iND83209 SDK User Guide V0.1

1 REVISION HISTORY

Table 1 Revision History

Rev #	Date	Action	By
0.1	08/03/2021	First draft	JackPan

2 TABLE OF CONTENTS

3 SYSTEM OVERVIEW

iND83209 SDK function structure is based on infinite loop with task priority. Task handling is based on event trigger which means that each task would have a flag to figure out the task to be handled.

The priority and the task id should be assigned before use:

```
/*! The list of task IDs. The IDs are sorted according to descending
priority. For each task ID there is the corresponding task handler function. */
typedef enum
{
  TASK_ID_SOFT_TIMER = 0U,      /*!< Task ID of Global timer*/
  TASK_ID_LIN_STACK,            /*!< Task ID of LIN stack*/
  TASK_ID_LINS,                 /*!< Task ID of LIN slave*/
  TASK_ID_PDS,                 /*!< Task ID of persistance data storage*/
#if LIN_MASTER_EN == 1
  TASK_ID_LINM,                 /*!< Task ID of LIN master*/
#endif
  TASK_ID_COLOR_MIXING,         /*!< Task ID of colorMixing*/
  TASK_ID_ADC_MEASURE,       /*!< Task ID of ADC measurement*/
  TASK_ID_SAFETY_MONITOR,       /*!< Task ID of Safety protection*/
  TASK_ID_COLOR_COMPENSATION,   /*!< Task ID of color compensation*/
  TASK_ID_APPL,                 /*!< Task ID of APPlication purpose for debug and demo*/
} TM_TaskId_t;
```

For example, according to the task id definition above, the Soft Timer has the top priority (the task id is 0) which means when more than 1 task is handled, the Soft Timer would be handled firstly until the Soft Timer task is idle.

The infinite loop function keeps checking continuously whether there is any task which is triggered and to be handled.

```
void main(void)
{
    /* !!!!!!!MUST BE called firstly here for initializing system parameters !!!!*/
    PDS_Init();
    /* System init for hardwre init */
    SYS_Init();
    /* system main infinite loop */
    for(;;){
      TM_RunTasks();
    }
}
```

```
void TM_RunTasks(void)
{
    uint8_t taskId;
    for (taskId = 0U; taskId < (uint8_t)(sizeof(taskHandlers) / sizeof(taskHandlers[0])); taskId++){
        if ( (((uint16_t)1UL << taskId) & (taskFlags & tasksMask)) != 0U ){
            __atomic_enter()
                taskFlags &= ~((uint16_t)1UL << taskId);
            __atomic_exit()
            if (taskHandlers[taskId] != NULL){
                taskHandlers[taskId]();
            }
            break;
        }
    }
#if WATCH_DOG_EN == 1U
    WDTA_Clear(); /* Feeding Watch dog */
#endif
    if (taskFlags == 0U){
        IdleTask();
    }
}
```

check task to be handled

execute the task

The task function to be handled by calling TM_PostTask.

```
/*
 *  Disables specified task.
 *  @param [in] taskId - task to be disabled.
 *  @return none
 */
void TM_DisableTask(TM_TaskId_t taskId)
{
    tasksMask &= ~(uint16_t)(1UL << taskId);
}

/*
 *  brief Enables specified task.
 *  @param [in] taskId - task to be enabled.
 *  @return none
 */
void TM_EnableTask(TM_TaskId_t taskId)
{
    tasksMask |= (uint16_t)(1UL << taskId);
}

/*
 *  brief runs specified task.
 *  @param [in] taskId - task to be run.
 *  @return none
 */
void TM_PostTask(TM_TaskId_t taskId)
{
    __atomic_enter()
      taskFlags |= (uint16_t)(1UL << taskId);
    __atomic_exit()
}
```

Disable the task even though the task is post.

Enble the task to enable the TM_PostTask function

Set the task id would trigger the task to be handled.

3.1 TASK

In order to make the code readable and maintainable, task is recommended to be used in this SDK.

Take LINS_Task as an example:

Firstly, add task ID in the definition:

```
/*! The list of task IDs. The IDs are sorted according to descending
priority. For each task ID there is the corresponding task handler function. */
typedef enum
{
  TASK_ID_SOFT_TIMER = 0U,       /*!< Task ID of Global timer*/
  TASK_ID_LIN_STACK,             /*!< Task ID of LIN stack*/
  TASK_ID_LINS,                  /*!< Task ID of LIN slave*/
  TASK_ID_PDS,                  /*!< Task ID of persistance data storage*/
#if LIN_MASTER_EN == 1
  TASK_ID_LINM,                  /*!< Task ID of LIN master*/
#endif
  TASK_ID_COLOR_MIXING,          /*!< Task ID of colorMixing*/
  TASK_ID_ADC_MEASURE,         /*!< Task ID of ADC measurement*/
  TASK_ID_SAFETY_MONITOR,        /*!< Task ID of Safety protection*/
  TASK_ID_COLOR_COMPENSATION,    /*!< Task ID of color compensation*/
  TASK_ID_APPL,                  /*!< Task ID of APPlication purpose for debug and demo*/
} TM_TaskId_t;

typedef enum{
  TASK_STATE_INIT = 0U,
  TASK_STATE_ACTIVE,
  TASK_STATE_PROCESSING,
  TASK_STATE_IDLE,
}TaskState_t;
```

add LINS Task ID

Secondly, add callback function in the register table:

```
typedef void (*taskHandler_t)(void);


static taskHandler_t taskHandlers[] = {
    [TASK_ID_SOFT_TIMER]        = SoftTimer_TaskHandler,/* Global Timer task */
    [TASK_ID_LIN_STACK]         = LinStack_TaskHandler, /*  LIN stack task   */
    [TASK_ID_LINS]              = LINS_TaskHandler,     /*  LIN Slave task   */
    [TASK_ID_PDS]               = PDS_TaskHandler,      /*!< Task ID of persistance data storage*/
#if LIN_MASTER_EN == 1U
    [TASK_ID_LINM]              = LINM_TaskHandler,     /*  LIN Master task */
#endif
    [TASK_ID_COLOR_MIXING]      = CLM_TaskHandler,      /*  Color mixing and control task*/
    [TASK_ID_ADC_MEASURE]       = MES_TaskHandler,      /* ADC measurement, chip temperature, Led PN volt, Battery Volt etc. */
    [TASK_ID_SAFETY_MONITOR]    = SAFM_TaskHandler,     /* safe monitor task */
    [TASK_ID_COLOR_COMPENSATION]= CCP_TaskHandler,      /*  Color compensation task      */
    [TASK_ID_APPL]              = APPL_TaskHandler,     /*  design for debug and demo    */
};
```

Add LINS_TaskHandler to the [TASK_ID_LINS]position in the task handle table

Thirdly, define LINS_TaskHandle function in the LinSlaveTask.c

```
void LINS_TaskHandler(void)
{
    switch(linsTaskState){
    case TASK_STATE_ACTIVE:
      break;
    case TASK_STATE_INIT:
      (void)l_sys_init();
      ls_registerServices(UcndFramesTable, (uint8_t)(sizeof(UcndFramesTable)/sizeof(LIN_Device_Frame_t)), DIAG_DATA_BUFF_SIZE,&linsFramesCallback);
      (void)ld_set_tp_timeout(N_AS, N_CR);
      linsTaskState = TASK_STATE_ACTIVE;
      break;
    default:
      break;
    }
}
```

Task event handle

task init

Generally, task needs to be initialized before starting the task, in the task management, 4 states are defined below:

```
typedef enum{
  TASK_STATE_INIT = 0U,
  TASK_STATE_ACTIVE,
  TASK_STATE_PROCESSING,
  TASK_STATE_IDLE,
}TaskState_t;
```

The initial state of every task is TASK_STATE_INIT, every task would be initialized in the SYS_init function by default:

```
void SYS_Init(void)
{
    /* Init system clock */
    Clock_SystemMainClockInit(CLOCK_RC_16MHz, SYS_MAIN_CLOCK_DIV);
    pmu_init();
    /* Init global timer engine for driving soft timer */
    SysTick_Init(SOFT_TIMER_INTERVAL *1000U * MAIN_CPU_CLOCK, SoftTimer_ExpireCallback);
#if WATCH_DOG_EN == 1U
      WDTA_Enable(WDTA_MODE_RESET, WDTA_INTERVAL_1000MS, NULL); /* 1s */
#endif

    /* Init gpios settings */
    gpios_init();
    /* Init LED current and PWM settings */
    leds_driver_Init();

    /* tasks init must be called before use. */
    TM_PostTask(TASK_ID_SOFT_TIMER);
    TM_PostTask(TASK_ID_SAFETY_MONITOR);
    TM_PostTask(TASK_ID_LINS);
#if LIN_MASTER_EN == 1
    TM_PostTask(TASK_ID_LINM);
#endif
    TM_PostTask(TASK_ID_ADC_MEASURE);
    TM_PostTask(TASK_ID_COLOR_MIXING);
    TM_PostTask(TASK_ID_COLOR_COMPENSATION);
    TM_PostTask(TASK_ID_APPL);
}
```

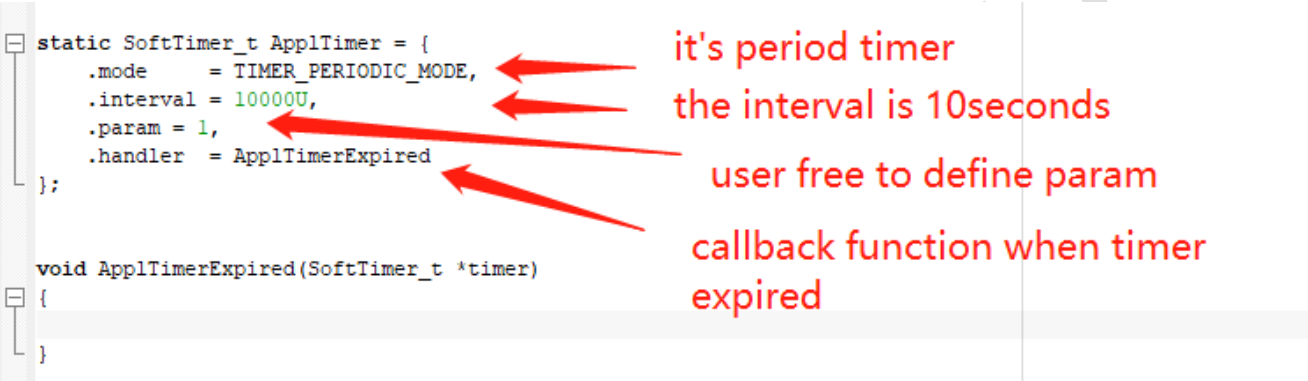
task init

3.2 SOFT TIMER

For the convenience of timing function, a serial software timer which is based the hardware global timer can be defined and used in the application.

3.2.1 Define and Run a Software Timer

Before using a software timer, a definition needs to be executed as following:

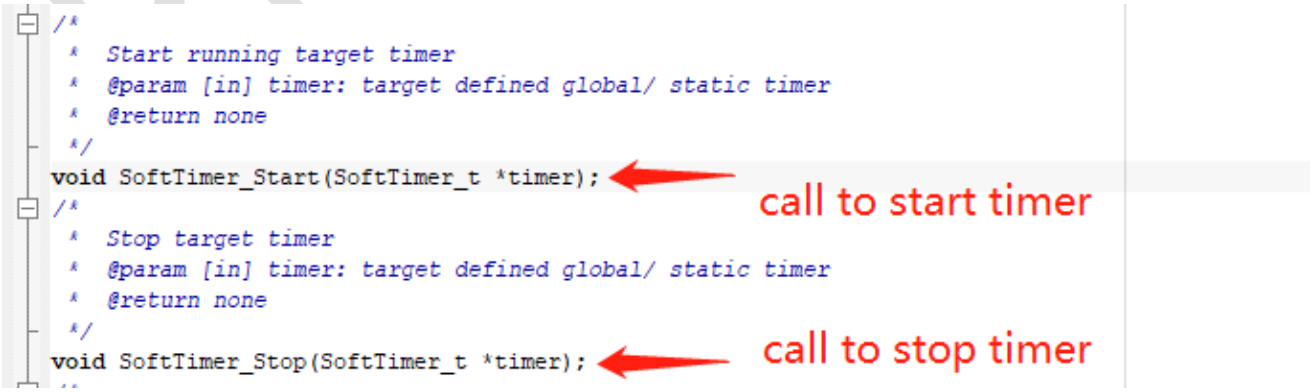


Note that don't change the other parameters which are not mentioned above.

The Software timer mode can be set to two modes: one-shot mode, and period mode.

```
/*- Types ------------------------------------------------------------------*/
typedef enum{
    TIMER_ONE_SHOT_MODE, /*this a one time timer, when timming expired, timer would be killed automatically*/
    TIMER_PERIODIC_MODE, /*this a period timer, timer would keep running unitl execute SoftTimer_Stop(), or change the mode by manual.*/
}SoftTimerMode_t;
```

1) For the one-shot mode, the time expired function would be executed only once and it will stop when the timer expired.
2) For the period mode, the expired function would be executed periodically according to the interval setting.
3) The defined software timer can be started and stopped by the following functions:



3.2.2 Define a Serial Software Timers

In some applications, a serial software timer needs to be defined and uses the same callback function. In this time, the “param” variable parameter is useful to help handle this use case, there is an example as follows:

```
static SoftTimer_t colorTransitionTimer[4] = {
  [0]={
    .mode     = TIMER_PERIODIC_MODE,
    .interval = COLOR_TRANSITION_INTERVAL,
    .handler  = colorTransitionTimerExpired,
    .param    = 0,
  },
  [1]={
    .mode     = TIMER_PERIODIC_MODE,
    .interval = COLOR_TRANSITION_INTERVAL,
    .handler  = colorTransitionTimerExpired,
    .param    = 1,
  },
  [2]={
    .mode     = TIMER_PERIODIC_MODE,
    .interval = COLOR_TRANSITION_INTERVAL,
    .handler  = colorTransitionTimerExpired,
    .param    = 2,
  },
  [3]={
    .mode     = TIMER_PERIODIC_MODE,
    .interval = COLOR_TRANSITION_INTERVAL,
    .handler  = colorTransitionTimerExpired,
    .param    = 3,
  },
};
```

define 4 timer with different param setting value

To read out the callback function timer param to distinguish who is the timer source:

```
void colorTransitionTimerExpired(struct SoftTimer *timer)
{
    if (timer->param == 0){

    }else if (timer->param == 1){

    }else if (timer->param == 2){

    }else if (timer->param == 3){

    }
}
```

3.2.3 Notes of Software Timer

1) Software timer definition must be global variable or static variable. Global and static variable is recommended.

   a) Wrong definition:

```
void ApplTimerExpired(SoftTimer_t *timer)
{

}

void APPL_TaskHandler(void)
{
    SoftTimer_t ApplTimer = {
        .mode     = TIMER_PERIODIC_MODE,
        .interval = 10000U,
        .param = 1,
        .handler  = ApplTimerExpired
    };

    switch(applState){
    case TASK_STATE_INIT:
      SoftTimer_Start(&ApplTimer);
      l_ifc_wake_up_bus();
      applState = TASK_STATE_ACTIVE;
      break;
    case TASK_STATE_ACTIVE:
      break;
    default:
      break;
    }
}
```

wrong definition

   b) Right definition:

```
void ApplTimerExpired(SoftTimer_t *timer);


static SoftTimer_t ApplTimer = {
    .mode     = TIMER_PERIODIC_MODE,
    .interval = 10000U,
    .param = 1,
    .handler  = ApplTimerExpired
};


void ApplTimerExpired(SoftTimer_t *timer)
{

}
```

Right definition

2) Timer Interval limitation:

   The hardware timer interval is about 8ms, so the shortest time interval of software timer is 8ms and the time resolution is 8ms correspondingly. Using hardware timer if more accurate time interval is needed.

3) Timer priority:

   a) Software timer has the highest priority, so don't do complex work in the timer expired callback function. Suggest using TM_PostTask to trigger task to handle the complex work.

   b) For the accuracy of software timer, make sure that every Task handle time is less than the minimum time interval(8ms) of hardware timer interval.

3.3 COLOR MIXING TASK

Color mixing task is used to set color mixing configuration and do color controlling, the color parameters should be initialized when calling color mixing task after system initialization.

3.3.1 Led Color Parameters Settings

1) The following parameters of Color Led need to be configured before use:
    a) Red, Green, Blue coordinate (x, y)
    b) Typical Red, Green, Blue intensity

   The following function would be used:

```
/*
 * Set Led physical param : xy coordinate, intensity in Lumin (for example)
 *  @param [in] type: LED type
 *  @param [in] temperatureR,G,B: current temperature -40~100 in 0.1C
 *  @param [in] coordinate: Led coordinate for example:P(0.3333,0.3333) = P(21845,21845);
 *  @param [in] intensity: 0~65535 in mcd (for example)
 *  @return 0:OK. others :FAIL
 */
uint8_t CLM_SetLedPhyParams(LedNum_t ledIndex,ColorTemperature_t temperature,LedcolorParam_t *param);

/*
 * Get Led physical param : x,y coordinate, intensity in Lumin (for example)
 *  @param [in] type: LED type
 *  @param [in] temperatureR,G,B: current temperature -40~100 in 0.1C
 *  @param [Out] coordinate: Led coordinate for example:P(0.3333,0.3333) = P(21845,21845);
 *  @param [Out] intensity: 0~65535 in Lumin (for example)
 *  @return 0:OK. others :FAIL
 */
uint8_t CLM_GetLedPhyParams(LedNum_t ledIndex,ColorTemperature_t temperature,LedcolorParam_t *param);
```

2) In order to make the stability of light intensity in different temperature and the light intensity between different LEDs, the minimum intensity of Red, Green and Blue need to configured:

```
/*
 *  In order to make the stability of light intensity  in different temperature, Red has maximum intensity degration
 *  @param [in] redIntensity:   0~65535 in Lumin (for example)
 *  @param [in] greenIntensity: 0~65535 in Lumin (for example)
 *  @param [in] blueIntensity:  0~65535 in Lumin (for example)
 *  @return 0
 */
int8_t CLM_SetMinimumIntensity(uint16_t redIntensity,uint16_t greenIntensity,uint16_t blueIntensity );

/*
 *  In order to make the stability of light intensity  in different temperature
 *  @param [out] redIntensity:   0~65535 in Lumin (for example)
 *  @param [out] greenIntensity: 0~65535 in Lumin (for example)
 *  @param [out] blueIntensity:  0~65535 in Lumin (for example)
 *  @return 0
 */
int8_t CLM_GetMinimumIntensity(uint16_t *redIntensity,uint16_t *greenIntensity,uint16_t *blueIntensity);
```

3) Set the white balance point to unify the color display:

```
/*
 * Set calibration point physical param : x,y coordinate
 *  @param [in] coordinate: Led whitePoint coordinate for example:P(0.3333, 0.3333) = P(21845, 21845);
 *  @return 0
 */
int8_t CLM_SetWhitePointParams(Coordinate_t const *coordinate);

/*
 * Get calibration point physical param : x,y coordinate
 *  @param [in] coordinate: Led coordinate for example:P(0.3333, 0.3333) = P(21845, 21845);
 *  @return coordinate
 */
uint8_t CLM_GetWhitePointParams(Coordinate_t *coordinate);
```

4) Set the initial Led PN voltage for Led temperature compensation:

   In general, this configuration is often done in room temperature, the parameter “temperature” should be close to 25’C.

```
/*
 * Set Led PN volt @specifical temperature in mV
 *  @param [in] temperatureR,G,B: current temperature -40-100 in 0.1C
 *  @param [in] volt_R: Red Led PN volt (mV);
 *  @param [in] volt_G: Green Led PN volt (mV);
 *  @param [in] volt_B: Blue Led PN volt (mV);
 *  @return 0:OK, others :FAIL
 */
uint8_t CLM_SetLedPNVolts(LedNum_t ledIndex,ColorTemperature_t temperature, int16_t volt_R, int16_t volt_G, int16_t volt_B);

/*
 * Get Led PN volt @25C in mV
 *  @param [in] temperatureR,G,B: current temperature -40-100 in 0.1C
 *  @param [out] volt_R: Red Led PN volt (mV);
 *  @param [out] volt_G: Green Led PN volt (mV);
 *  @param [out] volt_B: Blue Led PN volt (mV);
 *  @return 0:OK, others :FAIL
 */
uint8_t CLM_GetLedPNVolts(LedNum_t ledIndex,ColorTemperature_t temperature, int16_t *volt_R, int16_t *volt_G, int16_t *volt_B);
```

5) After configuration, executing storage to save the parameters to flash:

```
/*
 * storage color settings to flash
 *  @return 0,OK,others:NG
 */
int8_t CLM_StoragColorParams(void);
```

3.3.2 Led Color Calibrating

In general, after the parameters’ configuration mentioned above, the color configuration should be finished, but in real application, some customers want more accurate color control. For these requirements, the color mixing task provides white balance point calibration API functions. There are two calibrating API functions and a color control API function to finish the calibration operation below:

1) Do white balance point calibrating:

```
/*
 *  whitePoint intensity calculating, the room temperature is MUST when doing white point calculating.
 *  @param [in] temperatureR,G,B: current temperature -40-100 in 0.1C
 *  @param [in] whitePoint: white point coordinatefor example:P(0.3333,0.3333) = P(21845,21845);
 *  @return 0:OK. others :FAIL
 */
uint8_t CLM_CalculatingWhitePoint(LedNum_t ledIndex,ColorTemperature_t temperature,Coordinate_t *whitePoint);
```

Measure the real white balance point coordinate with color instrument. Go to step2 or go to step 4 until the measured coordinate has reached to acceptable range, for example: the set white balance point is (0.3333,0.3333), and measured value with color instrument is very close to this value, then stop the calibration step and go to step 4.

2) Read out maximum typical intensity of red, green and blue:

```
/*
 *  get typical max red, green and blue intensity
 *  @param [in]  ledIndex : led index
 *  @param [out] redIntensity:   0-65535 in Lumin (for example)
 *  @param [out] greenIntensity: 0-65535 in Lumin (for example)
 *  @param [out] blueIntensity:  0-65535 in Lumin (for example)
 *  @return 0:OK. others :FAIL
 */
uint8_t CLM_GetTypicalMaxIntensity(LedNum_t ledIndex,uint16_t *redIntensity,uint16_t *greenIntensity,uint16_t *blueIntensity);
```

3) Adjust the maximum typical intensity of red, green value which were read out from the above API function, and set the adjusted intensities with API function:

```
/*
 *  Set typical max red, green and blue intensity
 *  @param [in]  ledIndex : led index
 *  @param [in] redIntensity:   0-65535 in Lumin (for example)
 *  @param [in] greenIntensity: 0-65535 in Lumin (for example)
 *  @param [in] blueIntensity:  0-65535 in Lumin (for example)
 *  @return 0:OK. others :FAIL
 */
uint8_t CLM_SetTypicalMaxIntensity(LedNum_t ledIndex,uint16_t redIntensity,uint16_t greenIntensity,uint16_t blueIntensity);
```

Go to step 1.

4) Execute storage to save data to flash:

```
/*
 * storage color settings to flash
 *  @return 0,OK,others:NG
 */
int8_t CLM_StoragColorParams(void);
```

3.3.3 System and LED Protection Settings

In some critical situation, system needs to downgrade the Led intensity to protect the system. Color mixing task provides API function to limit the maximum Led intensity to protect LED and microcontroller from being in danger:

```
/*
 *  Set max limited intensity level for all of color control commands excepts CLM_SetPWMs command, this function often is used to limit the led brightness when Led is hot for led over hot protection.
 *  @param [in] intensity: 0-1024 ,means 0% -100%
 *  @return 0
 */
int8_t CLM_SetLimitedMaxIntensity(uint16_t intensity);

/*
 *  Get max limited intensity level see "CLM_SetLimitedMaxIntensity" function for the specific function introduction .
 *  @param [out] intensity: 0-1024 ,means 0% -100%
 *  @return 0
 */
int8_t CLM_GetLimitedMaxIntensity(uint16_t *intensity);
```

3.3.4 LED Color Control

3.3.4.1 CLM_SetFadingMode

Color mixing task provides two dimming modes when Led color and intensity are changed from one setting to the other one:

1) Proportion Mode: the intensity would change in liner mode; this mode is better for dynamic scene and color fast change application.
2) Derivative Mode: the intensity would change in nonlinear mode; the intensity changes is softer; this mode is better for slow color and intensity change application.

The following function is used for mode switch:

```
/*
 * set Fading mode
 * @param [in] FADING_MODE_DERIVATIVE: derivative mode more accurate at low brightness, FADING_MODE_PROPORATION: proporation in whole range
 * @return none
 */
void CLM_SetFadingMode(FadingMode_t mode);
```

The default mode is Derivative Mode if no operation.

3.3.4.2 CLM_SetXYY

Use absolute color coordinate to control color:

```
/*
 *  Set xyY output
 *  @param [in] temperatureR,G,B: current temperature -40-100 in 0.1C
 *  @param [in] x: 0-65535
 *  @param [in] y: 0-65535
 *  @param [in] Y: 0-1023
 *  @param [in] transitionTime: 0-65535ms
 *  @return 0:OK. others :FAIL
 */
uint8_t CLM_SetXYY(LedNum_t ledIndex,ColorTemperature_t temperature, uint16_t x, uint16_t y, uint16_t Y, uint16_t transitionTime);
```

3.3.4.3 CLM_SetLUV

Use u'v'L color coordinate to control color:

```
/*
 *  Set Lu'v' output
 *  @param [in] temperatureR,G,B: current temperature -40-100 in 0.1C
 *  @param [in] u': 0-65535
 *  @param [in] v': 0-65535
 *  @param [in] level: 0-1023
 *  @param [in] transitionTime: 0-65535ms
 *  @return 0:OK. others :FAIL
 */
uint8_t CLM_SetLUV(LedNum_t ledIndex,ColorTemperature_t temperature, uint16_t u, uint16_t v, uint16_t level, uint16_t transitionTime);
```

3.3.4.4 CLM_SetRGBL

```
 *  Set RGBL output
 *  @param [in] temperatureR,G,B: current temperature -40-100 in 0.1C
 *  @param [in] red: 0-255
 *  @param [in] green: 0-255
 *  @param [in] blue: 0-255
 *  @param [in] level: 0-1023
 *  @param [in] transitionTime: 0-65535ms
 *  @return 0:OK. others :FAIL
 */
uint8_t CLM_SetRGBL(LedNum_t ledIndex,ColorTemperature_t temperature, uint8_t red, uint8_t green, uint8_t blue,uint16_t level, uint16_t transitionTime);
```

3.3.4.5 CLM_SetHSL

```
/*
 *  Set hue saturation level output
 *  @param [in] temperatureR,G,B: current temperature -40-100 in 0.1C
 *  @param [in] hue: 0-65535
 *  @param [in] saturation: 0-255
 *  @param [in] level: 0-1023
 *  @param [in] transitionTime: 0-65535ms
 *  @return 0:OK. others :FAIL
 */
uint8_t CLM_SetHSL(LedNum_t ledIndex,ColorTemperature_t temperature, uint16_t hue, uint8_t saturation, uint16_t level, uint16_t transitionTime);
```

3.3.4.6 CLM_SetRGB

```
 *  Set RGB output
 *  @param [in] temperatureR,G,B: current temperature -40-100 in 0.1C
 *  @param [in] red: 0-255
 *  @param [in] green: 0-255
 *  @param [in] blue: 0-255
 *  @param [in] transitionTime: 0-65535ms
 *  @return 0:OK. others :FAIL
 */
uint8_t CLM_SetRGB(LedNum_t ledIndex,ColorTemperature_t temperature, uint8_t red, uint8_t green, uint8_t blue, uint16_t transitionTime);
```

3.3.4.7 CLM_SetPWMs

```
 * Directly Set PWMs output , only for test purpose,don't use for color mixing parameters
 *  @param [in] PWM_R: 0-65535 Red   Channel PWM value
 *  @param [in] PWM_G: 0-65535 Green Channel PWM value
 *  @param [in] PWM_B: 0-65535 Blue  Channel PWM value
 *  @param [in] transitionTime: 0-65535ms
 *  @return 0:OK. others :FAIL
 */
uint8_t CLM_SetPWMs(LedNum_t ledIndex,uint16_t pwmR, uint16_t pwmG, uint16_t pwmB, uint16_t transitionTime);
```

3.3.5 Accurate Color Control

Generally, each color with different intensity would have some color deviation. In the extremely accurate color control application, each color with different intensity needs to be calibrated. Color mixing task provides some color control APIs with start with “Accurate” characters. The iRatio parameters can be used to adjust the color ratio of Red, Green and Blue to adapt the real requirements slightly. For example, saving calibrated iRatio value of specific color into flash. When executing this color control, reading out the iRatio from flash to do the accurate color control.

There is an example for ratio adjustment:

During color calibrating, color instrument measured the target color had more red percentage than expected, then you can adjust iRatio.red value to high to decrease the red percentage to achieve accurate color calibration.

3.3.5.1 CLM_SetAccurateXYY

```
/*
 *  Set accurate color xyY output with calibration parameters
 *  @param [in] iRatio:0-255 for 100%-125% of max Intensity, for accurate calibrated color Intensity compensation, set all 0 for normal color control
 *  @param [in] temperatureR,G,B: current temperature -40-100 in 0.1C
 *  @param [in] x: 0-65535
 *  @param [in] y: 0-65535
 *  @param [in] Y: 0-100
 *  @param [in] transitionTime: 0-65535ms
 *  @return 0:OK. others :FAIL
 */
uint8_t CLM_SetAccurateXYY(LedNum_t ledIndex,IntensityRatio_t iRatio, ColorTemperature_t temperature, uint16_t x, uint16_t y, uint8_t Y, uint16_t transitionTime);
```

3.3.5.2 CLM_SetAccurateLUV

```
/*
 *  Set accurate color L'u'v output with calibration parameters
 *  @param [in] iRatio:0-255 for 100%-125% of max Intensity, for accurate calibrated color Intensity compensation, set all 0 for normal color control
 *  @param [in] temperatureR,G,B: current temperature -40-100 in 0.1C
 *  @param [in] u': 0-65535
 *  @param [in] v': 0-65535
 *  @param [in] level: 0-100
 *  @param [in] transitionTime: 0-65535ms
 *  @return 0:OK. others :FAIL
 */
uint8_t CLM_SetAccurateLUV(LedNum_t ledIndex,IntensityRatio_t iRatio, ColorTemperature_t temperature, uint16_t u, uint16_t v, uint8_t level, uint16_t transitionTime);
```

3.3.5.3 CLM_SetAccurateRGBL

```
 *  Set accurate color RGBL output with calibration parameters
 *  @param [in] iRatio:0-255 for 100%-125% of max Intensity, for accurate calibrated color Intensity compensation, set all 0 for normal color control
 *  @param [in] temperatureR,G,B: current temperature -40-100 in 0.1C
 *  @param [in] red: 0-255
 *  @param [in] green: 0-255
 *  @param [in] blue: 0-255
 *  @param [in] level: 0-100
 *  @param [in] transitionTime: 0-65535ms
 *  @return 0:OK. others :FAIL
 */
uint8_t CLM_SetAccurateRGBL(LedNum_t ledIndex,IntensityRatio_t iRatio, ColorTemperature_t temperature, uint8_t red, uint8_t green, uint8_t blue,uint8_t level, uint16_t transitionTime);
```

3.4 LIN SLAVE TASK

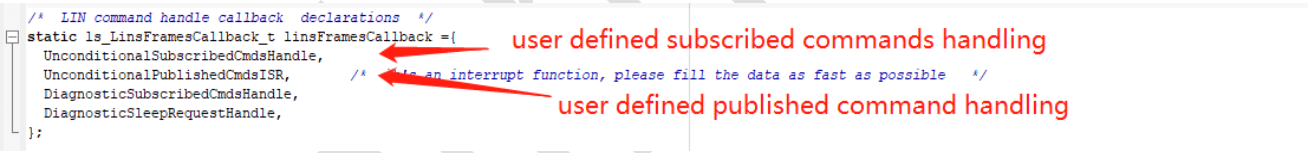
Task manager assigns a dedicated task “LINS_TaskHandler” for LIN slave commands handling:


```
void LINS_TaskHandler(void)
{
    switch(linsTaskState){
    case TASK_STATE_ACTIVE:
      break;
    case TASK_STATE_INIT:
      (void)l_sys_init();
      ls_register_services(UnconditionalCmdsTable, (uint8_t)(sizeof(UnconditionalCmdsTable)/sizeof(LIN_Device_Frame_t)), DIAG_DATA_BUFF_SIZE,&linsFramesCallback);
      (void)ls_set_tp_timeout(N_AS, N_CR);
      linsTaskState = TASK_STATE_ACTIVE;
      break;
    default:
      break;
    }
}
```

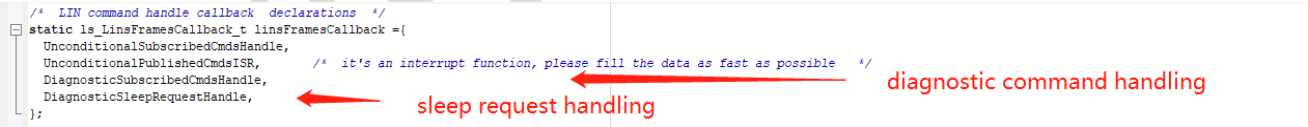
Callback functions and transport layer parameters settings:


```
void LINS_TaskHandler(void)
{
    switch(linsTaskState){
    case TASK_STATE_ACTIVE:
      break;
    case TASK_STATE_INIT:
      (void)l_sys_init();
      ls_register_services(UnconditionalCmdsTable, (uint8_t)(sizeof(UnconditionalCmdsTable)/sizeof(LIN_Device_Frame_t)), DIAG_DATA_BUFF_SIZE,&linsFramesCallback);
      (void)ls_set_tp_timeout(N_AS, N_CR);
      linsTaskState = TASK_STATE_ACTIVE;
      break;
    default:
      break;
    }
}
```

There are four callback functions for LIN slave commands handling:




```
/*  LIN command handle callback  declarations  */
static ls_LinsFramesCallback_t linsFramesCallback ={
  UnconditionalSubscribedCmdsHandle,
  UnconditionalPublishedCmdsISR,      /*  it's an interrupt function, please fill the data as fast as possible   */
  DiagnosticSubscribedCmdsHandle,
  DiagnosticSleepRequestHandle,
};
```




```
/*  LIN command handle callback  declarations  */
static ls_LinsFramesCallback_t linsFramesCallback ={
  UnconditionalSubscribedCmdsHandle,
  UnconditionalPublishedCmdsISR,      /*  it's an interrupt function, please fill the data as fast as possible   */
  DiagnosticSubscribedCmdsHandle,
  DiagnosticSleepRequestHandle,
};
```

1) void UnconditionalSubscribedCmdsHandle(LIN_Device_Frame_t const *frame):
   this callback function is used to handling the command and data from LIN master. for ambient light controlling system, it’s often a color control command from LIN master.
2) void UnconditionalPublishedCmdsISR(LIN_Device_Frame_t *const frame):
   this callback function is used to handling the command head from LIN master and send back the data to LIN master. It’s often used to send back LIN status to LIN master. Note that it’s interrupt function, don’t do complex work in this function.
3) void DiagnosticSubscribedCmdsHandle(const DiagReqInfo_t * const frameInfo):
   LIN Diagnostic commands handling function includes LIN node configuration and identification commands handling which are defined in LIN spec, self-defined the multi PDUs and single PDU examples code. Refers to LIN specification for more detailed information.

4) void DiagnosticSleepRequestHandle(SleepRequestType_t type):
   there are two kinds of sleep request types:

```
typedef enum{
    SLEEP_REQ_TYPE_MASTER_REQ = 0,
    SLEEP_REQ_TYPE_BUS_IDLE_TIMEROUT,
}SleepRequestType_t;
```

master send command request to sleep

LIN bus idle timeout to sleep

3.4.1 LIN Command Register Table:

All the commands to be handled are defined in the command Register Table:

```
/* **FID table declarations****/
static LIN_Device_Frame_t UnconditionalCmdsTable[DEFAULT_LINS_FID_SIZE] = {
  [FID_COLOR_CTRL_INDEX] = {
      .frame_id = FID_COLOR_CTRL,
      .msg_type = LIN_MSG_TYPE_RX,
      .checksum = LIN_CHECKSUM_ENHANCED,
      .length   = LIN_BUFF_SIZE,
      .frameIsValid = TRUE,
  },
  [FID_STATUS_FRAME_INDEX] = {
      .frame_id = FID_STATUS_FRAME,   /* status management frame */
      .msg_type = LIN_MSG_TYPE_TX,
      .checksum = LIN_CHECKSUM_ENHANCED,
      .length   = LIN_BUFF_SIZE,
      .frameIsValid = TRUE,
  },
  [USER_DATA_REPORT_INDEX] = {
      .frame_id = USER_DATA_REPORT,   /* user data report */
      .msg_type = LIN_MSG_TYPE_TX,
      .checksum = LIN_CHECKSUM_ENHANCED,
      .length   = LIN_BUFF_SIZE,
      .frameIsValid = TRUE,
      .linkedEventTriggerFidTableIndex = EVENT_TRIGGERED_INDEX,
  },
  [EVENT_TRIGGERED_INDEX] = {
      .frame_id = EVENT_TRIGGER_DATA_REPORT,   /* event trigger data report */
      .msg_type = LIN_MSG_TYPE_TX_EVENT,       /*event trigger data request from sender for slave; */
      .checksum = LIN_CHECKSUM_ENHANCED,
      .length   = LIN_BUFF_SIZE,
      .frameIsValid = TRUE,
      .eventTriggered = FALSE,
      .linkedEventTriggerFidTableIndex = USER_DATA_REPORT_INDEX,
  },
};
```

Then followings parameters should be configured when defining a new frame command:

1) frame_id: the frame to be handled, the parity is not included.
2) msg_type: message includes three types:
   a) LIN_MSG_TYPE_RX: Received data frame from master, it's a subscriber.
   b) LIN_MSG_TYPE_TX: Received frame head from master and response data frame to master, it's a publisher.
   c) LIN_MSG_TYPE_TX_EVENT: Received frame head from master and response data frame to master when a specific defined event has been triggered, it's a publisher.

3) checksum: there are two types of checksum which is defined in LIN specification:

```
typedef enum{
    LIN_CHECKSUM_CLASSIC = 0U,  /*!< classic checksum does not include ID Byte.  */
    LIN_CHECKSUM_ENHANCED       /*!< "enhanced" checksum includes ID Byte.  */
}ChecksumType_t;
```

4) length: data buffer length.

5) frameIsValid: this parameter should be set to TRUE when the frame is defined and to be used, the command from LIN master would be ignored if this parameter was set to FALSE.

6) linkedEventTriggerFidTableIndex: this parameter should be configured when this frame msg_type was set to LIN_MSG_TYPE_TX_EVENT.

   the linkedEventTriggerFidTableIndex parameter should be assigned a frame ID which was defined in the table and was used to solve the event collision issue and the parameter: linkedEventTriggerFidTableIndex in the frame ID should be assigned to the event trigger frame ID, for example:

```
/* **FID table declarations****/
static LIN_Device_Frame_t UnconditionalCmdsTable[DEFAULT_LINS_FID_SIZE] = {
  [FID_COLOR_CTRL_INDEX] = {
      .frame_id = FID_COLOR_CTRL,
      .msg_type = LIN_MSG_TYPE_RX,
      .checksum = LIN_CHECKSUM_ENHANCED,
      .length   = LIN_BUFF_SIZE,
      .frameIsValid = TRUE,
  },
  [FID_STATUS_FRAME_INDEX] = {
      .frame_id = FID_STATUS_FRAME,   /* status management frame */
      .msg_type = LIN_MSG_TYPE_TX,
      .checksum = LIN_CHECKSUM_ENHANCED,
      .length   = LIN_BUFF_SIZE,
      .frameIsValid = TRUE,
  },
  [USER_DATA_REPORT_INDEX] = {
      .frame_id = USER_DATA_REPORT,   /* user data report */
      .msg_type = LIN_MSG_TYPE_TX,
      .checksum = LIN_CHECKSUM_ENHANCED,
      .length   = LIN_BUFF_SIZE,
      .frameIsValid = TRUE,
      .linkedEventTriggerFidTableIndex = EVENT_TRIGGERED_INDEX,
  },
  [EVENT_TRIGGERED_INDEX] = {
      .frame_id = EVENT_TRIGGER_DATA_REPORT,   /* event trigger data report */
      .msg_type = LIN_MSG_TYPE_TX_EVENT,       /*event trigger data request from sender for slave; */
      .checksum = LIN_CHECKSUM_ENHANCED,
      .length   = LIN_BUFF_SIZE,
      .frameIsValid = TRUE,
      .eventTriggered = FALSE,
      .linkedEventTriggerFidTableIndex = USER_DATA_REPORT_INDEX,
  },
};
```


Refers to LIN specification 2.2A page 34, 2.3.3.2 Event triggered frame for more information.