OneNET Studio 设备接入 SDK

使用说明

版本号：1.2.0

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	
2020/12/10	v1.2.0	宋伟、陈龙宇	新增 OTA 功能、文件管理服务	

目录

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
services	ota		远程升级 OTA 文件
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
FEATURE_TM_GATEWAY_ENABLED	子设备功能使能
FEATURE_TM_OTA_ENABLED	OTA 功能使能

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。

3.2.7 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

3.2.8 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

3.2.9 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 tm_subdev_cbs 定义如下：

```
struct tm_subdev_cbs
{
    int32_t (*subdev_props_get)(const int8_t *product_id, const int8_t *dev_name, const int8_t *props_list,
                                int8_t **props_data);
    int32_t (*subdev_props_set)(const int8_t *product_id, const int8_t *dev_name, int8_t *props_data);
    int32_t (*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);
    int32_t (*subdev_topo)(int8_t *topo_data);
};
```

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	

3.3.4 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

3.3.5 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

3.3.6 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

3.3.7 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

3.6 远程升级接口

3.6.1 OTA 服务接口

3.6.1.1 创建上下文

功能	创建 OTA 上下文
函数定义	uint8_t ota_init(void **ota_ctx, const char *product_id, const char *device_sn, const char *access_key, uint8_t task_type, const char *firmware_version, const char *software_version, uint32_t recv_max_len, uint16_t check_period_s, uint32_t timeout_ms)

参数	ota_ctx：待初始化的上下文指针地址 product_id：产品 id device_sn：设备 sn access_key：产品 key task_type：任务类型，1：fota；2：sota firmware_version：模组固件版本号 software_version：应用固件版本号 recv_max_len：用于接收 tcp 报文的 buffer 总长度 check_period_s：定时检查任务周期 timeout_ms：超时时间（毫秒）
返回值	OTA_OK：初始化成功；OTA_ERROR：初始化失败

3.6.1.2 销毁上下文

功能	销毁 OTA 上下文
函数定义	uint8_t ota_deinit(ota_context *ota_ctx)
参数	ota_ctx：ota 上下文

3.6.1.3 上报版本

功能	上报固件版本号
函数定义	uint8_t OTA_Submit_Version(ota_context *ota_ctx, uint32_t timeout_ms)
参数	ota_ctx：ota 上下文 timeout_ms：超时时间（毫秒）
返回值	OTA_OK：上报成功；OTA_ERROR：上报失败
说明	将根据初始化 ota 服务时传入的模组固件、应用固件版本号进行上报

3.6.1.4 检查任务

功能	检查升级任务
函数定义	uint8_t OTA_Check_Task (ota_context *ota_ctx, uint32_t timeout_ms)
参数	ota_ctx：ota 上下文 timeout_ms：超时时间（毫秒）

返回值	OTA_OK：检查成功；OTA_ERROR：检查失败
说明	检查任务成功表示成功获取检查结果，此时可能存在升级任务，也可能不存在升级任务

3.6.1.5 校验 TID

功能	检测设备任务状态
函数定义	uint8_t OTA_Validate_TID (ota_context *ota_ctx, uint32_t timeout_ms)
参数	ota_ctx：ota 上下文 timeout_ms：超时时间（毫秒）
返回值	OTA_OK：检验成功；OTA_ERROR：上报失败

3.6.1.6 下载文件

功能	下载设备升级包
函数定义	uint8_t OTA_Download_Package (ota_context *ota_ctx, uint32_t segment_start, uint32_t segment_end, uint32_t timeout_ms)
参数	ota_ctx：ota 上下文 segment_start：本次下载分片包起始位置 segment_end：本次下载分片包终止位置 timeout_ms：下载文件超时时间
返回值	OTA_OK：下载成功；OTA_ERROR：下载失败

3.6.1.7 上报进度

功能	上报设备升级进度或状态
函数定义	uint8_t OTA_Update_Progress (ota_context *ota_ctx, uint32_t timeout_ms)
参数	ota_ctx：ota 上下文 timeout_ms：超时时间（毫秒）

返回值	OTA_OK：上报成功；OTA_ERROR：上报失败

3.6.2 OTA 移植接口

3.6.2.1 获取下载范围

功能	获取下次获取分片包的范围
函数定义	void ota_calculate_segment_range(uint32_t package_now, uint32_t *start, uint32_t *end)
参数	package_now：目前已经下载的升级包的范围 start：用于保存下次获取的范围的起始位置 end：用于保存下次获取的范围的终止位置

3.6.2.2 保存现场

功能	保存 ota 相关信息至存储介质
函数定义	uint8_t ota_scene_store (void *user_context)
参数	user_context：ota 上下文

3.6.2.3 恢复现场

功能	从存储介质中恢复 ota 相关信息
函数定义	uint8_t ota_scene_recall(void *user_context)
参数	user_context：ota 上下文

3.6.2.4 日志输出

功能	日志输出
函数定义	int ota_log_printf(const char *format, ...)
说明	可根据需要选择保存 ota 日志

3.7 文件管理接口

目前文件管理相关的接口都是基于 HTTP 接口，要求终端必须具备 TCP 传输方式。

3.7.1 通用定义

```
struct tm_file_data_t
{
    /** 本次获取的数据在文件数据的起始位置，从0开始*/
    uint32_t seq;
    /** 获取到的数据缓冲区地址*/
    uint8_t *data;
    /** 获取到的数据长度*/
    uint32_t data_len;
};

enum tm_file_event_e
{
    /** 获取文件数据回调，参数为struct tm_file_data_t*/
    TM_FILE_EVENT_GET_DATA = 0,
    /** 上传文件成功，参数为文件在平台保存的唯一ID，字符串型*/
    TM_FILE_EVENT_POST_SUCCESSED,
    /** 上传文件失败，参数为平台返回的错误信息，字符串型*/
    TM_FILE_EVENT_POST_FAILED,
};
typedef int32_t tm_file_cb(enum tm_file_event_e event, void *event_data);
```

3.7.2 文件获取

功能	获取文件
函数定义	int32_t tm_file_get(tm_file_cb callback, int8_t *file_id, uint32_t file_size, uint32_t segment_size, uint32_t timeout_ms)
参数	callbacks：获取文件事件回调，参见通用定义； file_id：平台下发的目标文件唯一 ID，可在服务调用回调中获取； file_size：需要获取的文件大小，可在服务调用回调中获取； segment_size：为节省终端资源，文件支持分片获取，本参数用于设置分片大小； timeout_ms：操作超时时间，单位毫秒
返回值	0：成功 非 0：失败
说明	文件获取功能基于 OneNET Studio 的服务实现，通过下发服务指令，通知终端需要获取的目标文件唯一 ID、文件名、文件大小，终端在服务处理回调（tm_svc_$AsyncGet_cb、tm_svc_$SyncGetFile_cb）中使用本接口来获取文件内容数据。 由于接口调用嵌入到服务回调中，使用的 timeout_ms 参数过大可能导致后续数据处理超时或丢失。

3.7.3 文件上传

功能	向平台上传指定文件
函数定义	int32_t tm_file_post(tm_file_cb callback, const int8_t *product_id, const int8_t *dev_name, const int8_t *access_key, const int8_t *file_name, uint8_t *file, uint32_t file_size, uint32_t timeout_ms)
参数	callbacks：上传文件事件回调，参见通用定义； product_id：设备所属的产品 ID； dev_name：设备名称； access_key：产品 key； file_name：文件名（平台目前仅支持少数文件格式，具体限制参考平台文件管理页面说明）； file：文件数据地址； file_size：需要上传的文件长度； timeout_ms：操作超时时间，单位毫秒
返回值	0：上传成功 非 0：错误
说明	

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：

```
int32_t tm_prop_switch_wr_cb(void *res)
{
    boolean val = 0;
    tm_resource_get_bool(res, &val);
    /** 根据变量val的值，填入下发控制逻辑 */
    if (val == 1)
    { //点亮LED
    }
    else
    {
        //熄灭LED
    }
    return 0;
}
```

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：

```
int32_t tm_prop_switch_rd_cb(void *data)
{
    boolean val = 0;

    /** 根据业务逻辑获取功能点值，设置到val */
    tm_resource_set_bool(data, "switch", val);

    return 0;
}
```

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：

```
int32_t tm_svc_add_cb(void *in_data, void *out_data)
{
    struct svc_add_in_t in_param;
    struct svc_add_out_t out_param;

    int32_t ret = 0;

    tm_data_struct_get_int32(in_data, "a", &in_param.a);
    tm_data_struct_get_int32(in_data, "b", &in_param.b);

    /** 根据输入参数，生成输出参数*/
    //out_param.c = in_param.a + in_param.b;

    tm_data_struct_set_int32(out_data, "c", out_param.c);
    return ret;
}
```

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：

```
if(0 == (tm_prop_switch_notify(NULL, 1, 0, 2000)))
{
    printf("switch notify ok\n");
}
if(0 == (tm_prop_temp_notify(NULL, 30, 1591926170000, 2000)))
{
    printf("temperature notify ok\n");
}
```

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：

```
static int32_t tm_prop_combine_notify(uint64_t timeout_ms)
{
    void *data = tm_data_create();

    tm_prop_a_notify(data, TRUE, 0, 0);
    tm_prop_b_notify(data, 3, 0, 0);
    tm_prop_c_notify(data, 3.14, 0, 0);

    return tm_post_property(data, timeout_ms);
}
```

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：

```
static int32_t tm_prop_lbs_notify(uint64_t timeout_ms)
{
    void *data = tm_data_create();
    struct prop_$OneNET_LBS_t val;

    val.mnc = 1;
    val.mcc = 2;
    tm_prop_$OneNET_LBS_notify(data, val, 1591926170000, 0);

    val.mnc = 11;
    val.mcc = 22;
    tm_prop_$OneNET_LBS_notify(data, val, 0, 0);

    val.mnc = 111;
    val.mcc = 222;
    tm_prop_$OneNET_LBS_notify(data, val, 0, 0);

    return tm_post_property(data, timeout_ms);
}

if (0 == tm_prop_lbs_notify(3000))
{
    printf("prop lbs notify ok\n");
}
```

4.4 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
```

4.5 远程升级说明

1. 需要在 config.h 中打开远程升级功能使能宏：FEATURE_TM_OTA_ENABLED
2. 在调用 tm_init 进行设备初始化时，需要在 struct tm_downlink_tbl_t *downlink_tbl 参数中对 ota_cb 这一结构体成员进行赋值，例如：

```
    downlink_tbl.prop_tbl = tm_prop_list;
    downlink_tbl.prop_tbl_size = tm_prop_list_size;
    downlink_tbl.svc_tbl = tm_svc_list;
    downlink_tbl.svc_tbl_size = tm_svc_list_size;
#if FEATURE_TM_OTA_ENABLED
    // int32_t tm_ota_event(void);
    dl_tbl.ota_cb = tm_ota_event;
#endif

    if(0 == tm_init(&downlink_tbl))
    {
        printf("tm init ok\n");
    }
```

3. 当 OneNET Studio 下发远程升级通知后，会通过 ota_cb 进行回调函数通知，随后可自行根据业务逻辑触发远程升级
4. 进行远程升级前需调用 ota_init 函数进行初始化，其中模组固件版本和应用固件版本不可同时为空，且任务类型需要与 OneNET Studio 远程升级页面所创建的升级任务类型所匹配（1 表示 fota 任务，2 表示 sota 任务）
5. 可根据实际需求自行调用 ota_api 中的函数（可根据 ota_loop() 参考实现），也可直接循环调用 ota_loop() 函数由既定的流程自动进行 ota 升级
6. ota 远程升级任务执行过程中，会触发不同的事件回调至 OTA_Event_Handle 中，其中较为重要的事件有：
   a) OTA_EVENT_custom_save_packet：需要将下载的分片包存入存储介质；
   b) OTA_EVENT_custom_delete_package：擦除升级包存储区域；
   c) OTA_EVENT_custom_ready_update：升级包下载完毕后，需要校验升级包完整性并进行固件升级；
   d) OTA_EVENT_REPORT_DOWNLOAD_PROGRESS：选择上报下载进度的方式，默认每 10%上报一次。

4.6 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。