

开发指南

编制人	TerryLi	审核人	AndyGao	批准人	
产品名称		产品编号		文档编号	
会签日期			版本	0.5	

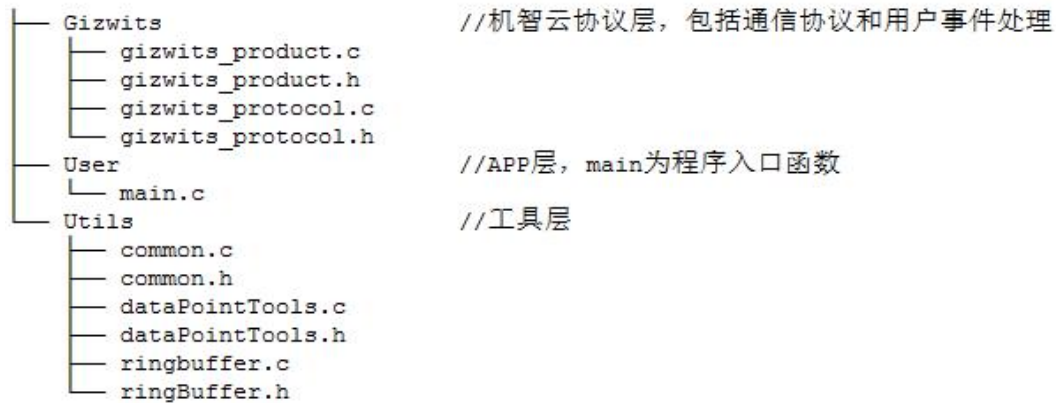
修改记录:

修改时间	修改记录	修改人	版本	备注
20160909	初建	TerryLi	V0.1	
20161030	增加模组产测、绑定和获取 NTP 时间接口	TerryLi	V0.2	
20161228	增加透传通道支持	TerryLi	V0.3	
20170915	修改文件目录	TerryLi	V0.4	
20180521	文档排版	AndyGao	V0.5	

目录:

1 文件介绍.....	4
2 API 介绍.....	4
void gizwitsInit(void).....	4
void gizwitsSetMode(uint8_t mode).....	4
void gizwitsGetNTP(void).....	5
void gizwitsHandle(dataPoint_t *dataPoint).....	5
int8_t gizwitsEventProcess(eventInfo_t *info, uint8_t *data, uint32_t len).....	5
int32_t gizwitsPassthroughData(uint8_t *data, uint32_t len).....	5
3 移植.....	5
实现串口 A 驱动.....	6
实现定时器.....	7
实现芯片复位函数.....	8
实现串口打印驱动.....	8
实现配置入网.....	9
实现下行动作执行.....	10
实现上行数据采集.....	10
实现模组状态处理.....	11

1 文件介绍



重要文件解读:

1. gizwits_product.c
该文件为产品相关处理函数, 如 gizwitsEventProcess()。
2. gizwits_product.h
该文件为 gizwits_product.c 的头文件, 如 HARDWARE_VERSION、SOFTWARE_VERSION。
3. gizwits_protocol.c
该文件为 SDK API 接口函数定义文件。
4. gizwits_protocol.h
该文件为 gizwits_protocol.c 对应头文件, 相关 API 的接口声明均在此文件中。
5. 其他文件
 - a) User/main.c
MCU 程序入口函数所在文件, 入口函数为 main(void)。

2 API 介绍

void gizwitsInit(void)

gizwits 协议初始化接口。

用户调用该接口可以完成 Gizwits 协议相关初始化 (包括协议相关定时器、串口的初始化)。

void gizwitsSetMode(uint8_t mode)

参数 mode[in]: WIFI_MODE_TYPE_T 枚举值

- 参数为 WIFI_RESET_MODE, 恢复模组出厂配置接口, 调用会清空所有配置参数, 恢复到出厂默认配置。
- 参数为 WIFI_SOFTAP_MODE 或 WIFI_AIRLINK_MODE, 配置模式切换接口, 支持 SoftAP 和 AirLink 模式。参数为 WIFI_SOFTAP_MODE 时配置模组进入 SoftAp 模式, 参数为 WIFI_AIRLINK_MODE 配置模组进入 AirLink 模式。
- 参数为 WIFI_PRODUCTION_TEST, 模组进入产测模式。

- 参数为 WIFI_NINABLE_MODE，模组进入可绑定模式，可绑定时间为 NINABLETIME(gizwits_protocol.h 中声明)，默认为 0，表示模组永久可绑定。

void gizwitsGetNTP(void)

获取 NTP 时间接口。

用户调用该接口可以获取当前网络时间，MCU 发起请求，模组回复后将产生 WIFI_NTP 事件，用户可在 gizwitsEventProcess 函数中进行相应处理。

void gizwitsHandle(dataPoint_t *dataPoint)

参数 dataPoint[in]:用户设备数据点。

该函数中完成了相应协议数据的处理及数据上报的等相关操作。

int8_t gizwitsEventProcess(eventInfo_t *info, uint8_t *data, uint32_t len)

参数 info[in]:事件队列

参数 data[in]:数据

参数 len [in]:数据长度

用户数据处理函数,包括 wifi 状态更新事件和控制事件。

a) Wifi 状态更新事件

WIFI_开头的事件为 wifi 状态更新事件，data 参数仅在 WIFI_RSSI 有效，data 值为 RSSI 值,数据类型为 uint8_t，取值范围 0~7。

b) 控制事件

与数据点相关,本版本代码会打印相关事件信息，相关数值也一并打印输出，用户只需要做命令的具体执行即可。

int32_t gizwitsPassthroughData(uint8_t *data, uint32_t len)

参数 data[in]:数据

参数 len [in]:数据长度

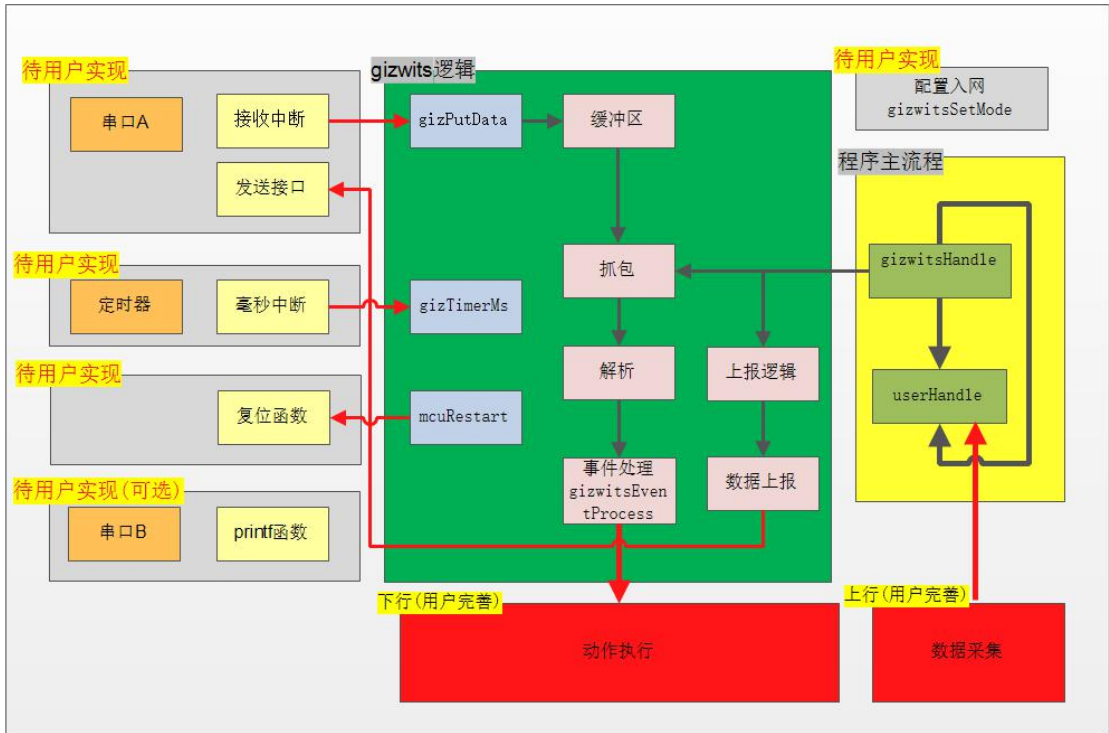
用户调用该接口可以完成私有协议数据的上报。

3 移植

MCU 通用平台版代码对硬件平台的要求：

- 平台支持两个串口接口（至少一个），一个负责与 wifi 模组间的数据收发（必须），一个用于调试信息打印（可复用数据收发串口）。
 - 平台支持定时器功能（1ms 精确定时）。
 - 平台支持至少 2K 的 RAM 空间（太少会导致数据协议的处理异常）。
- 自动化代码生成工具已经根据用户定义的产品数据点信息，生成了对应的机智云串口协

议层代码，用户需要移植代码到自己的工程中，完成设备的接入工作。程序结构框图如下：



gizwits 逻辑和程序主流程已经帮用户实现，图中用黄色小标注明的部分待用户实现并完成代码的移植。用户的移植工作主要分以下几个方面进行。

实现串口 A 驱动

MCU 方案需要用户实现一个串口，用于设备 MCU 与 WIFI 模组之间数据通信。用户首先要实现串口接收中断服务函数接口 UART_IRQ_FUN()，该接口调用 gizPutData()函数实现串口数据的接收并且写入协议层数据缓冲区。另外，用户需要实现串口的发送接口，uartWrite()函数调用该接口实现设备数据的发送。需要特别注意的是 gizwits_product.c 文件中 uartWrite()函数是伪函数，用户需根据自己实现的串口发送接口完善 uartWrite()，请关注相关注释信息，以防出错。

下面以 STM32F103C8T6 平台为例，本例使用 USART2 与模组通信，串口初始化不在此罗列，中断服务函数和串口发送报文函数实现如下：

```
/**
 * @brief USART2 串口中断服务函数
 *
 * 接收功能，用于接收与 WiFi 模组间的串口协议数据
 * @param none
 * @return none
 */
void UART_IRQ_FUN(void)
{
    uint8_t value = 0;
    if(USART_GetITStatus(USART2, USART_IT_RXNE) != RESET)
    {
```

```

        USART_ClearITPendingBit(USART2,USART_IT_RXNE);
        value = USART_ReceiveData(USART2);

        gizPutData(&value, 1);
    }
}

/**
 * @brief 串口写操作，发送数据报文(报文数据中遇 0xFF 需要用 0x55 转义)到 WiFi 模组
 *
 * @param buf      : 数据地址
 * @param len      : 数据长度
 *
 * @return : 正确返回有效数据长度;-1，错误返回
 */
int32_t uartWrite(uint8_t *buf, uint32_t len)
{
    uint32_t i = 0;

    if(NULL == buf)
    {
        return -1;
    }
    for(i=0; i<len; i++)
    {
        USART_SendData(UART, buf[i]);
        while (USART_GetFlagStatus(UART, USART_FLAG_TXE) == RESET);
        if(i >= 2 && buf[i] == 0xFF)
        {
            USART_SendData(UART, 0x55);
            while (USART_GetFlagStatus(UART, USART_FLAG_TXE) == RESET);
        }
    }
    return len;
}

```

实现定时器

协议层使用到了一个系统时间，该事件单位为毫秒，所以要求用户实现一个毫秒定时器，并且实现中断服务函数 `TIMER_IRQ_FUN()`，该函数调用 `gizTimerMs()` 实现协议层系统时间的维护。

下面以 STM32F103C8T6 平台为例，本例使用 TIM3 实现时间维护，定时器初始化不在此罗列，中断服务函数实现如下：

```
/**
```

```

* @brief 定时器 TIM3 中断处理函数

* @param none
* @return none
*/
void TIMER_IRQ_FUN(void)
{
    if (TIM_GetITStatus(TIMER, TIM_IT_Update) != RESET)
    {
        TIM_ClearITPendingBit(TIMER, TIM_IT_Update );
        gizTimerMs();
    }
}

```

实现芯片复位函数

根据串口协议文档规定，模组可以发送命令复位设备 MCU，所以用户需要实现 mcuRestart()接口完成设备的复位。

下面以 STM32F103C8T6 平台为例，本例使用 TIM3 实现时间维护，定时器初始化不在此罗列，中断服务函数实现如下：

```

/**
* @brief MCU 复位函数

* @param none
* @return none
*/
void mcuRestart(void)
{
    __set_FAULTMASK(1);
    NVIC_SystemReset();
}

```

实现串口打印驱动

如果用户需要打印日志调试信息，要求用户实现 printf 函数。协议层将用 GIZWITS_LOG 宏替代 printf，进行相关信息的打印。如果用户不使用日志调试，那么需要将协议层相关日志打印部分的代码屏蔽掉方可运行。如果用户不使用日志调试，遇到问题请咨询机智云工程师。

下面以 STM32F103C8T6 平台为例，本例使用 USART1 实现串口打印，串口初始化不在此罗列，printf 重定向实现如下：

```

#ifdef __GNUC__
#define PUTCHAR_PROTOTYPE int __io_putchar(int ch)
#else

```



```

#define PUTCHAR_PROTOTYPE int fputc(int ch, FILE *f)
#endif

/**
 * @brief printf 打印重定向
 * @param none
 * @return none
 */
PUTCHAR_PROTOTYPE
{
    USART_SendData(USART1,(u8)ch);
    while (USART_GetFlagStatus(USART1, USART_FLAG_TXE) == RESET);
    return ch;
}

```

实现配置入网

模组支持 SoftAp 和 AirLink 两种方式配置入网，相应接口为 gizwitsSetMode()，建议采用按键的方式，相应的按键动作触发执行具体的模式设置。

另外，可以通过 gizwitsSetMode()接口复位模组，恢复默认出厂设置。

下面以 STM32F103C8T6 平台为例，本例使用按键方式实现配置入网和控制模组复位功能，按键初始化不在此罗列，按键触发操作实现如下：

```

/**
 * key1 按键长按处理
 * @param none
 * @return none
 */
void key1LongPress(void)
{
    printf("KEY1 PRESS LONG ,Wifi Reset\n");
    gizwitsSetMode(WIFI_RESET_MODE);
}

/**
 * key2 按键短按处理
 * @param none
 * @return none
 */
void key2ShortPress(void)
{
    printf("KEY2 PRESS ,Soft AP mode\n");
    gizwitsSetMode(WIFI_SOFTAP_MODE);
}

```

```

/**
 * key2 按键长按处理
 * @param none
 * @return none
 */
void key2LongPress(void)
{
    //AirLink mode
    printf("KEY2 PRESS LONG ,AirLink mode\n");
    gizwitsSetMode(WIFI_AIRLINK_MODE);
}

```

实现下行动作执行

数据点方式将转换成数据点事件，开发者只需要在 gizwits_product.c 文件的 gizwitsEventProcess()相应事件下作具体处理即可。

下面使用 STM32F103C8T6 平台，以微信宠物屋实现 APP 控制电机转动为例，motorStatus()函数是需要开发者自己实现，其他代码自动化工具会帮助开发者生成，gizwitsEventProcess()函数省略其他代码段，案例如下：

```

case EVENT_MOTOR_SPEED:
    currentDataPoint.valueMotor_Speed = dataPointPtr->valueMotor_Speed;
    GIZWITS_LOG("Evt:EVENT_MOTOR_SPEED %d\n",currentDataPoint.valueMotor_Speed);
    motorStatus(currentDataPoint.valueMotor_Speed);
break;

```

实现上行数据采集

该工程代码默认在 userHandle()中实现传感器数据采集，并且该函数在 while 循环执行，原则上用户只需要关心如何采集数据。特别提醒，默认 while 循环执行速度较快，需要针对不同的需求，用户可调整数据点数据的采集周期和接口实现位置，预防由于传感器数据采集过快引发的不必要的问题。

下面使用 STM32F103C8T6 平台，以微信宠物屋实现红外探测为例，irHandle()函数是需要开发者自己实现，其他代码自动化工具会帮助开发者生成，案例如下：

```

void userHandle(void)
{
    currentDataPoint.valueInfrared = irHandle();
}

```

实现模组状态处理

参考接口 `gizwitsEventProcess()`，本版软件已经将 wifi 状态数据转换成了 `event`，开发者仅关注相应事件即可。用户可以通过获取到的 WIFI 状态做相应的逻辑处理。

下面使用 STM32F103C8T6 平台，以微信宠物屋实现配置入网后接收到网络连接到路由器关闭 RGB 灯为例，`ledRgbControl()`函数是需要开发者自己实现，其他代码自动化工具会帮助开发者生成，`gizwitsEventProcess()`函数省略其他代码段，案例如下：

```
case WIFI_CON_M2M:  
    ledRgbControl(0,0,0);  
break;
```