

自动测试平台终端接入库接口设计说明书

|  |  |  |  |
| --- | --- | --- | --- |
| 文件版本： | V1.0.3 | 文件编号： |  |
| 首版发布日期： | 2019年11月20日 | 编制： |  |
| 审核： |  | 批准： |  |

中国移动统计集团设计院有限公司

**修订情况**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **版本** | **修订章节** | **修订内容概述** | **修订日期** | **修订人** | **修订前版本号** | **批准人** | **备注** |
| V1.0 |  | 文档初建。 | 2019年11月26日 | 蒋巍 |  |  |  |
| V1.0.3 | 2.2 | 插入2.2.2，补充接口“获取接入库句柄” | 2020年04月02日 | 蒋巍 |  |  |  |
| V1.1 | 3.1，3.3.4 | 更新消息定义，注明废弃消息 | 2021年03月30日 | 蒋巍 | V1.0.3 |  |  |
| 3.2，3.3.5 | 更新消息结构定义，注明废弃消息 |  |  |
| 3.1，3.3.4 | 增加模块配置上报消息定义 |  |  |
| 3.2，3.3.5 | 增加模块配置上报消息结构定义 |  |  |
| 4.2.10 | 补充模块配置上报流程说明 |  |  |
| V1.2 | 3.2.31 | 修改模块配置上报消息及定义 | 2021年04月06日 | 蒋巍 | V1.1 |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

**目录**

[1 引言 1](#_Toc68014423)

[1.1 目的 1](#_Toc68014424)

[1.2 范围 1](#_Toc68014425)

[1.3 术语定义 1](#_Toc68014426)

[1.4 参考资料 1](#_Toc68014427)

[2 接口说明 2](#_Toc68014428)

[2.1 客户端接口说明 2](#_Toc68014429)

[2.1.1 接入库版本 2](#_Toc68014430)

[2.1.2 初始队列 3](#_Toc68014431)

[2.1.3 释放队列 5](#_Toc68014432)

[2.1.4 消息发送 5](#_Toc68014433)

[2.1.5 数据接收（阻塞方式） 8](#_Toc68014434)

[2.1.6 数据接收（回调方式） 9](#_Toc68014435)

[2.2 平台侧接口说明 12](#_Toc68014436)

[2.2.1 接入库版本 12](#_Toc68014437)

[2.2.2 获取接入库句柄 13](#_Toc68014438)

[2.2.3 连接消息队列 14](#_Toc68014439)

[2.2.4 释放队列 16](#_Toc68014440)

[2.2.5 消息发送 17](#_Toc68014441)

[2.2.6 数据接收等待（阻塞方式） 20](#_Toc68014442)

[2.2.7 数据读取 21](#_Toc68014443)

[3 接口数据结构定义 23](#_Toc68014444)

[3.1 消息类型定义 23](#_Toc68014445)

[3.2 客户端结构体定义 24](#_Toc68014446)

[3.2.1 登录请求 24](#_Toc68014447)

[3.2.2 登录响应 24](#_Toc68014448)

[3.2.3 登出请求 25](#_Toc68014449)

[3.2.4 登出响应 25](#_Toc68014450)

[3.2.5 配置更新请求 25](#_Toc68014451)

[3.2.6 配置更新响应 26](#_Toc68014452)

[3.2.7 配置数据下发请求 26](#_Toc68014453)

[3.2.8 配置数据下发响应 27](#_Toc68014454)

[3.2.9 配置更新通知请求 27](#_Toc68014455)

[3.2.10 配置更新通知响应 28](#_Toc68014456)

[3.2.11 事件上报请求（废弃） 28](#_Toc68014457)

[3.2.12 事件上报响应（废弃） 28](#_Toc68014458)

[3.2.13 告警请求（废弃） 29](#_Toc68014459)

[3.2.14 告警响应（废弃） 29](#_Toc68014460)

[3.2.15 告警消除请求（废弃） 30](#_Toc68014461)

[3.2.16 告警消除响应（废弃） 30](#_Toc68014462)

[3.2.17 GPS上报请求（废弃） 30](#_Toc68014463)

[3.2.18 GPS上报响应（废弃） 31](#_Toc68014464)

[3.2.19 状态上报请求（废弃） 31](#_Toc68014465)

[3.2.20 状态上报响应（废弃） 32](#_Toc68014466)

[3.2.21 上传文件请求 32](#_Toc68014467)

[3.2.22 上传文件请求响应 33](#_Toc68014468)

[3.2.23 文件数据上传请求 33](#_Toc68014469)

[3.2.24 文件数据上传响应 34](#_Toc68014470)

[3.2.25 文件上传结束请求 34](#_Toc68014471)

[3.2.26 上传文件结束响应 35](#_Toc68014472)

[3.2.27 查询文件数据请求 35](#_Toc68014473)

[3.2.28 查询文件数据响应 36](#_Toc68014474)

[3.2.29 重启终端请求（废弃） 36](#_Toc68014475)

[3.2.30 重启终端响应（废弃） 36](#_Toc68014476)

[3.2.31 模块配置上报 37](#_Toc68014477)

[3.2.32 模块配置上报响应 38](#_Toc68014478)

[3.3 Protocolbuf消息定义 38](#_Toc68014479)

[3.3.1 语法要求 38](#_Toc68014480)

[3.3.2 消息正反序列化 38](#_Toc68014481)

[3.3.3 消息类型 39](#_Toc68014482)

[3.3.4 消息ID 39](#_Toc68014483)

[3.3.5 消息定义 40](#_Toc68014484)

[3.4 常量定义 58](#_Toc68014485)

[3.4.1 CmdErrorCode定义 58](#_Toc68014486)

[4 流程定义 60](#_Toc68014487)

[4.1 接口库调用流程 60](#_Toc68014488)

[4.1.1 初始化及释放流程： 60](#_Toc68014489)

[4.1.2 客户端的发送与接收 61](#_Toc68014490)

[4.1.3 远程管理单元的发送与接收 63](#_Toc68014491)

[4.2 业务功能流程 64](#_Toc68014492)

[4.2.1 登入登出 64](#_Toc68014493)

[4.2.2 配置更新 65](#_Toc68014494)

[4.2.3 事件上报（废弃） 66](#_Toc68014495)

[4.2.4 告警消警（废弃） 67](#_Toc68014496)

[4.2.5 GPS上报（废弃） 68](#_Toc68014497)

[4.2.6 状态上报（废弃） 68](#_Toc68014498)

[4.2.7 上传测试数据文件 69](#_Toc68014499)

[4.2.8 查询测试文件数据 70](#_Toc68014500)

[4.2.9 重启终端（废弃） 71](#_Toc68014501)

[4.2.10 模块配置上报 72](#_Toc68014502)

[5 裁剪说明 72](#_Toc68014503)

1. 引言
   1. 目的

自动测试平台的测试终端在接入平台时，需要通过接入库（libamtpca.so）实现，本文档针对测试终端对接入库的调用进行定义及说明，适用于终端开发人员。

* 1. 范围

本文档只涉及接入库的外部调用接口，接入库与平台的消息队列接口及交互流程等，会在其他相关文档中说明定义。

* 1. 术语定义

|  |  |
| --- | --- |
| 术语 | 定义 |
| *[术语名称或缩写]* | *[对术语或缩写进行说明或描述]* |

* 1. 参考资料

《自动路测系统接口规范V1.6.0》

1. 接口说明
   1. 客户端接口说明
      1. 接入库版本

* **定义**

const char \* amtpca\_version();

* **功能**

获取加载的接入库版本；

* **描述**

客户端加载libamtpca.so后，通过调用amtpca\_version()获取libamtpca.so的版本，版本信息以字符串方式返回，例如“libamtpca version 1.0.3.20200227”，此方法调用非必须；

* **参数**

无；

* **返回值**

返回字符串，SDK的版本版本信息；

* **流程**

参见第4章流程定义中的4.1.1初始化及释放流程；

* **范例**

using LIB\_AMTPCA\_VERSION = char\* (\*)();

LIB\_AMTPCA\_VERSION lib\_version ;

void \*handle;

lib\_version = NULL;

handle = dlopen("libamtpca.so", RTLD\_NOW);

if (handle == NULL)

{

fprintf(stderr, "Failed to open libaray libamtpca.so error:%s\n", dlerror());

return ;

}

lib\_version = (LIB\_AMTPCA\_VERSION)dlsym(handle, "amtpca\_version");

fprintf(stderr, " Version = %s\n", lib\_version());

……

dlclose(handle);

* + 1. 初始队列
* **定义**

int amtpca\_init(const char \* license\_path, const char \* log\_path, LIBAMTPCA\_CMD\_RECV\_CBFUN call\_back);

* **功能**

初始化客户端接入库；

* **描述**

客户端加载libamtpca.so后，需要调用amtpca\_init()完成对接入库的初始化，只有在初始化成功后，接入库才能完成资源分配及与平台的通道建立，接入库其他接口方法才有效（除amtpca\_version()），客户端才可以接入平台，进行消息及数据的交换。libamtpca.so加载后只需要进行一次初始化，amtpca\_init()与amtpca\_release()配合使用；

* **参数**

**license\_path**：客户端使用的授权文件的全路径文件名，客户端只有在经过平台授权后方可接入平台，授权以license文件方式发布；

**log\_path**：接入库日志的存放路径，如果该参数为NULL，日志将输出至stderr；

**call\_back**：接入库数据接收的回调函数。接入库采用回调和amtpca\_waitforcmd（详见2.1.5）进行数据接收，所有接收数据都会经过回调，如此参数为NULL，则应用程序只能通过amtpca\_waitforcmd从接入库读取接收数据；

* **返回值**

返回整型数，如果amtpca\_init () 函数执行成功，会返回0。否则返回小于0的整型数；

LIB\_AMTPA\_SUCCESS(0)：成功；

LIB\_AMTPA\_IS\_WORKING (-101)：已经进行了初始化；

LIB\_AMTPA\_PARA\_ERROR (-103)：参数错误或非法；

LIB\_AMTPA\_OPEN\_LIB\_FAILED(-104)：接入库资源加载错误；

LIB\_AMTPA\_LICENSE\_ERROR(-111)：授权信息错误；

LIB\_JWUMQ\_SOCKET\_FAILD(-6)：消息队列socket错误；

LIB\_JWUMQ\_CONNECT\_FAILD(-7)：消息队列连接错误；

* **流程**

参见第4章流程定义中的4.1.1初始化及释放流程；

* **范例**

using LIB\_AMTPCA\_VERSION = char\* (\*)();

using LIB\_AMTPCA\_INIT = int (\*)(const char \* , const char \* , LIBAMTPCA\_CMD\_RECV\_CBFUN );

using LIB\_AMTPCA\_RELEASE = void (\*)();

LIB\_AMTPCA\_VERSION lib\_version ;

LIB\_AMTPCA\_INIT lib\_init;

LIB\_AMTPCA\_RELEASE lib\_release;

void \*handle;

lib\_version = NULL;

handle = dlopen("libamtpca.so", RTLD\_NOW);

if (handle == NULL)

{

fprintf(stderr, "Failed to open libaray libamtpca.so error:%s\n", dlerror());

return ;

}

lib\_version = (LIB\_AMTPCA\_VERSION)dlsym(handle, "amtpca\_version");

lib\_init = (LIB\_AMTPCA\_INIT)dlsym(handle, "amtpca\_init");

lib\_release = (LIB\_AMTPCA\_RELEASE)dlsym(handle, "amtpca\_release");

fprintf(stderr, " Version = %s\n", lib\_version());

int result = lib\_init("dev.license", NULL/\*"/tmp/log"\*/, std::bind(&LibAmtpcaTest::LibamtpcaCmdRecvCbfun, this

, std::placeholders::\_1, std::placeholders::\_2, std::placeholders::\_3));

if(result != 0)

{

return;

}

……

lib\_release();

dlclose(handle);

* + 1. 释放队列
* **定义**

void amtpca\_release();

* **功能**

释放接入库资源；

* **描述**

客户端加载libamtpca.so后，需要调用amtpca\_init()完成对接入库的初始化。在完成客户端业务功能后，需要调用amtpca\_release()完成接入库的清理，避免内存泄露等问题。libamtpca.so加载后只需要进行一次初始化和释放，amtpca\_init()与amtpca\_release()配合使用；

* **参数**

无；

* **返回值**

无；

* **流程**

参见第4章流程定义中的4.1.1初始化及释放流程；

* **范例**

参见2.1.2初始化示例；

* + 1. 消息发送
* **定义**

int amtpca\_sendcmd(uint32\_t cmd, void \* cmd\_s, bool sync, uint32\_t timeout);

* **功能**

客户端向平台发送信息；

* **描述**

在完成接入库libamtpca.so的初始化后，客户端可以通过amtpca\_sendcmd ()向平台侧发送消息，该方法通过参数定义了同步、异步两种发送模式，同步发送可以设置超时时间；

* **参数**

**cmd：**需要发送的信息类型；该参数定义在AMTP\_CMD\_ENUM枚举类中，具体定义见第3章中3.1消息类型定义；

**cmd\_s：**待发送消息的消息结构体指针，该消息结构体的定义参见第3章中3.2客户端消息结构体定义；

**sync：**bool值，true表示本次发送采用同步发送，超时时间由timeout参数输入，此时本次发送会在收到消息队列ack后返回（消息队列的ack与应用层的response是两类消息，消息队列的ack是底层传输消息，应用层不可见）；false表示异步发送，消息送入消息队列缓冲区后立即返回。为保证传输可靠性，建议使用同步发送；

**timeout：**大于0的整型数，设置同步发送信息的超时时间；

* **返回值**

返回整型数，如果amtpca\_sendcmd () 函数执行成功，会返回0。否则返回小于0的整型数；

LIB\_AMTPA\_SUCCESS(0)：成功；

LIB\_AMTPA\_NO\_WORKING (-102)：接入库未进行初始化；

LIB\_AMTPA\_PARA\_ERROR (-103)：参数错误或非法；

LIB\_AMTPA\_TIMEOUT (-106)：发送超时错误；

LIB\_AMTPA\_UNKNOWN\_MSG (-113)：待发送信息未定义；

LIB\_JWUMQ\_SEND\_DATA\_FAILD (-8)：信息发送失败；

* **流程**

参见第4章流程定义中的4.1.2客户端的发送与接收；

* **范例**

using LIB\_AMTPCA\_VERSION = char\* (\*)();

using LIB\_AMTPCA\_INIT = int (\*)(const char \* , const char \* , LIBAMTPCA\_CMD\_RECV\_CBFUN );

using LIB\_AMTPCA\_SENDCMD = int (\*)(uint32\_t, void \* , bool , uint32\_t );

using LIB\_AMTPSA\_WAITFORCMD = int (\*)(uint32\_t, void \*, int);

using LIB\_AMTPCA\_RELEASE = void (\*)();

LIB\_AMTPCA\_VERSION lib\_version ;

LIB\_AMTPCA\_INIT lib\_init;

LIB\_AMTPCA\_SENDCMD lib\_sendcmd;

LIB\_AMTPSA\_WAITFORCMD lib\_waitforcmd;

LIB\_AMTPCA\_RELEASE lib\_release;

void \*handle;

lib\_version = NULL;

handle = dlopen("libamtpca.so", RTLD\_NOW);

if (handle == NULL)

{

fprintf(stderr, "Failed to open libaray libamtpca.so error:%s\n", dlerror());

return ;

}

lib\_version = (LIB\_AMTPCA\_VERSION)dlsym(handle, "amtpca\_version");

lib\_init = (LIB\_AMTPCA\_INIT)dlsym(handle, "amtpca\_init");

lib\_sendcmd = (LIB\_AMTPCA\_SENDCMD)dlsym(handle, "amtpca\_sendcmd");

lib\_waitforcmd = (LIB\_AMTPSA\_WAITFORCMD)dlsym(handle, "amtpca\_waitforcmd");

lib\_release = (LIB\_AMTPCA\_RELEASE)dlsym(handle, "amtpca\_release");

int result = lib\_init("dev.license", NULL/\*"/tmp/log"\*/, std::bind(&LibAmtpcaTest::LibamtpcaCmdRecvCbfun, this

, std::placeholders::\_1, std::placeholders::\_2, std::placeholders::\_3));

if(result != 0)

{

return;

}

fprintf(stderr, " Version = %s\n", lib\_version());

MODULE\_CONF\_STRU module\_conf\_s;

memset(&module\_conf\_s, 0, sizeof(MODULE\_CONF\_STRU));

module\_conf\_s.module\_type = AMTP\_MODULE\_TYPE\_ENUM::NR;

module\_conf\_s.chip\_manufacturer = AMTP\_CHIP\_MANUFACTURER\_ENUM::Qualcomm;

strcpy(module\_conf\_s.chip\_type, "Snapdragon 865");

strcpy(module\_conf\_s.icd\_version, "96-V3708-1");

strcpy(module\_conf\_s.imei, "355672052457519");

strcpy(module\_conf\_s.phone\_number, "13488828654");

strcpy(module\_conf\_s.network\_operator, "CMCC");

result = lib\_sendcmd(static\_cast<uint32\_t>(AMTP\_CMD\_ENUM::module\_conf), &module\_conf\_s, true, 5000);

fprintf(stderr, "%s:%s module conf report result = %d\n", Time().c\_str(), name.c\_str(), result);

MODULE\_CONF\_RESP\_STRU module\_conf\_resp\_s;

result = lib\_waitforcmd(static\_cast<uint32\_t>(AMTP\_CMD\_ENUM::module\_conf\_resp), (void \*)&module\_conf\_resp\_s, 5000);

fprintf(stderr, "%s:%s -----------------------------wait for module conf report response(%d) result = %d\n", Time().c\_str(), name.c\_str(), result, module\_conf\_resp\_s.result);

……

lib\_release();

dlclose(handle);

* + 1. 数据接收（阻塞方式）
* **定义**

int amtpca\_waitforcmd(uint32\_t cmd, void \* cmd\_s, uint32\_t timeout);

* **功能**

阻塞方式接收指定消息；

* **描述**

通过调用amtpca\_waitforcmd()，可以阻塞方式接收指定消息，是接入库数据接收的两种方式之一。用于在向发送信息后，等待接收平台应用层应答。该方法可以设置超时。在数据接收中amtpca\_waitforcmd()和回调接收可以并存；

注：平台主动下发消息不可用（如配置数据下发、终端设备重启、测试数据查询）

* **参数**

**cmd：**需要接收的信息类型；该参数定义在AMTP\_CMD\_ENUM枚举类中，具体定义见第3章中3.1消息类型定义；

**cmd\_s：**接收到的消息的结构体，该消息结构体的定义参见第3章中3.2消息结构体定义；

**timeout：**大于0的整型数，设置同步发送信息的超时时间；

* **返回值**

返回整型数，如果amtpca\_waitforcmd () 函数执行成功，会返回0。否则返回小于0的整型数；

LIB\_AMTPA\_SUCCESS(0)：成功；

LIB\_AMTPA\_NO\_WORKING (-102)：接入库未进行初始化；

LIB\_AMTPA\_PARA\_ERROR (-103)：参数错误或非法；

LIB\_AMTPA\_TIMEOUT (-106)：等待超时错误；

LIB\_AMTPA\_UNKNOWN\_MSG (-113)：待发送信息未定义；

LIB\_AMTPA\_WAITFORCMD\_ERROR (-115)：不支持需要接收的信息；

* **流程**

参见第4章流程定义中的4.1.2客户端的发送与接收；

* **范例**

参见2.1.4消息发送示例；

* + 1. 数据接收（回调方式）
* **定义**

using LIBAMTPCA\_CMD\_RECV\_CBFUN = std::function<void (AMTP\_CMD\_ENUM cmd, void \* cmd\_s, void\* reserve)>;

* **功能**

回调方式接收消息；

* **描述**

通过回调方式接收平台侧的所有消息，是接入库数据接收的两种方式之一。当阻塞式接收消息与回调方式并存时，接入库先进行回调中断，带中断结束后触发阻塞等待事件，在采用回调接收时，避免在回调函数里进行耗时的处理与计算，特别是界面类处理；

* **参数**

**cmd：**接收到的信息类型；该参数定义在AMTP\_CMD\_ENUM枚举类中，具体定义见第3章中3.1消息类型定义；

**cmd\_s：**接收到的消息的结构体指针，该消息结构体的定义参见第3章中3.2消息结构体定义；

**reserve：**预留；

* **返回值**

无；

* **流程**

参见第4章流程定义中的4.1.2客户端的发送与接收；

* **范例**

using LIB\_AMTPCA\_VERSION = char\* (\*)();

using LIB\_AMTPCA\_INIT = int (\*)(const char \* , const char \* , LIBAMTPCA\_CMD\_RECV\_CBFUN );

using LIB\_AMTPCA\_SENDCMD = int (\*)(uint32\_t, void \* , bool , uint32\_t );

using LIB\_AMTPSA\_WAITFORCMD = int (\*)(uint32\_t, void \*, int);

using LIB\_AMTPCA\_RELEASE = void (\*)();

using LIBAMTPCA\_CMD\_RECV\_CBFUN = std::function<void (AMTP\_CMD\_ENUM cmd, void \* cmd\_s, void\* reserve)>;

LIB\_AMTPCA\_VERSION lib\_version ;

LIB\_AMTPCA\_INIT lib\_init;

LIB\_AMTPCA\_SENDCMD lib\_sendcmd;

LIB\_AMTPSA\_WAITFORCMD lib\_waitforcmd;

LIB\_AMTPCA\_RELEASE lib\_release;

void \*handle;

lib\_version = NULL;

handle = dlopen("libamtpca.so", RTLD\_NOW);

if (handle == NULL)

{

fprintf(stderr, "Failed to open libaray libamtpca.so error:%s\n", dlerror());

return ;

}

lib\_version = (LIB\_AMTPCA\_VERSION)dlsym(handle, "amtpca\_version");

lib\_init = (LIB\_AMTPCA\_INIT)dlsym(handle, "amtpca\_init");

lib\_sendcmd = (LIB\_AMTPCA\_SENDCMD)dlsym(handle, "amtpca\_sendcmd");

lib\_waitforcmd = (LIB\_AMTPSA\_WAITFORCMD)dlsym(handle, "amtpca\_waitforcmd");

lib\_release = (LIB\_AMTPCA\_RELEASE)dlsym(handle, "amtpca\_release");

int result = lib\_init("dev.license", NULL/\*"/tmp/log"\*/, std::bind(&LibAmtpcaTest::LibamtpcaCmdRecvCbfun, this

, std::placeholders::\_1, std::placeholders::\_2, std::placeholders::\_3));

if(result != 0)

{

return;

}

fprintf(stderr, " Version = %s\n", lib\_version());

MODULE\_CONF\_STRU module\_conf\_s;

memset(&module\_conf\_s, 0, sizeof(MODULE\_CONF\_STRU));

module\_conf\_s.module\_type = AMTP\_MODULE\_TYPE\_ENUM::NR;

module\_conf\_s.chip\_manufacturer = AMTP\_CHIP\_MANUFACTURER\_ENUM::Qualcomm;

strcpy(module\_conf\_s.chip\_type, "Snapdragon 865");

strcpy(module\_conf\_s.icd\_version, "96-V3708-1");

strcpy(module\_conf\_s.imei, "355672052457519");

strcpy(module\_conf\_s.phone\_number, "13488828654");

strcpy(module\_conf\_s.network\_operator, "CMCC");

result = lib\_sendcmd(static\_cast<uint32\_t>(AMTP\_CMD\_ENUM::module\_conf), &module\_conf\_s, true, 5000);

fprintf(stderr, "%s:%s module conf report result = %d\n", Time().c\_str(), name.c\_str(), result);

MODULE\_CONF\_RESP\_STRU module\_conf\_resp\_s;

result = lib\_waitforcmd(static\_cast<uint32\_t>(AMTP\_CMD\_ENUM::module\_conf\_resp), (void \*)&module\_conf\_resp\_s, 5000);

fprintf(stderr, "%s:%s -----------------------------wait for module conf report response(%d) result = %d\n", Time().c\_str(), name.c\_str(), result, module\_conf\_resp\_s.result);

……

lib\_release();

dlclose(handle);

……..

void LibAmtpcaTest::LibamtpcaCmdRecvCbfun(AMTP\_CMD\_ENUM cmd, void \* cmd\_s, void\* reserve)

{

if(cmd == AMTP\_CMD\_ENUM::login\_resp)

{

LOGIN\_RESP\_STRU resp\_s;

memcpy(&resp\_s, cmd\_s, sizeof(LOGIN\_RESP\_STRU));

fprintf(stderr, "%s:%s Callback recv login response, manufacturer = %s, result = %d\n"

, Time().c\_str(), proc\_id, resp\_s.manufacturer, resp\_s.result);

}

if(cmd == AMTP\_CMD\_ENUM::module\_conf\_resp)

{

MODULE\_CONF\_RESP\_STRU resp\_s;

memcpy(&resp\_s, cmd\_s, sizeof(MODULE\_CONF\_RESP\_STRU));

fprintf(stderr, "%s:%s Callback recv module conf report response, result = %d\n"

, Time().c\_str(), proc\_id, resp\_s.result);

}

……..

}

* 1. 平台侧接口说明
     1. 接入库版本
* **定义**

const char \* amtpsa\_version();

* **功能**

获取加载的接入库版本；

* **描述**

远程管理单元加载libamtpsa.so后，通过调用amtpsa\_version()获取libamtpsa.so的版本，版本信息以字符串方式返回，例如“libamtpsa version 1.0.2.20200107”，此方法调用非必须；

* **参数**

无；

* **返回值**

返回字符串，SDK的版本版本信息；

* **流程**

参见第4章流程定义中的4.1.1初始化及释放流程；

* **范例**

using LIB\_AMTPSA\_VERSION = char\* (\*)();

LIB\_AMTPSA\_VERSION lib\_version;

void \*handle;

lib\_version = NULL;

handle = dlopen("libamtpsa.so", RTLD\_NOW);

if (handle == NULL)

{

fprintf(stderr, "Failed to open libaray libamtpsa.so error:%s\n", dlerror());

return ;

}

ib\_version = (LIB\_AMTPSA\_VERSION)dlsym(handle, "amtpsa\_version");

fprintf(stderr, " Version = %s\n", lib\_version());

……

dlclose(handle);

* + 1. 获取接入库句柄
* **定义**

void \* amtpsa\_load();

* **功能**

获取加载的接入库的句柄；

* **描述**

远程管理单元加载libamtpsa.so后，通过调用amtpsa\_load()获取libamtpsa.so的句柄，用于支持so在进程内的多线程重入；

* **参数**

无；

* **返回值**

返回void\*类型的句柄，该句柄标识消息队列的接入，消息队列收发时需要该句柄，在收发时使用不同的句柄则表示使用不同的消息队列；

* **流程**

参见第4章流程定义中的4.1.1初始化及释放流程；

* **范例**

using LIB\_AMTPSA\_VERSION = char\* (\*)();

using LIB\_AMTPSA\_LOAD = void\* (\*)();

LIB\_AMTPSA\_VERSION lib\_version;

LIB\_AMTPSA\_LOAD lib\_load;

void \*handle;

void \* sa\_handle;

lib\_version = NULL;

handle = dlopen("libamtpsa.so", RTLD\_NOW);

if (handle == NULL)

{

fprintf(stderr, "Failed to open libaray libamtpsa.so error:%s\n", dlerror());

return ;

}

ib\_version = (LIB\_AMTPSA\_VERSION)dlsym(handle, "amtpsa\_version");

lib\_load = (LIB\_AMTPSA\_LOAD)dlsym(so, "amtpsa\_load");

fprintf(stderr, " Version = %s\n", lib\_version());

sa\_handle = lib\_load();

……

dlclose(handle);

* + 1. 连接消息队列
* **定义**

int amtpsa\_connection(void \* handle, const char \* mq\_id, char \* cmd\_address, char \* data\_address);

* **功能**

初始化客户端接入库；

* **描述**

远程管理单元libamtpsa.so后，需要调用amtpsa\_connection ()完成对消息队列的连接，只有在连接消息队列成功后，接入库才能完成资源分配及与客户端的通道建立，接入库其他接口方法才有效（除amtpsa\_version()）。libamtpsa.so加载后只需要进行一次消息队列连接（与客户端的初始化方法类似），amtpsa\_connection ()与amtpsa\_release()配合使用；

* **参数**

**handle：**通过amtpsa\_load()获取的句柄，标识对不同消息队列的使用；

**mq\_id**：要接入的平台侧消息队列的ID，远程管理单元如采用分布式部署，则接入的消息队列ID唯一；

**cmd\_address**：需要连接的命令通道地址，如"tcp://127.0.0.1:5581"；

**data\_address**：需要连接的数据通道地址，如"tcp://127.0.0.1:5580"；

注：当前数据通道只限于传输测试文件数据，数据通道与命令通道不通用；

* **返回值**

返回整型数，如果amtpsa\_connection() 函数执行成功，会返回0。否则返回小于0的整型数；

LIB\_AMTPA\_SUCCESS(0)：成功；

LIB\_AMTPA\_IS\_WORKING (-101)：已经进行了初始化；

LIB\_AMTPA\_PARA\_ERROR (-103)：参数错误或非法；

LIB\_AMTPA\_OPEN\_LIB\_FAILED(-104)：接入库资源加载错误；

LIB\_JWUMQ\_SOCKET\_FAILD(-6)：消息队列socket错误；

LIB\_JWUMQ\_CONNECT\_FAILD(-7)：消息队列连接错误；

* **流程**

参见第4章流程定义中的4.1.1初始化及释放流程；

* **范例**

using LIB\_AMTPSA\_VERSION = char\* (\*)();

using LIB\_AMTPSA\_LOAD = void\* (\*)();

using LIB\_AMTPSA\_CONNECTION = int (\*)(const char \* , const char \*, const char \* );

using LIB\_AMTPSA\_RELEASE = void (\*)();

LIB\_AMTPSA\_VERSION lib\_version ;

LIB\_AMTPSA\_LOAD lib\_load;

LIB\_AMTPSA\_CONNECTION lib\_connection;

LIB\_AMTPSA\_RELEASE lib\_release;

void \*handle;

void \* sa\_handle;

lib\_version = NULL;

handle = dlopen("libamtpsa.so", RTLD\_NOW);

if (handle == NULL)

{

fprintf(stderr, "Failed to open libaray libamtpsa.so error:%s\n", dlerror());

return ;

}

lib\_version = (LIB\_AMTPSA\_VERSION)dlsym(handle, "amtpsa\_version");

lib\_load = (LIB\_AMTPSA\_LOAD)dlsym(so, "amtpsa\_load");

lib\_connection = (LIB\_AMTPSA\_CONNECTION)dlsym(handle, "amtpsa\_connection");

lib\_release = (LIB\_AMTPSA\_RELEASE)dlsym(handle, "amtpsa\_release");

fprintf(stderr, " Version = %s\n", lib\_version());

sa\_handle = lib\_load();

int reslut = lib\_connection(sa\_handle, name.c\_str(), cmd\_address, data\_address);

fprintf(stderr, " lib\_amtpsa\_connection(%d)!\n", name.c\_str(), reslut);

if(result != 0)

{

return;

}

……

lib\_release(sa\_handle);

dlclose(handle);

* + 1. 释放队列
* **定义**

void amtpsa\_release(void \* handle);

* **功能**

释放接入库资源；

* **描述**

远程管理单元加载libamtpsa.so后，需要调用amtpsa\_connection ()完成对消息队列的接入。在完成客户端业务功能后，需要调用amtpsa\_release()完成接入库的清理，避免内存泄露等问题。libamtpca.so加载后只需要进行一次连接和释放，amtpsa\_connection()与amtpca\_release()配合使用；

* **参数**

**handle：**通过amtpsa\_load()获取的句柄，标识对不同消息队列的使用；

* **返回值**

无；

* **流程**

参见第4章流程定义中的4.1.1初始化及释放流程；

* **范例**

参见2.2.3连接消息队列示例；

* + 1. 消息发送
* **定义**

int amtpsa\_sendmsg(void \* handle, uint32\_t cmd, char \* des\_id, void \* data, int data\_len);

* **功能**

远程管理单元向客户端发送信息；

* **描述**

远程管理单元在完成消息队列连接后，可以通过amtpsa\_sendmsg()向客户端发送消息，该方法为异步发送，考虑效率问题，接入库只向远程管理单元提供异步发送；

* **参数**

**handle：**通过amtpsa\_load()获取的句柄，标识对不同消息队列的使用；

**cmd：**需要发送的信息类型；该参数定义在AMTP\_CMD\_ENUM枚举类中，具体定义见第3章中3.1消息类型定义；

**des\_id**：发送目的地的消息队列ID字符串，该ID在向客户端提供的license中已经分配且全网唯一。这个ID可以从平台数据库或客户发起的请求消息中获取；

**data：**待发送消息的数据buffer的首地址；这里的buffer存放的是远程管理单元要发送的消息经过protocolbuf序列化后的数据，protocolbuf消息定义参见该消息结构体的定义参见第3章中3.3Protocolbuf消息定义；

**data\_len：**整型值，参数data的数据长度；

* **返回值**

返回整型数，如果amtpca\_sendcmd () 函数执行成功，会返回0。否则返回小于0的整型数；

LIB\_AMTPA\_SUCCESS(0)：成功；

LIB\_AMTPA\_NO\_WORKING(-102)：接入库未进行连接；

LIB\_AMTPA\_PARA\_ERROR(-103)：接入库参数错误或非法；

LIB\_JWUMQ\_PARA\_ABNORMAL(-3)：消息队列参数错误；

LIB\_JWUMQ\_SEND\_ID\_FAILD(-9)：ID信息发送失败；

LIB\_JWUMQ\_SEND\_DATA\_FAILD (-8)：data信息发送失败；

* **流程**

参见第4章流程定义中的4.1.3远程管理单元的发送与接收；

* **范例**

using LIB\_AMTPSA\_VERSION = char\* (\*)();

using LIB\_AMTPSA\_LOAD = void\* (\*)();

using LIB\_AMTPSA\_CONNECTION = int (\*)(const char \* , const char \*, const char \* );

using LIB\_AMTPSA\_SETLIMIT = int (\*)(int);

using LIB\_AMTPSA\_SENDMSG = int (\*)(uint32\_t, const char \* , void \* , int );

using LIB\_AMTPSA\_READMSG = int (\*)(char \* , void \* , int );

using LIB\_AMTPSA\_WAITFORMSG = int (\*)(int \*, int);

using LIB\_AMTPSA\_RELEASE = void (\*)();

LIB\_AMTPSA\_VERSION lib\_version ;

LIB\_AMTPSA\_LOAD lib\_load;

LIB\_AMTPSA\_CONNECTION lib\_connection;

LIB\_AMTPSA\_SETLIMIT lib\_setlimit;

LIB\_AMTPSA\_SENDMSG lib\_sendmsg;

LIB\_AMTPSA\_READMSG lib\_readmsg ;

LIB\_AMTPSA\_WAITFORMSG lib\_waitformsg;

LIB\_AMTPSA\_RELEASE lib\_release;

lib\_version = (LIB\_AMTPSA\_VERSION)dlsym(handle, "amtpsa\_version");

lib\_load = (LIB\_AMTPSA\_LOAD)dlsym(so, "amtpsa\_load");

lib\_connection = (LIB\_AMTPSA\_CONNECTION)dlsym(handle, "amtpsa\_connection");

lib\_setlimit = (LIB\_AMTPSA\_SETLIMIT)dlsym(handle, "amtpsa\_setlimit");

lib\_sendmsg = (LIB\_AMTPSA\_SENDMSG)dlsym(handle, "amtpsa\_sendmsg");

lib\_readmsg = (LIB\_AMTPSA\_READMSG)dlsym(handle, "amtpsa\_readmsg");

lib\_waitformsg = (LIB\_AMTPSA\_WAITFORMSG)dlsym(handle, "amtpsa\_waitformsg");

lib\_release = (LIB\_AMTPSA\_RELEASE)dlsym(handle, "amtpsa\_release");

fprintf(stderr, " Version = %s\n", lib\_version());

sa\_handle = lib\_load();

char\* cmd\_address = (char\*)"tcp://127.0.0.1:5581";

char\* data\_address = (char\*)"tcp://127.0.0.1:5580";

int reslut = lib\_connection(sa\_handle, name.c\_str(), cmd\_address, data\_address);

if(reslut != LIB\_AMTPA\_SUCCESS)

{

return ;

}

while(true)

{

int msg\_len = 0;

int reslut = lib\_waitformsg(sa\_handle, &msg\_len, 10000);

if(reslut > 0 && msg\_len > 0)

{

fprintf(stderr, " amtpsa has a msg(len=%d), remain= %d!\n", msg\_len, reslut);

unique\_ptr<char[]> recv\_buf(new char[msg\_len]());

char src\_id[MAX\_IDENTITY\_BUF\_SIZE] = {0};

lib\_readmsg(sa\_handle, src\_id, recv\_buf.get(), msg\_len);

amtpap::CmdPrimitive req;

req.ParseFromArray(recv\_buf.get(), msg\_len);

fprintf(stderr, " recv request, CmdPrimitive type = %d, serial\_number = %d, msg\_id = %d, direction = %d\n", req.type(), req.serial\_number(), req.msg\_id(), req.direction());

if(req.msg\_id() == amtpap::MsgID::REPORT\_EVENT\_V1)

{

amtpap::ReportEventV1 req;

req.ParseFromArray(data.c\_str(), len);

fprintf(stderr, " recv event command token = %s, module = %d, event\_code = %d, ts\_usec = %d, ts\_sec = %d\n"

, req.token().c\_str(), req.module(), req.event\_code(), req.ts\_usec(), req.ts\_sec());

memset(token, 0, TOKEN\_BUFFER\_SIZE);

sprintf (token, "-----event-----%s", name.c\_str());

amtpap::ReportEventRespV1 resp;

resp.set\_token(token, strlen(token));

resp.set\_result(0);

size\_t resp\_len = resp.ByteSizeLong();

unique\_ptr<char[]> resp\_buf(new char[resp\_len]());

resp.SerializeToArray(resp\_buf.get(), (int)resp\_len);

amtpap::CmdPrimitive cmdprimitive;

cmdprimitive.set\_type(amtpap::MsgType::PUBLIC);

cmdprimitive.set\_serial\_number(cmdprimitive\_sn++);

cmdprimitive.set\_msg\_id(amtpap::MsgID::REPORT\_EVENT\_RESP\_V1);

cmdprimitive.set\_direction(0);

cmdprimitive.set\_payload\_data(resp\_buf.get(), resp\_len);

size\_t length = cmdprimitive.ByteSizeLong();

unique\_ptr<char[]> send\_buf(new char[length]());

cmdprimitive.SerializeToArray(send\_buf.get(), (int)length);

int result = lib\_sendmsg(sa\_handle, static\_cast<uint32\_t>(AMTP\_CMD\_ENUM::report\_event\_resp), src\_id.c\_str(), send\_buf.get(), length);

fprintf(stderr, " send event response (%d, %d) to %s!\n", (int)length, result, src\_id.c\_str());

}

else if(req.msg\_id() == amtpap::MsgID::ALARM\_V1)

{

}

}

else if(reslut == LIB\_AMTPA\_TIMEOUT)

{

fprintf(stderr, " amtpsa wait for msg tome out(%d)!\n", reslut);

}

else if(reslut == 0)

{

fprintf(stderr, " amtpsa no msg(%d)!\n", reslut);

}

else

{

fprintf(stderr, " amtpsa wait for msg error(%d)!\n", reslut);

}

}

lib\_release(sa\_handle);

* + 1. 数据接收等待（阻塞方式）
* **定义**

int amtpsa\_waitformsg(void \* handle, int \* msg\_len, int timeout);

* **功能**

以阻塞方式获取数据接收状态；

* **描述**

平台侧的消息队列接入库在处理数据接收时，首先会将消息队列接收到的数据送入接入库的数据缓冲，在远程管理单元读取数据时，先通过调用amtpsa\_waitformsg()获取接入库缓冲队列的状态，再通过amtpsa\_readmsg()读取消息数据，当接入库接收缓冲为空时，amtpsa\_waitformsg()阻塞等待直到接收新消息或超时；

* **参数**

**handle：**通过amtpsa\_load()获取的句柄，标识对不同消息队列的使用；

**msg\_len：**参数类型为整型指针，函数返回时，如果当前接入库的缓冲中有待读取的消息数据时，msg\_len等于接入库缓冲中首消息的数据长度。等待超时或无可读数据或，msg\_len则小于0；

**timeout：**大于0的整型数，设置阻塞等待的超时时间；

* **返回值**

整型数，如果接入库中有可读消息时amtpsa\_waitformsg()，返回接入库缓冲中待读取的消息个数，超时或无数据可读取是返回小于0的整型数；

LIB\_AMTPA\_NO\_WORKING (-102)：接入库未进行连接；

LIB\_AMTPA\_PARA\_ERROR (-103)：参数错误或非法；

LIB\_AMTPA\_TIMEOUT (-106)：等待超时错误；

LIB\_AMTPA\_MSG\_IS\_NULL (-108)：无消息可读；

* **流程**

参见第4章流程定义中的4.1.2客户端的发送与接收；

* **范例**

参见2.2.5消息发送示例；

* + 1. 数据读取
* **定义**

int amtpsa\_readmsg(void \* handle, char \* src\_id, void \* data, int data\_len);

* **功能**

读取接入库缓冲的消息数据；

* **描述**

平台侧的消息队列接入库在处理数据接收时，首先会将消息队列接收到的数据送入接入库的数据缓冲，在远程管理单元读取数据时，当通过调用amtpsa\_waitformsg()获取接入库缓冲队列的状态为有待读取数据时，再根据amtpsa\_waitformsg()返回的待接收数据长度，通过amtpsa\_readmsg()读取消息数据；

* **参数**

**handle：**通过amtpsa\_load()获取的句柄，标识对不同消息队列的使用；

**src\_id：**字符串指针，不可为NULL；在amtpsa\_readmsg读取消息成功时，src\_id会被写入被读取消息的发送端的消息队列ID。失败时，不写入。在本消息队列中，远程管理单元与客户端为1🡨--🡪N的关系，故在远程管理单元收发数据时，均需要明确客户端的消息队列ID，在本方法中为src\_id，在发送时为des\_id（参见2.2.4平台侧的消息发送）；

**data：**用于存放读取数据的buffer的首地址指针；amtpsa\_readmsg读取消息成功时，会将消息的数据写入该buffer，失败时不写入。这里读取的数据是客户端发送的经过protocolbuf序列化后的数据（参见2.2.4消息发送），所以远程管理单元需要将读取的数据用protocolbuf反序列化后再进行处理，Protocolbuf消息定义参见该消息结构体的定义参见第3章中3.3Protocolbuf消息定义

**data\_len：**整型数，要读取的读取数据长度；

* **返回值**

整型数，如果读取成功时返回0，否则返回小于0的整数；

LIB\_AMTPA\_SUCCESS(0)：成功

LIB\_AMTPA\_NO\_WORKING (-102)：接入库未进行连接；

LIB\_AMTPA\_PARA\_ERROR (-103)：参数错误或非法；

LIB\_AMTPA\_QUEUE\_EMPTY(-107)：队列为空，无消息可读

LIB\_AMTPA\_MSG\_IS\_NULL (-108)：消息不可用；

LIB\_AMTPA\_MSG\_LEN\_ERROR (-109)：读取长度错误；

* **流程**

参见第4章流程定义中的4.1.3远程管理单元的发送与接收；

* **范例**

参见2.2.5消息发送示例；

1. 接口数据结构定义
   1. 消息类型定义

enum class AMTP\_CMD\_ENUM : uint32\_t

{

reserve = 0x00,

login = 0x00010001,

login\_resp = 0x00020001,

logout = 0x00030001,

logout\_resp = 0x00040001,

config = 0x00050001,

config\_resp = 0x00060001,

report\_event = 0x00070001,//废弃

report\_event\_resp = 0x00080001, //废弃

alarm = 0x00090001, //废弃

alarm\_resp = 0x000A0001, //废弃

alarm\_clear = 0x000B0001, //废弃

alarm\_clear\_resp = 0x000C0001, //废弃

gps\_info = 0x000D0001, //废弃

gps\_info\_resp = 0x000E0001, //废弃

status\_info = 0x000F0001, //废弃

status\_info\_resp = 0x00100001, //废弃

config\_data = 0x00110001,

config\_data\_resp = 0x00120001,

config\_notify = 0x00130001,

config\_notify\_resp = 0x00140001,

upload\_file = 0x00150001,

upload\_file\_resp = 0x00160001,

upload\_eof = 0x00170001,

upload\_eof\_resp = 0x00180001,

restart\_cmd = 0x00190001, //废弃

restart\_cmd\_resp = 0x001A0001, //废弃

query\_data = 0x001B0001,

query\_data\_resp = 0x001C0001,

upload\_file\_data = 0x001D0001,

upload\_file\_data\_resp= 0x001E0001,

module\_conf = 0x001F0001,

module\_conf\_resp = 0x00200001,

};

* 1. 客户端结构体定义

客户端在发送消息时，需要输入消息类型和对应的消息结构体，接收时，接入库会返回接收的消息类型及对应的结构体。消息类型定义见3.1章对应的结构体定义如下：

* + 1. 登录请求
* **结构体定义**

using LOGIN\_STRU = struct login\_stru

{

char box\_id[32];

uint32\_t conf\_version;

};

* **描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **类型** | **取值范围** | **参数说明** |
| box\_id | 字符串 |  | 终端设备ID，字符串以’\0’作为结束符，最大长度32 |
| conf\_version | uint32\_t | >=0 | 设备当前的平台测试计划版本号，第一次使用默认为0. |

* + 1. 登录响应
* **结构体定义**

using LOGIN\_RESP\_STRU = struct login\_resp\_stru

{

uint32\_t conf\_version;

int32\_t result;

};

* **描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **类型** | **取值范围** | **参数说明** |
| conf\_version | uint32 | >=0 | 该终端在平台的最新配置版本 |
| result | int32\_t | 参见3.4.1 | 登录请求处理结果 |

* + 1. 登出请求
* **结构体定义**

无；

注：登出时无信息输入；登出示例：

lib\_sendcmd(static\_cast<uint32\_t>(AMTP\_CMD\_ENUM::logout), NULL, true, 5000);

* **描述**

无；

* + 1. 登出响应
* **结构体定义**

using LOGOUT\_RESP\_STRU = struct logout\_resp\_stru

{

int32\_t result;

};

* **描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **类型** | **取值范围** | **参数说明** |
| result | int32\_t | 参见3.4.1 | 登出请求处理结果 |
|  |  |  |  |

* + 1. 配置更新请求
* **结构体定义**

using CONFIG\_STRU = struct config\_stru

{

uint32\_t version;

};

* **描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **类型** | **取值范围** | **参数说明** |
| version | uint32 | >=0 | 设备当前的平台测试计划版本号，第一次使用默认为0 |
|  |  |  |  |

* + 1. 配置更新响应
* **结构体定义**

using CONFIG\_RESP\_STRU = struct config\_resp\_stru

{

uint32\_t version;

int32\_t result;

bool update;//true:need to update

int32\_t packet\_count;//if update = false packet = 0;else .....

char md5[32];//if update = false md5 is 0;else .....

};

* **描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **类型** | **取值范围** | **参数说明** |
| version | uint32 | >=0 | 平台最新的测试计划版本号 |
| result | Int32 | 参见3.4.1 | 登录更新处理结果 |
| update | bool | True，false | 终端设备是否需要更新 |
| packet\_count | Int32 | >0 | Update=true时有效，平台下发的配置数据总包数 |
| md5 | string |  | Update=true时有效，下发配置文件的MD5值 |

* + 1. 配置数据下发请求
* **结构体定义**

using CONFIG\_DATA\_STRU = struct config\_data\_stru

{

uint32\_t version;

int32\_t packet\_count;

int32\_t packet\_no;

uint32\_t data\_length;

char \*config\_data;

};

* **描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **类型** | **取值范围** | **参数说明** |
| version | uint32 | >0 | 下发配置版本 |
| packet\_count | int32 | >0 | 下发数据总包数 |
| packet\_no | int32 | >0 | 当前包序号，从1开始 |
| data\_length | uint32\_t | >0 | 此包数据长度 |
| config\_data | Char\* |  | 配置文件内容，二进制流 |

* + 1. 配置数据下发响应
* **结构体定义**

using CONFIG\_DATA\_RESP\_STRU = struct config\_data\_resp\_stru

{

uint32\_t version;

int32\_t packet\_no;

int32\_t result;

};

* **描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **类型** | **取值范围** | **参数说明** |
| version | uint32 | >0 | 下发配置版本 |
| packet\_no | int32 | >0 | 当前包序号，从1开始 |
| result | Int32 | 参见3.4.1 | 配置数据下发请求处理结果 |

* + 1. 配置更新通知请求
* **结构体定义**

using CONFIG\_NOTIFY\_STRU = struct config\_notify\_stru

{

uint32\_t version;

int32\_t packet\_count;

char md5[32];

};

* **描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **类型** | **取值范围** | **参数说明** |
| version | uint32 | >0 | 下发配置版本 |
| packet\_count | int32 | >0 | 当前包序号，从1开始 |
| md5 | string |  | 下发配置文件的MD5值 |
|  |  |  |  |

* + 1. 配置更新通知响应
* **结构体定义**

using CONFIG\_NOTIFY\_RESP\_STRU = struct config\_notify\_resp\_stru

{

int32\_t result;

};

* **描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **类型** | **取值范围** | **参数说明** |
| result | Int32 | 参见3.4.1 | 配置更新请求处理结果 |
|  |  |  |  |

* + 1. 事件上报请求（废弃）
* **结构体定义**

using REPORT\_EVENT\_STRU = struct report\_event\_stru

{

uint32\_t ts\_sec;

uint32\_t ts\_usec;

uint32\_t module;

uint32\_t event\_code;

};

* **描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **类型** | **取值范围** | **参数说明** |
| ts\_sec | uint32 |  | 事件生成时间，1970/1/1开始的秒数 |
| ts\_usec | uint32 |  | 事件生成时间，微秒数 |
| module | uint32 | >=0 | 事件生成模块号，0代表终端设备 |
| event\_code | uint32 |  | 事件ID |

* + 1. 事件上报响应（废弃）
* **结构体定义：**

using REPORT\_EVENT\_RESP\_STRU = struct report\_event\_resp\_stru

{

int32\_t result;

};

* **描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **类型** | **取值范围** | **参数说明** |
| result | Int32 | 参见3.4.1 | 事件上报请求处理结果 |

* + 1. 告警请求（废弃）
* **结构体定义**

using ALARM\_STRU = struct alarm\_stru

{

uint32\_t module;

uint32\_t seq;

uint32\_t alarm\_code;

uint32\_t reason\_code;

uint32\_t ts\_sec;

uint32\_t ts\_usec;

};

* **描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **类型** | **取值范围** | **参数说明** |
| module | uint32 | >=0 | 告警生成模块号，0代表终端设备 |
| seq | uint32 |  | 终端设备告警序号，从1累加 |
| alarm\_code | uint32 |  | 告警ID |
| reason\_code | uint32 |  | 告警原因ID |
| ts\_sec | uint32 |  | 告警生成时间，1970/1/1开始的秒数 |
| ts\_usec | uint32 |  | 告警生成时间，微秒数 |

* + 1. 告警响应（废弃）
* **结构体定义**

using ALARM\_RESP\_STRU = struct alarm\_resp\_stru

{

int32\_t result;

};

* **描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **类型** | **取值范围** | **参数说明** |
| result | Int32 | 参见3.4.1 | 告警请求处理结果 |

* + 1. 告警消除请求（废弃）
* **结构体定义**

using ALARM\_CLEAR\_STRU = struct alarm\_clear\_stru

{

uint32\_t module;

uint32\_t seq;

uint32\_t alarm\_code;

uint32\_t ts\_sec;

uint32\_t ts\_usec;

};

* **描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **类型** | **取值范围** | **参数说明** |
| module | uint32 | >=0 | 消除告警的模块号，0代表终端设备 |
| seq | uint32 |  | 对应的告警序号，从1累加 |
| alarm\_code | uint32 |  | 告警ID |
| ts\_sec | uint32 |  | 消除告警时间，1970/1/1开始的秒数 |
| ts\_usec | uint32 |  | 消除告警时间，微秒数 |

* + 1. 告警消除响应（废弃）
* **结构体定义**

using ALARM\_CLEAR\_RESP\_STRU = struct alarm\_clear\_resp\_stru

{

int32\_t result;

};

* **描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **类型** | **取值范围** | **参数说明** |
| result | Int32 | 参见3.4.1 | 告警消除处理结果 |

* + 1. GPS上报请求（废弃）
* **结构体定义**

using GPS\_INFO\_STRU = struct gps\_info\_stru

{

float longitude;

float latitude;

float altitude;

float speed;

float heading;

uint32\_t ts\_sec;

uint32\_t ts\_usec;

};

* **描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **类型** | **取值范围** | **参数说明** |
| longitude | float |  | 经度 |
| latitude | float |  | 纬度 |
| altitude | float |  | 高度，单位：米 |
| speed | float |  | 速度，单位：km/h |
| heading | float | [0,360) | 移动方向，正北逆时针取值，单位：度 |
| ts\_sec | uint32 |  | GPS上报时间，1970/1/1开始的秒数 |
| ts\_usec | uint32 |  | GPS上报时间，微秒数 |

* + 1. GPS上报响应（废弃）
* **结构体定义**

using GPS\_INFO\_RESP\_STRU = struct gps\_info\_resp\_stru

{

int32\_t result;

};

* **描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **类型** | **取值范围** | **参数说明** |
| result | Int32 | 参见3.4.1 | GPS上报处理结果 |

* + 1. 状态上报请求（废弃）
* **结构体定义**

using STATUS\_INFO\_STRU = struct status\_info\_stru

{

uint32\_t power\_mode;

uint32\_t files\_letf;

float temperature;

float disk\_space;

uint32\_t ts\_sec;

uint32\_t ts\_usec;

};

* **描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **类型** | **取值范围** | **参数说明** |
| power\_mode | uint32 | [0,1] | 供电方式，0：内置电池供电，1：外接电源供电 |
| files\_letf | uint32 |  | 剩余未上传文件数 |
| temperature | float |  | 前端设备工作温度，单位：℃ |
| disk\_space | float |  | 剩余存储空间，单位：MB |
| ts\_sec | uint32 |  | 上报时间，1970/1/1开始的秒数 |
| ts\_usec | uint32 |  | 上报时间，微秒数 |

* + 1. 状态上报响应（废弃）
* **结构体定义**

using STATUS\_INFO\_RESP\_STRU = struct status\_info\_resp\_stru

{

int32\_t result;

};

* **描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **类型** | **取值范围** | **参数说明** |
| result | Int32 | 参见3.4.1 | 状态上报处理结果 |

* + 1. 上传文件请求
* **结构体定义**

using UPLOAD\_FILE\_STRU = struct upload\_file\_stru

{

char file\_name[FILENAME\_LENGTH];// FILENAME\_LENGTH = 64

bool retransmit;

uint32\_t module;

};

* **描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **类型** | **取值范围** | **参数说明** |
| file\_name | Char |  | 上传文件名 |
| retransmit | Bool |  | 重传状态，true：重传，false：新文件 |
| module | uint32 | >=0 | 模块号 |

* + 1. 上传文件请求响应
* **结构体定义**

using UPLOAD\_FILE\_RESP\_STRU = struct upload\_file\_resp\_stru

{

char file\_name[FILENAME\_LENGTH];

uint32\_t file\_id;

uint32\_t file\_size;

uint32\_t module;

int32\_t result;

};

* **描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **类型** | **取值范围** | **参数说明** |
| file\_name | Char |  | 待上传文件名 |
| result | Int32 | 参见3.4.1 | 文件上传请求处理结果 |
| module | uint32 | >=0 | 模块号 |
| file\_id | uint32 |  | 平台分配的上传文件ID，后续文件上传使用 |
| file\_size | uint32 |  | 该文件已经上传的文件大小，新文件时为0 |

* + 1. 文件数据上传请求
* **结构体定义**

using UPLOAD\_FILE\_DATA\_STRU = struct upload\_file\_data\_stru

{

uint32\_t file\_id;

uint32\_t packet\_no;

uint32\_t data\_len;

unsigned char \* data;

};

* **描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **类型** | **取值范围** | **参数说明** |
| file\_id | uint32 |  | 平台分配的上传文件ID |
| packet\_no | uint32 | >0 | 文件数据上传包序号，从1累加 |
| data\_len | uint32 | >0 | 上传文件数据长度，最大64K |
| data | Bytes |  | 上传文件数据 |

* + 1. 文件数据上传响应
* **结构体定义**

using UPLOAD\_FILE\_DATA\_RESP\_STRU = struct upload\_file\_data\_resp\_stru

{

uint32\_t file\_id;

uint32\_t packet\_no;

int32\_t result;

};

* **描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **类型** | **取值范围** | **参数说明** |
| result | Int32 | 参见3.4.1 | 文件数据上传处理结果 |
| file\_id | uint32 |  | 平台分配的上传文件ID |
| packet\_no | uint32 | >0 | 文件数据上传请求的包序号 |

* + 1. 文件上传结束请求
* **结构体定义**

using UPLOAD\_EOF\_STRU = struct upload\_eof\_stru

{

char file\_name[FILENAME\_LENGTH];

uint32\_t file\_id;

uint32\_t packet\_count;

uint32\_t total\_size;

uint32\_t module;

char md5[32];

};

* **描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **类型** | **取值范围** | **参数说明** |
| file\_name | char |  | 上传结束的文件名 |
| file\_id | uint32 | >0 | 平台分配的上传文件ID |
| packet\_count | uint32 | >0 | 上传文件的总包数 |
| total\_size | uint32 | >0 | 上传文件的字节数 |
| module | uint32 |  | 上传文件所属模块 |
| md5 | char |  | 上传文件的MD5值 |

* + 1. 上传文件结束响应
* **结构体定义**

using UPLOAD\_EOF\_RESP\_STRU = struct upload\_eof\_resp\_stru

{

char file\_name[FILENAME\_LENGTH];

uint32\_t file\_id;

uint32\_t packet\_count;

uint32\_t module;

int32\_t result;

};

* **描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **类型** | **取值范围** | **参数说明** |
| file\_name | char |  | 上传结束的文件名 |
| file\_id | uint32 |  | 平台分配的上传文件ID |
| packet\_count | uint32 |  | 上传文件的总包数 |
| module | uint32 |  | 上传文件所属模块 |
| result | int32 | 参见3.4.1 | 文件上传结束请求处理结果 |

* + 1. 查询文件数据请求
* **结构体定义**

using QUERY\_DATA\_STRU = struct query\_data\_stru

{

char file\_name[FILENAME\_LENGTH];

uint32\_t file\_id;

uint32\_t packet\_no;

};

* **描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **类型** | **取值范围** | **参数说明** |
| file\_name | char |  | 需要查询的文件名 |
| file\_id | uint32 | >0 | 平台分配的文件ID |
| packet\_no | uint32 | >0 | 该文件被查询的包序号 |

* + 1. 查询文件数据响应
* **结构体定义**

using QUERY\_DATA\_RESP\_STRU = struct query\_data\_resp\_stru

{

int32\_t result;

};

* **描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **类型** | **取值范围** | **参数说明** |
| result | Int32 | 参见3.4.1 | 查询文件数据请求的处理结果 |

* + 1. 重启终端请求（废弃）
* **结构体定义**

using RESTART\_CMD\_STRU = struct restart\_cmd\_stru

{

uint32\_t ts\_sec;

uint32\_t ts\_usec;

};

* **描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **类型** | **取值范围** | **参数说明** |
| ts\_sec | uint32 |  | 上报时间，1970/1/1开始的秒数 |
| ts\_usec | uint32 |  | 上报时间，微秒数 |

* + 1. 重启终端响应（废弃）
* **结构体定义**

using RESTART\_CMD\_RESP\_STRU = struct restart\_cmd\_resp\_stru

{

int32\_t result;

};

* **描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **类型** | **取值范围** | **参数说明** |
| result | Int32 | 参见3.4.1 | 重启终端处理结果 |

* + 1. 模块配置上报
* **结构体定义**

enum class AMTP\_MODULE\_TYPE\_ENUM : uint32\_t

{

unknown = 0,

GPS = 1,

LTE = 2,

NR = 3,

SCANNER = 4,

WLAN = 5,

NB = 6,

other = 255,

};

enum class AMTP\_CHIP\_MANUFACTURER\_ENUM : uint32\_t

{

Qualcomm = 0,

Hisilicon = 1,

other = 255,

};

using MODULE\_CONF\_STRU = struct module\_conf\_stru

{

uint32\_t module\_num;

AMTP\_MODULE\_TYPE\_ENUM module\_type;

AMTP\_CHIP\_MANUFACTURER\_ENUM chip\_manufacturer;

char chip\_type[32];

char icd\_version[32];

char imei[32];

char imsi[32];

char module\_system[32];

char module\_factory[32];

char phone\_number[16];

char network\_operator[16];

};

* **描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **类型** | **取值范围** | **参数说明** |
| module\_num | uint32 |  | 模块号 |
| module\_type | enum | 0-255 | 模组网络制式 |
| chip\_manufacturer | enum | 0-255 | 模组芯片厂商 |
| chip\_type | char | 长度<=32字节 | 模组芯片型号 |
| icd\_version | char | 长度<=32字节 | 模组icd版本 |
| imei | char | 长度<=32字节 | IMEI号 |
| imsi | char | 长度<=32字节 | IMSI号 |
| module\_system | char | 长度<=32字节 | 模块对应的操作系统及版本 |
| module\_factory | char | 长度<=32字节 | 模块制造厂商即手机制造商 |
| phone\_number | char | 长度<=16字节 | 手机号 |
| network\_operator | char | 长度<=16字节 | 运营商 |

* + 1. 模块配置上报响应
* **结构体定义**

using MODULE\_CONF\_RESP\_STRU = struct module\_conf\_resp\_stru

{

int32\_t result;

};

* **描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **类型** | **取值范围** | **参数说明** |
| result | Int32 | 参见3.4.1 | 重启终端处理结果 |

* 1. 服务端

* 1. 常量定义
     1. CmdErrorCode定义

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 编码 | 名称 | 含义 | 说明 | 适应指令 |
| 0x00 | Normal | 成功，没有错误 | 如果成功，可以不下发Code | ALL |
| 0x01 | Config error | 配置错误 | 如数据库配置错误等，无法找到数据，无法验证信息，设置不正确等 | ALL |
| 0x02 | Invalid Device ID | 终端ID非法 | 设备终端ID在服务器上找不到 | Login |
| 0x03 | unauthorized | 未授权 | 终端未授权 | ALL |
| 0x04 | Already logon | 已经登录 | 终端已经登录，重复登录了 | Login |
| 0x05 | Not logon | 终端未登录 | 终端必须在Login指令之后才能继续其他的指令 | 除Login外 |
| 0x06 | Unknown command | 未知的指令 | 服务器不支持相关指令，可能是错误指令也可能是服务器版本较老 | ALL |
| 0x07 | No Valid Config | 无有效测试计划 | 没有配置测试计划，或者不需要更新 | Config |
| 0x08 | Invalid Data | 无效数据 | 非法的，无效的，不可解释的数据 | Alarm, GPS, Event, MOS, Status |
| 0x09 | Invalid Packet | 无效数据包 | 数据包或者命令包不符合格式 | ALL |
| 0x0A | File open error | 打开文件失败 | 在Upload指令中打开对应数据文件失败 | Upload |
| 0x0B | File close error | 关闭文件失败 | 在Eof指令中关闭对应数据失败 | Eof |
| 0x0C | Test Config Error | 测试配置错误 | 测试配置非法，不存在，错误，冲突，格式，不对等等 | Config |
| 0x0D | File Already Eof | 文件已上传完毕 | 数据上传请求是后台检测文件正常EOF | Data Upload相关 |
| 0x0E | Server Reject | 服务器主动拒绝 | 客户端上发请求之后，服务器拒绝。 | ALL |
|  |  |  |  |  |

1. 流程定义
   1. 接口库调用流程
      1. 初始化及释放流程：



* + 1. 客户端的发送与接收





* + 1. 远程管理单元的发送与接收



* 1. 业务功能流程
     1. 登入登出



* + 1. 配置更新



* + 1. 事件上报（废弃）



* + 1. 告警消警（废弃）



* + 1. GPS上报（废弃）



* + 1. 状态上报（废弃）



* + 1. 上传测试数据文件



* + 1. 查询测试文件数据



* + 1. 重启终端（废弃）



* + 1. 模块配置上报



1. 裁剪说明

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **裁剪内容** | **裁剪类别** | **裁剪说明** | **其他要求** |
|  |  |  |  |  |
|  |  |  |  |  |