### 6.6.2 Hook_Event.function

This function is a pointer to handle the LoRaWAN events. The developer s need to write function to handle events by themselves then hook to this p ointer.

Hook_Event.function(mt_ev_t ev ,u1_t port,u1_t * Buffer, u2_t len);

*Ev :* define the LoRaWAN event.

*Port:* define the packet port if the event is receiving downlink data from server

*Buffer :* defines the start address of packet if the event is receiving downlink data from server.

*len :* defines the packet length if the event is receiving downlink data from server

# 7. Quick Start

Many functions can be realized by the complex LoRaWAN stack, but it is easy to use in most applications. For quick start, just keep the default para meters and send the packet to server by the API of mpos_lws. LW_TxData. Mean while, receive the downlink data from server by Hook_Event.function and do proc ession.

Please contact to ManThink to obtain the setup methord of default para meter if you don't want to know the details of each parameter, then the Lo RaWAN application can be realized by txdata & Hook_Event.function.

# 8.Contact

Please contact us for more supports and details.

**ManThink Technology Co., Ltd**

Web：www.manthink.cn

BBS : www.loramaker.com

E-mail：info@manthink.cn

Tel：+86-10-5622 9170

Add：Room601 Ronghua International Building No.5, Ronghua South Road No.10, E-town, Beijing, P.R.China