

OneNET Studio设备接入SDK使用说明

版本号：1.1.0

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 日期 | 版本 | 作者 | 更新记录 | 备注 |
| 2020/06/24 | v1.0.0 | 吴优、陈龙宇 | 初版发布。 |  |
| 2020/07/22 | v1.0.1 | 吴优 | 添加数组上传的说明 |  |
| 2020/09/27 | v1.0.2 | 宋伟 | 新增读属性、服务调用接口 |  |
| 2020/10/22 | v1.1.0 | 宋伟 | 新增网关子设备接口 |  |

目　　录

[一．前言 1](#_Toc44229356)

[二．SDK文件说明 1](#_Toc44229357)

[三．SDK使用说明 1](#_Toc44229358)

[3.1 宏定义 1](#_Toc44229359)

[3.2 用户API接口 2](#_Toc44229360)

[3.2.1 设备初始化 2](#_Toc44229361)

[3.2.2 设备登录 2](#_Toc44229362)

[3.2.3 设备注销 2](#_Toc44229363)

[3.2.4 数据解析 3](#_Toc44229364)

[3.3 配置文件接口 3](#_Toc44229365)

[3.3.1 功能点数组 3](#_Toc44229366)

[3.3.2 数据下发 3](#_Toc44229367)

[3.3.3 数据上传 4](#_Toc44229368)

[3.4 系统适配接口 4](#_Toc44229369)

[3.4.1 内存接口 4](#_Toc44229370)

[3.4.1.1 内存分配 4](#_Toc44229371)

[3.4.1.2 内存释放 4](#_Toc44229372)

[3.4.1.3 内存拷贝 4](#_Toc44229373)

[3.4.1.4 内存初始化 5](#_Toc44229374)

[3.4.2 网络接口 5](#_Toc44229375)

[3.4.2.1 TCP接口 5](#_Toc44229376)

[3.4.2.1.1 创建网络连接 5](#_Toc44229377)

[3.4.2.1.2 数据发送 5](#_Toc44229378)

[3.4.2.1.3 数据接收 6](#_Toc44229379)

[3.4.2.1.4 断开网络连接 6](#_Toc44229380)

[3.4.2.2 UDP接口 6](#_Toc44229381)

[3.4.2.2.1 创建网络连接 6](#_Toc44229382)

[3.4.2.2.2 数据发送 6](#_Toc44229383)

[3.4.2.2.3 数据接收 7](#_Toc44229384)

[3.4.2.2.4 断开网络连接 7](#_Toc44229385)

[3.4.3 时间接口 7](#_Toc44229386)

[3.4.3.1 获取系统时间（毫秒） 7](#_Toc44229387)

[3.4.3.2 倒计时器启动 7](#_Toc44229388)

[3.4.3.3 倒计时器配置 8](#_Toc44229389)

[3.4.3.4 倒计时器剩余时间 8](#_Toc44229390)

[3.4.3.5 倒计时器超时判断 8](#_Toc44229391)

[3.4.3.6 倒计时器停止 8](#_Toc44229392)

[四．SDK移植说明 8](#_Toc44229393)

[4.1 移植流程说明 9](#_Toc44229394)

[4.2 系统接口说明 9](#_Toc44229395)

[4.3 配置文件说明 9](#_Toc44229396)

[4.3.1 数据下发 9](#_Toc44229397)

[4.3.2 数据上传 10](#_Toc44229398)

[4.4 移植注意事项 12](#_Toc44229399)

一．前言

本文用于说明OneNET Studio物模型设备接入SDK的使用，适用于“MCU+标准通信模组”或单板SOC的方案。SDK由固定代码包和平台生成的配置文件构成，用户根据使用的硬件平台集成SDK，通过配置宏选择不同的接入协议，并调用相应API接口即可实现OneNET Studio的快速接入。

二．SDK文件说明

表2-1 SDK文件说明

|  |  |  |  |
| --- | --- | --- | --- |
| 目录 | | | 说明 |
| config.h | | | 宏配置文件 |
| onenet | thing\_model | core | 用户API接口，物模型数据封装与解析，OneNET相关数据模型 |
| user | 用户配置文件，根据平台定义的物模型生成，包含数据上传、读写接口 |
| protocols | mqtt | | paho-mqtt库 |
| coap | | er-coap-13库 |
| security | wolfssl | | wolfssl加密库 |
| platforms | linux | | Linux系统适配接口，包括内存接口，网络接口，时间接口等 |

三．SDK使用说明

3.1 宏定义

用户需通过配置宏来决定使用的接入协议，心跳周期，数据缓冲区大小，以及是否加密等。

|  |  |
| --- | --- |
| 宏 | 含义 |
| FEATURE\_TM\_PROTOCOL\_MQTT | MQTT协议 |
| FEATURE\_TM\_MQTT\_TLS\_NONE | 非加密（MQTT） |
| FEATURE\_TM\_MQTT\_TLS\_WOLFSSL | TLS加密（MQTT） |
| FEATURE\_TM\_PROTOCOL\_COAP | CoAP协议 |
| FEATURE\_TM\_VERSION | 物模型版本（默认1.0） |
| FEATURE\_TM\_LIFE\_TIME | 心跳周期（秒） |
| FEATURE\_TM\_REPLY\_TIMEOUT | 设备响应超时时间（毫秒） |
| FEATURE\_TM\_SEND\_BUF\_LEN | 发送缓冲区长度 |
| FEATURE\_TM\_RECV\_BUF\_LEN | 接收缓冲区长度 |
| FEATURE\_TM\_PAYLOAD\_BUF\_LEN | 数据最大封包长度 |

3.2 设备通用接口

用户通过调用物模型相关API接口，可实现设备初始化，登录、注销平台，以及数据解析等功能，可用于直连设备和网关设备。

3.2.1 设备初始化

|  |  |
| --- | --- |
| 功能 | 设备初始化，需传入配置文件中生成的属性功能点数组及其长度 |
| 函数定义 | int32\_t tm\_init(struct tm\_downlink\_tbl\_t \*downlink\_tbl) |
| 参数 | downlink\_tbl：下行数据处理回调定义表，包含属性和服务表，详细定义参考tm\_api.h |
| 返回值 | 0：初始化成功；-1：初始化失败 |
| 说明 | 需在设备登录之前调用 |

3.2.2 设备登录

|  |  |
| --- | --- |
| 功能 | 发起设备登录请求，需传入产品与设备相关信息 |
| 函数定义 | int32\_t tm\_login(  const int8\_t \*product\_id,  const int8\_t \*dev\_name,  const int8\_t \*access\_key,  uint32\_t timeout\_ms  ) |
| 参数 | product\_id：产品id  dev\_name：设备名称  access\_key：产品key  timeout\_ms：超时时间（毫秒） |
| 返回值 | 0：登录成功；-1：登录失败 |

3.2.3 设备注销

|  |  |
| --- | --- |
| 功能 | 发起设备注销请求，释放内存，断开平台连接 |
| 函数定义 | int32\_t tm\_logout(uint32\_t timeout\_ms) |
| 参数 | timeout\_ms：超时时间（毫秒） |
| 返回值 | 0：注销成功 |

3.2.4 数据解析

|  |  |
| --- | --- |
| 功能 | 解析平台下发的数据 |
| 函数定义 | int32\_t tm\_step(uint32\_t timeout\_ms) |
| 参数 | timeout\_ms：超时时间（毫秒） |
| 返回值 | 0：解析正常；-1：解析错误 |
| 说明 | 需在设备登录成功之后反复调用，若数据解析错误（由网络异常等原因引起），则需注销设备，然后重新登录；超时时间推荐设置200ms |

3.2.5 获取期望属性

|  |  |
| --- | --- |
| 功能 | 获取平台为设备设置的期望属性值 |
| 函数定义 | int32\_t tm\_get\_desired\_props(uint32\_t timeout\_ms) |
| 参数 | timeout\_ms：超时时间（毫秒） |
| 返回值 | 0：获取成功 |
| 说明 | 调用接口后，SDK内部解析平台下发的期望属性值，将自动调用对应的写属性接口。 |

3.2.6 清除期望属性

|  |  |
| --- | --- |
| 功能 | 清除平台为设备设置的期望属性值 |
| 函数定义 | int32\_t tm\_delete\_desired\_props(uint32\_t timeout\_ms) |
| 参数 | timeout\_ms：超时时间（毫秒） |
| 返回值 | 0：清除成功 |
| 说明 | 默认清除平台侧配置的所有期望属性值。 |

3.2.6 打包设备数据

|  |  |
| --- | --- |
| 功能 | 打包设备的属性和事件数据，可用于子设备。 |
| 函数定义 | void\* tm\_pack\_device\_data(void \*data, const int8\_t \*product\_id, const int8\_t \*dev\_name, void \*prop, void \*event, int8\_t as\_raw) |
| 参数 | data：需要打包的目标指针地址，用于后续调用上报接口。设置为空时由接口内部分配空间，并通过返回值返回地址  product\_id：需要打包数据的产品id  dev\_name：需要打包数据的设备名称  prop：传入属性数据。支持json格式(as\_raw为1)，也可以仿照tm\_user文件的数据上传接口使用tm\_data接口构造数据（as\_raw为0）  event：定义同prop，用于传入事件数据 |
| 返回值 | 打包后的数据指针地址 |

3.2.6 上报批量数据

|  |  |
| --- | --- |
| 功能 | 向平台批量上报若干设备的属性、事件 |
| 函数定义 | int32\_t tm\_post\_pack\_data(void \*pack\_data, uint32\_t timeout\_ms) |
| 参数 | pack\_data：调用tm\_pack\_device\_data打包的返回数据指针  timeout\_ms：超时时间（毫秒） |
| 返回值 | 0：上报成功 |

调用tm\_pack\_device\_data时，传入的属性和事件的json格式如下：

|  |
| --- |
| {“prop1”:{ “value”:1, “time”:1603329629000},  “prop2”:{ “value”: “test”, “time”:1603329629000}} |

当使用tm\_data接口进行打包时可参考以下方式：

|  |
| --- |
| void \*data = tm\_data\_create();  tm\_data\_set\_int32(data, “prop1”, 1, 1603329629000);  tm\_data\_set\_string(data, “prop2”, “test”, 1603329629000);  // 对于每个属性功能点都可以调用相应的tm\_data\_set\_xxx接口来向data中打包数据，然后调用tm\_pack\_device\_data将data及其对应的产品信息进行打包，再进行发送 |

3.2.6 上报历史数据

|  |  |
| --- | --- |
| 功能 | 向平台批量上报若干设备的历史属性、事件 |
| 函数定义 | int32\_t tm\_post\_history\_data(void \*history\_data, uint32\_t timeout\_ms) |
| 参数 | history\_data：调用tm\_pack\_device\_data打包的返回数据指针  timeout\_ms：超时时间（毫秒） |
| 返回值 | 0：上报成功 |

调用tm\_pack\_device\_data时，传入的属性和事件的json格式如下：

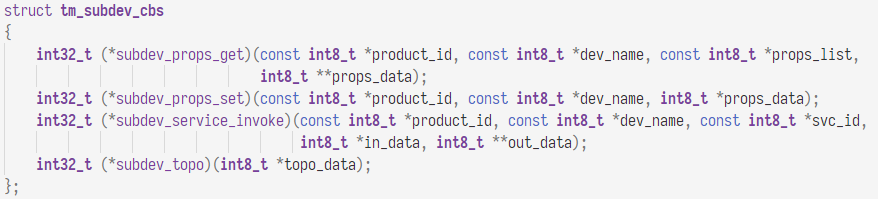
|  |
| --- |
| {“prop1”:[{ “value”:1, “time”:1603329629000},{ “value”:2, “time”:1603329630000}],  “prop2”:[{ “value”: “test”, “time”:1603329629000}]} |

3.3 子设备接口

3.3.1 子设备功能初始化

|  |  |
| --- | --- |
| 功能 | 为设备初始化子设备通道。(目前仅MQTT协议可用) |
| 函数定义 | int32\_t tm\_subdev\_init(struct tm\_subdev\_cbs callbacks) |
| 参数 | callbacks：子设备数据下发处理回调，定义参考移植说明。 |
| 返回值 | 0：初始化成功 |
| 说明 | 需要在调用tm\_init后使用 |

其中，struct\_subdev\_cbs定义如下：



|  |  |
| --- | --- |
| 功能 | 子设备属性获取 |
| 函数定义 | int (\*subdev\_props\_get)(const int8\_t \*product\_id, const int8\_t \*dev\_name, const int8\_t \*props\_list, int8\_t \*\*props\_data) |
| 参数 | product\_id：需要获取的子设备产品ID  dev\_name：需要获取的子设备名称  props\_list：需要获取的属性列表，json格式，例如“[“prop1”, “prop2”]”  props\_data：返回的属性数据，json格式，例如“{“prop1”:1,“prop2”:3}” |
| 返回值 | 0：获取成功，其它：获取失败 |
| 说明 | 返回参数props\_data指向的地址空间不得使用静态地址。 |

|  |  |
| --- | --- |
| 功能 | 子设备属性设置 |
| 函数定义 | int (\*subdev\_props\_set)(const int8\_t \*product\_id, const int8\_t \*dev\_name, int8\_t \*props\_data) |
| 参数 | product\_id：需要获取的子设备产品ID  dev\_name：需要获取的子设备名称  props\_data：需要设置的属性数据，json格式，例如“{“prop1”:1,“prop2”:3}” |
| 返回值 | 0：设置成功，其它：设置失败 |

|  |  |
| --- | --- |
| 功能 | 子设备服务调用 |
| 函数定义 | int (\*subdev\_service\_invoke)(const int8\_t \*product\_id, const int8\_t \*dev\_name, const int8\_t \*svc\_id, int8\_t \*in\_data, int8\_t \*\*out\_data) |
| 参数 | product\_id：需要调用的子设备产品ID  dev\_name：需要调用的子设备名称  svc\_id：需要调用的子设备服务标识符  in\_data：服务输入参数，json格式，例如“{“in1”:1,“in2”:3}”  out\_data：服务输出参数，json格式，例如“{“out”:4}” |
| 返回值 | 0：服务调用成功 |
| 说明 | 返回参数out\_data指向的地址空间不得使用静态地址。 |

|  |  |
| --- | --- |
| 功能 | 拓扑关系同步 |
| 函数定义 | int (\*subdev\_topo)(int8\_t \*topo\_data) |
| 参数 | topo\_data：平台下发的拓扑关系，包含当前网关设备已绑定的子设备信息，json格式，例如  “[{“productID”:“pid1”,“deviceName”:“dev1”}, {“productID”:“pid2”,“deviceName”:“dev2”}]” |
| 返回值 | 0：同步成功 |

3.3.2 添加子设备

|  |  |
| --- | --- |
| 功能 | 将指定子设备绑定到当前网关设备。 |
| 函数定义 | int32\_t tm\_subdev\_add(const int8\_t \*product\_id, const int8\_t \*dev\_name, const int8\_t \*access\_key, uint32\_t timeout\_ms) |
| 参数 | product\_id：子设备产品ID  dev\_name：子设备名称  access\_key：子设备产品key  timout\_ms：超时时间（毫秒） |
| 返回值 | 0：添加成功 |
| 说明 |  |

3.3.3 删除子设备

|  |  |
| --- | --- |
| 功能 | 将指定子设备解除与当前网关设备的绑定关系。 |
| 函数定义 | int32\_t tm\_subdev\_delete(const int8\_t \*product\_id, const int8\_t \*dev\_name, const int8\_t \*access\_key, uint32\_t timeout\_ms) |
| 参数 | product\_id：子设备产品ID  dev\_name：子设备名称  access\_key：子设备产品key  timout\_ms：超时时间（毫秒） |
| 返回值 | 0：添加成功 |
| 说明 |  |

3.3.3 获取子设备拓扑关系

|  |  |
| --- | --- |
| 功能 | 从平台获取当前网关绑定的子设备信息 |
| 函数定义 | int32\_t tm\_subdev\_topo\_get(uint32\_t timeout\_ms) |
| 参数 | timout\_ms：超时时间（毫秒） |
| 返回值 | 0：获取成功 |
| 说明 | 获取的子设备拓扑关系将通过tm\_subdev\_init注册的subdev\_topo回调返回 |

3.3.4 子设备登录

|  |  |
| --- | --- |
| 功能 | 子设备登录平台 |
| 函数定义 | int32\_t tm\_subdev\_login(const int8\_t \*product\_id, const int8\_t \*dev\_name, uint32\_t timeout\_ms) |
| 参数 | product\_id：需要登录的子设备产品ID  dev\_name；需要登录的子设备名称  timout\_ms：超时时间（毫秒） |
| 返回值 | 0：登录成功 |
| 说明 | 子设备登录前需确保已绑定到当前网关 |

3.3.5 子设备登出

|  |  |
| --- | --- |
| 功能 | 子设备登出平台 |
| 函数定义 | int32\_t tm\_subdev\_logout(const int8\_t \*product\_id, const int8\_t \*dev\_name, uint32\_t timeout\_ms) |
| 参数 | product\_id：需要登出的子设备产品ID  dev\_name；需要登出的子设备名称  timout\_ms：超时时间（毫秒） |
| 返回值 | 0：登出成功 |

3.3.5 子设备上报数据

|  |  |
| --- | --- |
| 功能 | 子设备上报数据到平台 |
| 函数定义 | int32\_t tm\_subdev\_post\_data(const int8\_t \*product\_id, const int8\_t \*dev\_name, int8\_t \*prop\_json, int8\_t \*event\_json, uint32\_t timeout\_ms) |
| 参数 | product\_id：需要登出的子设备产品ID  dev\_name；需要登出的子设备名称  prop\_json：需要上报的属性数据，json格式，例如：“{“prop1”:{ “value”:1, “time”:1603329629000}}  event\_json：需要上报的事件数据，json格式，例如：“{“event1”:{ “value”:{ “a”:1, “b”:2}, “time”:1603329629000}}  timout\_ms：超时时间（毫秒） |
| 返回值 | 0：上报成功 |

3.4 配置文件接口

配置文件由平台定义好物模型之后生成，根据功能点的功能、读写和数据类型，会在配置文件中生成相应的接口函数，不同的功能点对应不同的接口。

3.4.1 功能点数组

配置文件中会生成属性和事件功能点结构体数组，数组中会列出定义的功能点的读写类型和标识符信息，在调用3.2.1接口时，用户需将功能点数组及其长度作为参数传入。

|  |  |  |
| --- | --- | --- |
| 功能点类型 | 数组名 | 数组长度 |
| 属性 | tm\_prop\_list | tm\_prop\_list\_size |
| 服务 | tm\_svc\_list | tm\_svc\_list\_size |

3.4.2 数据下发（写属性点）

|  |  |
| --- | --- |
| 功能 | 用于接收平台下发的功能点数据，生成条件为功能点具备写操作 |
| 函数定义 | int32\_t tm\_prop\_yyy\_wr\_cb(void \*data) |
| 参数 | data：数据资源 |
| 返回值 | 0：下发操作成功；其他：下发操作失败 |
| 说明 | yyy为功能点标识符，只有属性功能点具备写操作 |

3.4.3 数据下发（读属性点）

|  |  |
| --- | --- |
| 功能 | 用于接收平台下发的功能点数据，生成条件为功能点具备读操作 |
| 函数定义 | int32\_t tm\_prop\_yyy\_rd\_cb(void \*data) |
| 参数 | data：数据资源 |
| 返回值 | 0：下发操作成功；其他：下发操作失败 |
| 说明 | yyy为功能点标识符，只有属性功能点具备写操作 |

3.4.4 数据下发（服务调用）

|  |  |
| --- | --- |
| 功能 | 用于接收平台下发的功能点数据，生成条件为功能点具备读操作 |
| 函数定义 | int32\_t tm\_svc\_yyy\_cb(void \*in\_data, void \*out\_data) |
| 参数 | in\_data：由平台下发的服务执行输入数据  out\_data：需要返回平台的的服务执行输出数据 |
| 返回值 | 0：下发操作成功；其他：下发操作失败 |
| 说明 | yyy为功能点标识符，只有属性功能点具备写操作 |

3.4.5 数据上传

|  |  |
| --- | --- |
| 功能 | 以阻塞方式上传功能点数据，生成条件为功能点具备读操作 |
| 函数定义 | int32\_t tm\_xxx\_yyy\_notify(  void \*data,  zzz val,  uint64\_t timestamp,  uint32\_t timeout\_ms  ) |
| 参数 | data：数据资源  val：上传数据  timestamp：时间戳（没有则为0）  timeout\_ms：上传超时时间（毫秒） |
| 返回值 | 0：上传成功；其他：上传失败 |
| 说明 | xxx为prop或event（由功能类型决定），yyy为功能点标识符，zzz为数据类型 |

3.5 系统适配接口

3.5.1 内存接口

3.5.1.1 内存分配

|  |  |
| --- | --- |
| 功能 | 内存分配 |
| 函数定义 | void \*osl\_malloc(size\_t size) |
| 参数 | size：申请内存长度 |
| 返回值 | 若分配成功，则返回指向被分配内存区域的指针；若分配失败，则返回NULL |

3.5.1.2 内存释放

|  |  |
| --- | --- |
| 功能 | 内存释放 |
| 函数定义 | void osl\_free(void \*ptr) |
| 参数 | ptr：待释放内存区域地址 |
| 返回值 | 无 |

3.5.1.3 内存拷贝

|  |  |
| --- | --- |
| 功能 | 将源内存中的值拷贝到目标内存中 |
| 函数定义 | void \*osl\_memcpy(void \*dst, const void \*src, size\_t n) |
| 参数 | dst：目标内存地址  src：源内存地址  n：拷贝字节数 |
| 返回值 | 无 |

3.5.1.4 内存初始化

|  |  |
| --- | --- |
| 功能 | 内存区域初始化，将指定内存区域设置为某个值 |
| 函数定义 | void \*osl\_memset(void \*dst, int32\_t val, size\_t n) |
| 参数 | dst：待初始化内存区域地址  val：待初始化值  n：待初始化内存区域长度 |
| 返回值 | 无 |

3.5.2 网络接口

3.5.2.1 TCP接口

3.5.2.1.1 创建网络连接

|  |  |
| --- | --- |
| 功能 | 创建TCP连接 |
| 函数定义 | handle\_t tcp\_connect(  const char \*host,  uint16\_t port,  uint32\_t timeout\_ms  ) |
| 参数 | host：目标地址，支持点分十进制IP和域名形式  port：目标端口  timeout\_ms：执行连接的超时时间（毫秒） |
| 返回值 | -1：失败；其他：网络操作句柄 |

3.5.2.1.2 数据发送

|  |  |
| --- | --- |
| 功能 | 发送TCP数据 |
| 函数定义 | int32\_t tcp\_send(  handle\_t handle,  void \*buf,  uint32\_t len,  uint32\_t timeout\_ms  ) |
| 参数 | handle：网络操作句柄  buf：需要发送的数据缓冲区地址  len：需要发送的数据长度  timeout\_ms：执行发送的超时时间（毫秒） |
| 返回值 | -1：失败；0：超时；其他：成功发送的数据长度 |

3.5.2.1.3 数据接收

|  |  |
| --- | --- |
| 功能 | 接收TCP数据 |
| 函数定义 | int32\_t tcp\_recv(  handle\_t handle,  void \*buf,  uint32\_t len,  uint32\_t timeout\_ms  ) |
| 参数 | handle：网络操作句柄  buf：用于接收数据的缓冲区地址  len：需要接收的数据长度  timeout\_ms：执行接收的超时时间（毫秒） |
| 返回值 | -1：失败；0：超时；其他：成功接收的数据长度 |

3.5.2.1.4 断开网络连接

|  |  |
| --- | --- |
| 功能 | 断开TCP连接 |
| 函数定义 | int32\_t tcp\_disconnect(handle\_t handle) |
| 参数 | handle：需要断开的网络操作句柄 |
| 返回值 | 0：成功 |

3.5.2.2 UDP接口

3.5.2.2.1 创建网络连接

|  |  |
| --- | --- |
| 功能 | 创建UDP连接 |
| 函数定义 | handle\_t udp\_connect(const char \*host, uint16\_t port) |
| 参数 | host：目标地址，支持点分十进制IP和域名形式  port：目标端口 |
| 返回值 | -1：失败；其他：网络操作句柄 |

3.5.2.2.2 数据发送

|  |  |
| --- | --- |
| 功能 | 发送UDP数据 |
| 函数定义 | int32\_t udp\_send(  handle\_t handle,  void \*buf,  uint32\_t len,  uint32\_t timeout\_ms  ) |
| 参数 | handle：网络操作句柄  buf：需要发送的数据缓冲区地址  len：需要发送的数据长度  timeout\_ms：执行发送的超时时间（毫秒） |
| 返回值 | -1：失败；0：超时；其他：成功发送的数据长度 |

3.5.2.2.3 数据接收

|  |  |
| --- | --- |
| 功能 | 接收UDP数据 |
| 函数定义 | int32\_t udp\_recv(  handle\_t handle,  void \*buf,  uint32\_t len,  uint32\_t timeout\_ms  ) |
| 参数 | handle：网络操作句柄  buf：用于接收数据的缓冲区地址  len：需要接收的数据长度  timeout\_ms：执行接收的超时时间（毫秒） |
| 返回值 | -1：失败；0：超时；其他：成功接收的数据长度 |

3.5.2.2.4 断开网络连接

|  |  |
| --- | --- |
| 功能 | 断开UDP连接 |
| 函数定义 | int32\_t udp\_disconnect(handle\_t handle) |
| 参数 | handle：需要断开的网络操作句柄 |
| 返回值 | 0：成功 |

3.5.3 时间接口

3.5.3.1 获取系统时间

|  |  |
| --- | --- |
| 功能 | 获取毫秒级时间计数 |
| 函数定义 | uint64\_t time\_count\_ms(void) |
| 参数 | 无 |
| 返回值 | 当前毫秒级计数值 |

3.5.3.2 倒计时器启动

|  |  |
| --- | --- |
| 功能 | 启动倒计时 |
| 函数定义 | handle\_t countdown\_start(uint32\_t ms) |
| 参数 | ms：倒计时时间（毫秒） |
| 返回值 | 0：失败；其他：倒计时器操作句柄 |

3.5.3.3 倒计时器配置

|  |  |
| --- | --- |
| 功能 | 配置倒计时器 |
| 函数定义 | void countdown\_set(handle\_t handle, uint32\_t new\_ms) |
| 参数 | handle：倒计时器操作句柄  new\_ms：倒计时器超时时间（毫秒） |
| 返回值 | 无 |

3.5.3.4 倒计时器剩余时间

|  |  |
| --- | --- |
| 功能 | 返回倒计时器剩余时间 |
| 函数定义 | uint32\_t countdown\_left(handle\_t handle) |
| 参数 | handle：倒计时器操作句柄 |
| 返回值 | 倒计时器剩余时间 |

3.5.3.5 倒计时器超时判断

|  |  |
| --- | --- |
| 功能 | 判断倒计时器是否已超时 |
| 函数定义 | uint32\_t countdown\_is\_expired(handle\_t handle) |
| 参数 | handle：倒计时器操作句柄 |
| 返回值 | 0：未超时；1：已超时 |

3.5.3.6 倒计时器停止

|  |  |
| --- | --- |
| 功能 | 停止倒计时器，销毁资源 |
| 函数定义 | void countdown\_stop(handle\_t handle) |
| 参数 | handle：倒计时器操作句柄 |
| 返回值 | 无 |

四．SDK移植说明

除了标准C库，SDK包含了所有使用到的第三方库，用户可直接将SDK添加进自己的工程文件，或基于SDK文件进行开发。SDK包含CMakeLists文件，Linux环境下可用cmake指令生成makefile进行编译。

4.1 移植流程说明

1. 物模型SDK默认编译为32位程序，若用户使用的平台为64位，则修改CMakeLists中的编译系统位数为“64-bit”，并将platforms\include\data\_types.h文件中的handle\_t类型定义为long long；

2. 根据硬件平台与编译环境修改系统适配接口，如网络、时间等接口；

3. 在主逻辑中调用设备初始化与登录接口，实现设备平台接入；

4. 登录成功后，频繁调用数据解析接口，保证数据上下行业务的正常；

5. 在配置文件中实现功能点下发控制逻辑，并根据需要在主逻辑中调用数据上传接口上传数据；

6. 若数据解析接口返回错误，则调用设备注销接口断开平台连接，并重新调用设备登录接口登录平台。

4.2 系统接口说明

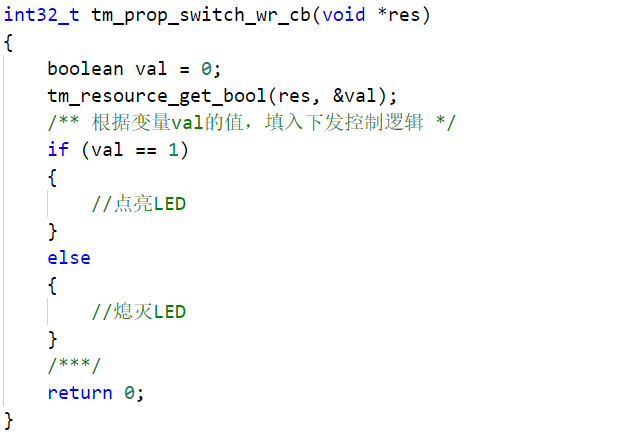
SDK系统接口默认基于Linux环境编写，若编译环境为Linux，则无需进行多少调整；若用户采用“单片机+标准通信模组”的方案，模组只提供TCP或UDP通信能力，则用户需根据具体平台调整时间接口，并将网络接口中的数据收发修改为单片机与模组之间的AT数据通信，数据为MQTT报文或CoAP报文，并且对于使用MQTT协议加密的情况时，用户需移植wolfssl加密库来适配相应的单片机平台，包括修改wolfssl\port\user\_settings.h文件，相对较为麻烦，所以推荐MQTT走非加密的方式。

4.3 配置文件说明

平台端下载的配置文件包括tm\_user.c和tm\_user.h文件，用户需将其放入SDK的onenet\thing\_model\user目录中。

4.3.1数据下发（写属性点）

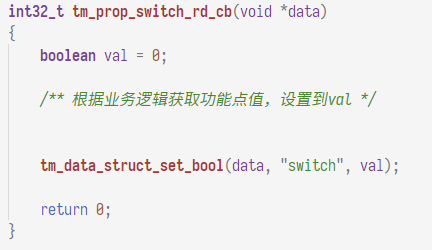
若用户在平台定义的属性功能点具备写属性，则配置文件会生成相应的写函数，平台进行数据下发时会进入对应的功能点写函数，用户需在函数中根据下发的数据执行相关逻辑，如控制开关、调整LED亮度等。例：



当平台同时下发多个功能点数据时，会分别进入各个功能点写函数执行用户逻辑，当所有逻辑执行完毕之后，统一进行平台回复。

4.3.2数据下发（读属性点）

若用户在平台定义的属性功能点具备读属性，则配置文件会生成相应的读函数，平台进行数据下发时会进入对应的功能点读函数，用户需在函数中根据实际硬件设计获取功能点值设置到参数val中。例：



4.3.3数据下发（服务调用）

若用户在平台定义的服务功能点后，则配置文件会生成相应的服务调用函数，平台进行数据下发时会进入对应的服务调用回调函数，用户需在函数中根据服务定义，由输入参数运算得到输出参数，设置到val中。例：

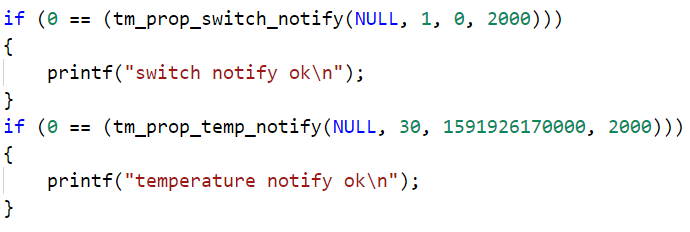


4.3.4 数据上传

若用户在平台定义的功能点具备读属性，则配置文件会生成相关的数据上传函数，用户调用上传函数即可实现对应功能点的数据上传。需注意以下几点：

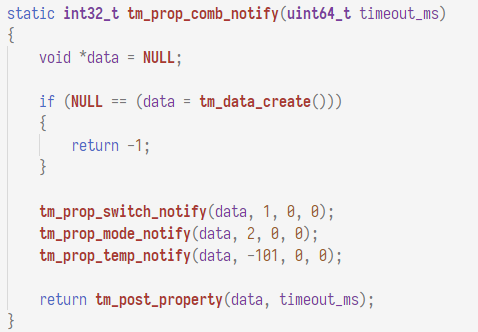
1. 单功能点上传：

若每次只需上传单个功能点，则直接调用上传函数即可，但data参数必须传入NULL，timestamp参数为毫秒级的Unix时间戳或0。例：



2. 多功能点上传：

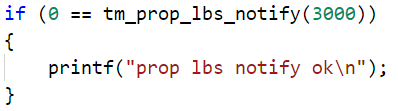
若想同时上传多个功能点（一包数据携带多个功能点信息），则需封装需要上传的功能点notify函数，可参考如下示例：



3. 数组上传：

数组上传类似多功能点上传，区别在于封装时调用相同的notify函数（调用次数取决于数组包含的元素个数），并传入各自的数组元素值。例：





4.5 子设备接口

要使用子设备接口，需要在config.h中设置为MQTT协议，并打开子设备功能宏：

|  |
| --- |
| #define FEATURE\_TM\_GATEWAY\_ENABLED |

4.7 移植注意事项

1. 为保证平台下发功能的即时响应，数据解析接口必须以尽量小的时间间隔调用，因此勿在主循环中加入等待延时；

2. 勿在回调函数中直接调用设备相关操作函数，例如不要在功能点写函数中调用数据上传函数，推荐在回调中标记该操作，然后在主循环中执行；

3. 代码使用了部分C99特性，编译时必须开启编译器的C99选项；

4. 在系统资源比较紧张的情况下，用户可适当减少功能点定义的数量，并尽量避免上传过长的数据（字符型），以减少内存的占用；

5．默认缓冲区和数据封包长度最大为1024字节，用户可根据上传或下发的数据量大小进行调整；

6. 采用MQTT协议，心跳时间不能超过5分钟；采用CoAP协议，心跳时间需介于16秒至7天之间。