CmdService接口开发规范

# 1 新版PIS软件概述

新版PIS软件最主要的改动是设备的监控机制从老版中的完全集中式改为集中分布式。首先新版PIS软件中的网管软件直接向分线中心发布针对终端设备的操作指令，并查询终端设备的状态，不再象老版那样直接向终端设备发布指令及查询状态；其次播放控制器等终端设备主动向分线中心查询自身的操作指令，并主动发布自身的设备状态，也不再象老版那样被动等待操作指令以及状态查询指令再答复。

新版PIS软件的集中分布式的监控机制相对于老版的完全集中式具有如下几个优点：

(1) 发布指令所需要的时间大大减少，大大提高操作的时效性；

老版PIS软件发布指令是依次向所有的终端设备发布指令，只有一台终端设备发布完毕后才向下一台发布，正常情况下，向一台终端设备发布指令大约耗时50～200ms，但如果终端设备出现故障导致网络无法连接，那么在该台设备耗费的时间将大大延长到2000～6000ms，因此故障设备越多，发布指令的总耗时就越长。而车地无线网络由于网络可靠性远远不如有线网络，出现网络无法连接的可能性大大增加，这是老版PIS发布指令耗时冗长的主要原因。采用多线程技术时能够减少总体发布时间，但对于大规模甚至超大规模的PIS系统来说，仍旧无济于事。

在一个多条线网的终端设备多达1200多台的实际PIS系统案例中，每条线网均建设有车地无线网络，当向所有的终端设备（包括车站和车载）发布一条紧急信息时，老版PIS软件总耗时达到了30分钟左右，显然已经没有任何时效性和可用性，是无法接受的。

新版PIS软件采用集中分布式的监控机制，网管软件发布指令时直接发送到分线中心缓存，然后等待一个固定时间（例如20秒钟），在此期间所有的终端设备去获取指令、执行、反馈结果，网管软件等待时间结束后将指令发布结果显示出来。由于终端设备各自独立的获取指令并执行，因此互不干扰，指令的整体发布时间不会因为存在终端设备故障而增加耗时，也不会因为PIS规模太大增加耗时。

(2) 发布指令及获取状态的可靠性大大增加；

老版PIS软件在发布指令和获取状态时，需要针对每个设备单独建立一个TCP的socket连接，使用完毕后立即关闭。由于TCP/IP协议的规定，当一个TCP的socket连接关闭后，socket资源还需要等待3～6分钟才能释放完毕。而网管软件需要定时的查询所有终端设备的状态并刷新显示，当PIS规模较大时，socket资源将有可能出现耗尽的情况，从而导致网管软件的整体对外通信故障。

另外，由于车地无线网络的可靠性不足，网管软件在连接一台车载设备时，不成功的概率较大，耗时增加后更容易导致socket资源耗尽。

新版PIS软件采用的是完全不同的监控机制，其网管软件只建立有限的几个TCP连接，socket资源永远不会出现耗尽的情况。终端设备在向分线中心查询指令和报告设备状态时，由于采用主动定时的机制，一次不成功还有其他机会，因此整体可靠性要大大提高。

# 2 CmdService

CmdService是新版PIS软件中的命令服务器程序，运行于总OCC的中心服务器和分线OCC的分线服务器上，用于缓存所有的网管指令以及设备反馈的状态数据。要使用CmdService的服务功能，还需要使用相对应的新版网管软件、新版InfoMonitor软件等。

新版网管软件在向播放控制器等终端设备发出操作指令（例如关机、重启、开关屏幕、发布紧急信息等）后，相应的指令不再象旧版一样直接发送给终端设备，而是发到CmdService软件进行缓存，然后由终端设备上的新版InfoMonitor等软件定时获取各自的操作指令，并执行和反馈，同时将自身的设备状态发送到CmdService进行缓存。

新版网管软件获取终端设备的状态时，直接向CmdService查询，后者会将缓存中的所有设备状态数据反馈回新版网管软件。

新版PIS软件中CmdService只在分线服务器上运行CmdService，为本线内的PIS设备提供服务，总OCC的中心服务器上也运行CmdService，为总OCC内的其他设备提供服务。

需要利用CmdService服务的软件有OCCMonitor（运行于总OCC的中心服务器）、SOCCMonitor（运行于分线服务器）、InfoMonitor（运行于车站和车载播放控制器）、SysMonitor（运行于工作站、直播服务器、接口服务器、及其他OCC设备），需要注意的是，这些软件不能同时运行于同一台设备上。另外还有一个软件NetMonitor，只用于监测网络设备(切记)，可安装在任何一台不关机的windows平台的设备上。

CmdService采用TCP连接方式提供服务，默认端口为7650。CmdService支持的并发连接数主要决定于硬件设备的性能规格，现有项目中配置的服务器应该能够满足需求。

# 3 CmdService服务接口

CmdService支持多种命令接口，见如下枚举结构：

enum CMD\_TYPE

{

CMD\_PUBLISH\_TASK = 0, //网管发布操作任务

CMD\_QUERY\_TASK\_RESULT, //网管查询操作任务执行结果

CMD\_QUERY\_STATUS, //网管查询设备状态

CMD\_QUERY\_SINGLE\_STATUS, //网管查询单个设备状态

CMD\_QUERY\_TASK, //设备查询自己的操作任务

CMD\_FEEDBACK\_TASK\_RESULT, //设备反馈操作任务的执行结果

CMD\_PUBLISH\_STATUS, //设备发布自己的状态

//下面4个指令只针对分线PIS接口程序有效

CMD\_SET\_CONFIG, //系统管理软件设置分线配置，预留

CMD\_QUERY\_SETCONFIG\_RESULT, //系统管理软件查询设置任务执行结果，预留

CMD\_GET\_CONFIG, //系统管理软件获取分线设备信息，预留

CMD\_QUERY\_GETCONFIG\_RESULT, //系统管理软件查询获取分线设备执行结果，预留

CMD\_CANCEL\_TASK = 50, //取消操作任务

CMD\_TEST = 80, //测试服务是否正常的指令

};

## 3.1 CMD\_HEAD结构

与CmdService通信的所有数据都以一个命令结构头CMD\_HEAD开始，详见如下：

#define FLAG\_SIZE 4

#define CMD\_FLAG "Cmd

"typedef struct

{

char szHead[FLAG\_SIZE]; //头部标识，固定为"Cmd"

int iDataSize; //本结构以及后续所有数据的字节大小

BYTE buType; //内容类型，由CMD\_TYPE描述

BYTE buAck; //1表示命令的答复，0表示原始命令

BYTE buCheckInterval; //命令服务器设定的监控查询间隔，单位秒

int iNum; //后续TASK\_LIST或者DEVICE\_STATUS结构的数量

union

{ char szTaskID[33]; //针对CMD\_QUERY\_TASK\_RESULT查询命令有效

char szDeviceID[9]; //针对CMD\_QUERY\_TASK查询命令有效

};

} CMD\_HEAD, \* PCMD\_HEAD;

备注：所有的数据结构都是1字节对齐（#pragma pack(1)），切记！

(1) szHead：CMD\_HEAD结构以szHead开始，长度4个字节，固定为“Cmd”，用于定位数据头部；

(2) iDataSize：描述本次数据通信中的所有数据的字节大小，包括CMD\_HEAD结构；

(3) buType：描述通信的命令类型，即上述所说的7种命令接口，由CMD\_TYPE枚举结构描述；

(4) buAck：描述是命令本身，还是命令的答复，1标识命令的答复，0表示原始命令；

(5) buCheckInterval：描述的是CmdService设定的监控查询间隔，在每次通信中都发送给客户端，后者根据该查询间隔定时向CmdService查询指令和汇报状态；

(6) iNum：描述跟随在后面的TASK\_LIST或者DEVICE\_STATUS结构的数量；

(7) szTaskID：用于CMD\_QUERY\_TASK\_RESULT命令，描述需要查询的操作任务ID；

(8) szDeviceID：用于CMD\_QUERY\_TASK命令，描述需要查询操作任务的设备的编号，例如“S0305-01”。

## 3.2 CMD\_PUBLISH\_TASK命令

该命令用于客户端软件（例如网管软件）向CmdService发布操作任务。

该命令以CMD\_HEAD开始，紧跟一个TASK\_SET结构和若干个TASK\_LIST结构，一个TASK\_LIST结构描述一个设备。CMD\_HEAD结构的iNum成员描述设备总数量。

CmdService接收到该指令后，以一个CMD\_HEAD结构作为答复，其中buAck成员设置为1，其他成员与接收到的内容保持不变。

### 3.2.1 TASK\_SET结构

客户端软件发布命令前，需要先设定该结构的所有成员。

typedef struct

{

char szTaskID[33];

\_\_int8 iTaskType;

\_\_int8 iEmType;

\_\_int8 iFullScreen;

\_\_int8 iReplaceInfo;

\_\_int8 iSoundType;

BYTE buLeftVolume;

BYTE buRightVolume;

char szEmergencyInfo[1022];

DATE dtTaskTime;

int iValidTime; //有效时长，单位秒

\_\_int16 iTimeLength;

char szEditorID[17];

BYTE buRightLevel;

//设备列表

CMapStringToPtr \* pMapDevice;

} TASK\_SET, \* PTASK\_SET;

(1) szTaskID[33]：操作任务的ID编号，客户端软件需要自动生成，并确保全局唯一性；

(2) iTaskType：操作任务的类型，由枚举结构MACRO\_TYPE\_INDEX描述，但不支持其中的MT\_POWER\_ON命令；

(3) iEmType：操作任务为发布紧急信息时的方式（紧急文本、紧急版式），由枚举结构NET\_ES\_TYPE描述；

(4) iFullScreen：操作任务为发布紧急信息时的显示方式，1表示全屏显示，0表示区域显示；

(5) iReplaceInfo：操作任务为发布紧急信息时的播出方式，1表示覆盖，0表示跟随；

(6) iSoundType：操作任务为发布紧急信息时的声音处理方式（不处理、关闭声音、设置音量），由枚举结构NET\_ES\_SOUND\_TYPE描述；

(7) buLeftVolume：操作任务为设置音量或者发布紧急信息（同时声音处理方式为调整音量）时的左声道音量，范围0～255；

(8) buRightVolume：操作任务为设置音量或者发布紧急信息（同时声音处理方式为调整音量）时的右声道音量，范围0～255；

(9) szEmergencyInfo[1022]：操作任务为发布紧急信息时的紧急文本或者紧急版式名称；

(10) dtTaskTime：操作任务的发布时间，CmdService接收后会以自身本机时间重新设定该成员；

(11) iValidTime：操作任务的有效时间，单位秒，从CmdService接收到后开始计算；该时间应该大于CmdService设定的查询间隔的2倍；例如如果CmdService设定的查询间隔为10秒，则iValidTime可设为22秒；

(12) iTimeLength：操作任务为发布紧急信息时，紧急状态的持续时间，单位秒，如果为0，表示紧急状态持续到停止命令；

(13) szEditorID[17]：操作任务发布者的ID；

(14) buRightLevel：操作任务发布者的权限级别，0～255，数字越大，级别越高；

(15) pMapDevice：CmdService内部使用。

enum MACRO\_TYPE\_INDEX

{

MT\_POWER\_ON = 0, //设备开机命令 网管软件提供该功能

MT\_POWER\_OFF, //设备关机命令 网管软件提供该功能

MT\_RESET\_DEVICE, //重新启动命令 网管软件提供该功能

MT\_SCREEN\_ON, //设备显示屏开机命令 网管软件提供该功能，紧急管理软件提供该功能

MT\_SCREEN\_OFF, //设备显示屏关机命令 网管软件提供该功能，紧急管理软件提供该功能

MT\_PUBLISH\_INFO, //发布紧急信息命令，即实施紧急状态 紧急管理软件提供该功能

MT\_STOP\_PUBLISH, //取消紧急信息命令，即结束紧急状态 紧急管理软件提供该功能

MT\_START\_RUNNING, //开始运行命令 网管软件提供该功能

MT\_STOP\_RUNNING, //停止运行命令 网管软件提供该功能

MT\_TAKE, //TAKE命令 网管软件提供该功能

MT\_SOUND\_ON, //开启声音命令 网管软件提供该功能，紧急管理软件提供该功能

MT\_SOUND\_OFF, //关闭声音命令 网管软件提供该功能，紧急管理软件提供该功能

MT\_SET\_VOLUME, //设置音量（0～255） 网管软件提供该功能

MT\_LIVE\_ON, //开始直播 网管软件提供该功能

MT\_LIVE\_OFF, //停止直播 网管软件提供该功能

MAX\_MACRO\_TYPE

};

enum NET\_ES\_TYPE

{

NET\_ES\_CONTENT = 0, //采用紧急内容，通过紧急模块实现，szContent维护紧急内容

NET\_ES\_LAYOUT, //采用紧急版式，szContent维护版式文件名

MAX\_NET\_ES\_TYPE

};

enum NET\_ES\_SOUND\_TYPE

{

NET\_ES\_SOUND\_NONE = 0,

NET\_ES\_SOUND\_OFF,

NET\_ES\_SOUND\_SET\_VOLUME,

MAX\_NET\_ES\_SOUND\_TYPE

};

### 3.2.2 TASK\_LIST结构

客户端软件发布命令前，需要先设定该结构的所有成员。

typedef struct

{

char szDeviceCode[9];

\_\_int16 iExecResult;

DATE dtExecTime;

} TASK\_LIST, \* PTASK\_LIST;

(1) szDeviceCode[9]：操作任务发布的对象的设备编号；

(2) iExecResult：客户端软件发布命令前，必须将该成员设定为CMD\_RESULT\_WAIT\_EXEC，表示等待处理；客户端软件获取操作任务的执行结果时，该成员表示对应设备的执行结果，由枚举结构CMD\_RESULT描述；

(3) dtExecTime：客户端软件获取操作任务的执行结果时，该成员表示对应设备的执行时间。

enum CMD\_RESULT

{

CMD\_RESULT\_WAIT\_EXEC = -100, //等待处理

CMD\_RESULT\_DB\_ERROR = -90, //数据库访问错误

CMD\_RESULT\_LOCAL\_ERROR = -10, //本地错误，例如参数非法

CMD\_RESULT\_CONN\_ERROR, //网络故障

CMD\_RESULT\_SERVER\_ERROR, //网络服务未启动，或者服务程序已经死机

CMD\_RESULT\_POWER\_OFF, //设备已关机

CMD\_RESULT\_RESET, //设备在重启

CMD\_RESULT\_PLAY\_DOWN, //播出已停止

CMD\_RESULT\_ERROR = 0, //执行失败，对于状态检测，故障原因由szError描述

CMD\_RESULT\_OK, //执行成功

//错误命令序号

CMD\_RESULT\_NO\_CMD, //没有指定的命令

CMD\_RESULT\_NO\_LOGON, //没有登录，不使用

//登录错误序号

CMD\_RESULT\_LOGON\_FAIL, //错误的用户名或密码，不使用

CMD\_RESULT\_NO\_RIGHT, //没有权限，不使用

//紧急状态或者播表、版式控制错误序号

CMD\_RESULT\_LOW\_RIGHT, //权限低，不能进行紧急状态的控制，或者是当前处于紧急状态，实施其他控制但权限较低

CMD\_RESULT\_LAYOUT\_NOT\_EXIST, //版式不存在

CMD\_RESULT\_LIST\_NOT\_EXIST, //播表不存在，不使用

CMD\_RESULT\_NOT\_EMERGENT, //取消设置失败，因为没有处于紧急状态

//TAKE命令错误序号

CMD\_RESULT\_NO\_PST, //当前没有PST节目，不能执行TAKE命令

//开始运行和停止运行错误序号

CMD\_RESULT\_NO\_LIST, //当前没有播表节目，不能执行命令

CMD\_RESULT\_NO\_LAYOUT, //当前时间段没有版式，不能执行命令

CMD\_ERSULT\_INVALID\_INDEX, //SKIP命令的版式组或者版式节目序号非法，不使用

//没有板卡

CMD\_RESULT\_NO\_BOARD, //不使用

//直播命令的错误序号

CMD\_RESULT\_NO\_LIVE\_MODULE, //没有直播模块

//控制器故障类型

CMD\_RESULT\_BOARD\_ERROR = 50, //板卡故障

CMD\_RESULT\_CL\_ERROR, //接口监听故障

CMD\_RESULT\_WDT\_ERROR, //wdt故障

CMD\_RESULT\_SCREEN\_ERROR, //电源控制器故障

CMD\_RESULT\_UPS\_ERROR, //UPS模块故障

CMD\_RESULT\_HD\_ERROR, //硬盘故障

CMD\_RESULT\_LOW\_SPACE, //硬盘空间不足

CMD\_RESULT\_CPU\_ERROR, //CPU温度过高

CMD\_RESULT\_CPU\_OVERLOAD, //CPU占用率过载

};

## 3.3 CMD\_QUERY\_TASK\_RESULT命令

该命令用于客户端软件（例如网管软件）向CmdService查询操作任务的执行结果。

该命令只有一个CMD\_HEAD结构，其成员szTaskID表示需要查询的操作任务。

CmdService接收到该指令后，以一个CMD\_HEAD结构和多个TASK\_LIST结构作为答复，其中buAck成员设置为1，iNum成员描述TASK\_LIST结构的数量，其他成员与接收到的内容保持不变。

TASK\_LIST结构描述设备对于指定操作任务的执行结果。

## 3.4 CMD\_QUERY\_STATUS命令

该命令用于客户端软件（例如网管软件）向CmdService查询所有设备的状态。

该命令只有一个CMD\_HEAD结构。

CmdService接收到该指令后，以一个CMD\_HEAD结构和多个DEVICE\_STATUS结构作为答复，其中buAck成员设置为1，iNum成员描述DEVICE\_STATUS结构的数量(即设备的数量)，其他成员与接收到的内容保持不变。

### 3.4.1 DEVICE\_STATUS结构

DEVICE\_STATUS结构引用了CHECK\_STATUS\_PARAM、NET\_ES\_PARAM、PLAY\_CONTENT结构，详见如下：

typedef struct

{

char szDeviceCode[9];

\_\_int8 iReserve1; //预留

\_\_int8 iReserve2; //预留

\_\_int16 iReserve; //预留

union

{ DATE dtUpdateTime;

int iTimeOffset;

};

CHECK\_STATUS\_PARAM csParam; //状态检测参数结构

NET\_ES\_PARAM emParam; //紧急状态控制参数结构

PLAY\_CONTENT PlayContent; //播出内容结构

} DEVICE\_STATUS, \* PDEVICE\_STATUS;

(1) szDeviceCode[9]：DEVICE\_STATUS结构数据对应的设备的编号；

(2) iCheckResult：设备状态的检测结果，由枚举结构CMD\_RESULT描述；

iAlarmLevel：设备故障时报警等级，0～4。设备报警调整为5个等级：1绿、2黄、3橙、4红、5深红。绿表示设备完全正常；黄表示出现轻微故障，不影响正常使用；橙表示出现中等故障，不影响使用，但需要尽快修复故障；红表示出现严重故障，不能正常使用，必须马上修复；深红表示出现严重故障，不能正常使用，同时影响到其他设备的正常工作，必须马上修复；

(3) dtUpdateTime：状态的更新时间，CmdService接收到一个设备的状态数据后，会以本地时间设定该成员，此时iTimeOffset成员无效；

(4) iTimeOffset：客户端软件查询状态数据时，CmdService会将当时的本地时间减去dtUpdateTime，转换为秒，然后设定该成员，因此该成员表示的是状态数据的过期时间。客户端软件接收到CmdService返回的状态数据后，应该只判断iTimeOffset成员，而不是dtUpdateTime，如果iTimeOffset大于阈值（例如60秒），则说明该状态数据已经无效，应认为该设备处于设备或者网络故障；

备注：当iCheckResult为CMD\_RESULT\_POWER\_OFF（开机）、CMD\_RESULT\_RESET（重启）状态时，建议不再判断iTimeOffset。

(5) csParam：设备状态数据结构，见后续描述；

(6) emParam：设备处于紧急状态时的紧急参数，该成员只对播放控制器设备有效，见后续描述；

(7) PlayContent：设备的播出内容描述，该成员只对播放控制器设备有效，见后续描述。

备注：只有一个设备主动汇报设备状态后，CmdService才会有该设备的状态数据，因此其返回的状态数据中仅仅包含所维护的设备，客户端软件不能认为CmdService会返回所有PIS设备的状态，当客户端软件从CmdService查询不到一个设备的状态时，应认为该设备处于设备或者网络故障。

### 3.4.2 CHECK\_STATUS\_PARAM结构

#define MAX\_PDP\_NUM 32

typedef struct

{

char szStatus[924]; //设定的显示信息的内容或者返回的状态字符串

char szError[194]; //返回的错误描述

\_int8 iStatus; //设备状态

\_int8 iAlarmLevel; //故障等级

\_int8 bEmergent; //是否处于紧急状态

//硬盘状态

\_int8 bHDProperty; //是否具备硬盘空间属性

\_int8 iHDStatus; //硬盘检测结果：CMD\_RESULT\_OK或者CMD\_RESULT\_ERROR

char szHDStatus[256]; //硬盘逻辑驱动器的空间描述

ULONGLONG uHDSumSpace; //放置素材的硬盘总空间，单位为节

ULONGLONG uHDFreeSpace; //放置素材的硬盘剩余空间，单位为字节

//对应的受控屏幕状态

\_int8 bScreenProperty; //是否具备受控屏幕属性

char szPDPName[32]; //受控屏幕名称，例如电源控制器，PDP屏

\_int8 iScreenDllOk; //屏幕控制DLL的加载状态，由DLL\_LOAD\_STATUS描述

\_int8 iPDPNum; //受控屏幕数量

\_int8 iPDPStatus[MAX\_PDP\_NUM]; //由SCREEN\_STATUS枚举结构描述

} CHECK\_STATUS\_PARAM, \* PCHECK\_STATUS\_PARAM;

(1) szStatus[924]：状态字符串描述，格式为：属性名称1\t属性值1\v属性名称2\t属性值2\v；

(2) szError[196]：错误描述字符串；

iStatus：设备状态的检测结果，由枚举结构CMD\_RESULT描述；

iAlarmLevel：设备故障时报警等级，0～4。设备报警调整为5个等级：1绿、2黄、3橙、4红、5深红。绿表示设备完全正常；黄表示出现轻微故障，不影响正常使用；橙表示出现中等故障，不影响使用，但需要尽快修复故障；红表示出现严重故障，不能正常使用，必须马上修复；深红表示出现严重故障，不能正常使用，同时影响到其他设备的正常工作，必须马上修复；；

(3) bEmergent：是否处于紧急状态；

(4) bHDProperty：是否具备硬盘空间属性；

(5) iHDStatus：硬盘状态检测结果，CMD\_RESULT\_OK或者CMD\_RESULT\_ERROR；

(6) szHDStatus[256]：硬盘空间描述，格式和szStatus成员一样；

(7) uHDSumSpace：放置素材的硬盘总空间（D盘），单位为字节；

(8) uHDFreeSpace：放置素材的硬盘剩余空间（D盘），单位为字节；

(9) bScreenProperty：是否具备受控显示屏或者电源控制器属性；

(10) szPDPName[32]：受控显示屏的名称，例如电源控制器，PDP屏；

(11) iScreenDllOk：屏幕控制DLL的加载状态，由DLL\_LOAD\_STATUS描述；

(12) iPDPNum：受控屏幕数量；

(13) iPDPStatus[MAX\_PDP\_NUM]：屏幕状态，由SCREEN\_STATUS枚举结构描述。

enum DLL\_LOAD\_STATUS

{

DL\_STATUS\_OK = 0, //屏幕控制DLL加载正常

DL\_STATUS\_SET\_NONE, //未设置屏幕控制DLL

DL\_STATUS\_LOAD\_FAIL, //屏幕控制DLL加载失败

DL\_STATUS\_INIT\_FAIL, //屏幕控制DLL初始化失败

MAX\_DL\_STATUS

};

enum SCREEN\_STATUS

{

SCREEN\_STATUS\_OFF = 0, //关闭状态

SCREEN\_STATUS\_ON, //开启状态

SCREEN\_STATUS\_ERROR, //故障状态

SCREEN\_STATUS\_UNKNOWN, //未知状态

MAX\_SCREEN\_STATUS

};

### 3.4.3 NET\_ES\_PARAM结构

NET\_ES\_PARAM结构描述的是紧急状态的内容。

typedef struct

{

int iRightLevel; //权限级别码，数字越大级别越高

int iEsType; //紧急状态类型，由enum NET\_ES\_TYPE结构描述

char szContent[1022]; //表示需要设置的紧急状态内容或者紧急版式名称，以NULL结束。

char iSoundType; //发布紧急信息时声音处理类型，0表示不处理，1表示关闭声音，2表示设置音量，由枚举结构NET\_ES\_SOUND\_TYPE描述

BYTE buLeftVolume; //发布紧急信息时调整的左声道音量

BYTE buRightVolume; //发布紧急信息时调整的右声道音量

BOOL bFullScreen; //如果是设置紧急内容，表示是否使用全屏显示。

//若为true，则使用全屏实现紧急内容显示，否则只使用视频区域

WORD wTimeLength; //紧急状态持续时间，若为0，表示一直处于紧急状态，直到手动停止

WORD wReplaceInfo; //紧急信息的播出模式，1表示替换之前的，0表示追加到最后

} NET\_ES\_PARAM, \* PNET\_ES\_PARAM;

enum NET\_ES\_TYPE

{

NET\_ES\_CONTENT = 0, //采用紧急内容，通过紧急模块实现，szContent维护紧急内容

NET\_ES\_LAYOUT, //采用紧急版式，szContent维护版式文件名

MAX\_NET\_ES\_TYPE

};

enum NET\_ES\_SOUND\_TYPE

{

NET\_ES\_SOUND\_NONE = 0, //不处理

NET\_ES\_SOUND\_OFF, //关闭声音

NET\_ES\_SOUND\_SET\_VOLUME, //设置音量

MAX\_NET\_ES\_SOUND\_TYPE

};

### 3.4.4 PLAY\_CONTENT结构

PLAY\_CONTENT结构描述播出模块的内容。

typedef struct

{

int iModuleNum; //模块个数

char szPlayContent[1280]; //播出内容描述字符串

} PLAY\_CONTENT, \* PPLAY\_CONTENT;

szPlayContent[1280]：格式为模块名称1\t播出内容1\v模块名称2\t播出内容2\v。

## 3.5 CMD\_QUERY\_SINGLE\_STATUS命令

该命令用于客户端软件（例如网管软件）向CmdService查询一个设备的状态。

该命令只有一个CMD\_HEAD结构，其成员szDeviceID描述需要查询的设备的编号。

CmdService接收到该指令后，如果存在对应设备的状态数据，则以一个CMD\_HEAD结构和一个DEVICE\_STATUS结构作为答复，其中buAck成员设置为1，iNum成员为1；若不存在对应设备的状态数据，则以一个CMD\_HEAD结构作为答复，其中buAck成员设置为1，iNum成员为0。

其他成员与接收到的内容保持不变。

## 3.6 CMD\_QUERY\_TASK命令

该命令用于终端设备软件（例如InfoMonitor）向CmdService查询自己的操作任务。

该命令只有一个CMD\_HEAD结构，其成员szDeviceID描述需要查询的设备的编号。

CmdService接收到该指令后，如果存在对应设备的有效操作任务，则以一个CMD\_HEAD结构和若干个TASK\_SET结构作为答复（一个TASK\_SET结构描述一个操作任务），其中buAck成员设置为1，iNum成员为操作任务的个数；若不存在对应设备的操作任务，则以一个CMD\_HEAD结构作为答复，其中buAck成员设置为1，iNum成员为0。

其他成员与接收到的内容保持不变。

## 3.7 CMD\_FEEDBACK\_TASK\_RESULT命令

该命令用于终端设备软件（例如InfoMonitor）向CmdService反馈操作任务的执行结果。

该命令有一个CMD\_HEAD结构和若干个TASK\_RESULT结构，CMD\_HEAD的成员szDeviceID描述需要设备的编号，iNum成员描述TASK\_RESULT结构的数量。一个TASK\_RESULT结构描述一个操作任务的执行结果。

CmdService接收到该指令后，以一个CMD\_HEAD结构作为答复，其中buAck成员设置为1，其他成员与接收到的内容保持不变。

typedef struct

{

TASK\_LIST TaskList;

char szTaskID[33];

} TASK\_RESULT, \* PTASK\_RESULT;

TaskList：TASK\_LIST结构，描述操作任务的执行结果；

szTaskID[33]：操作任务的ID。

## 3.8 CMD\_PUBLISH\_STATUS命令

该命令用于终端设备软件（例如InfoMonitor）向CmdService汇报设备状态。

该命令有一个CMD\_HEAD结构和若干个DEVICE\_STATUS结构，CMD\_HEAD的iNum成员描述DEVICE\_STATUS结构的数量（通常为1）。

CmdService接收到该指令后，以一个CMD\_HEAD结构作为答复，其中buAck成员设置为1，其他成员与接收到的内容保持不变。

## 3.9 CMD\_CANCEL\_TASK命令

该命令用于清除之前向CmdService发布的的操作任务。

该命令只有一个CMD\_HEAD结构，CMD\_HEAD的szTaskID成员描述要清除的任务ID。

CmdService接收到该指令后，以一个CMD\_HEAD结构作为答复，其中buAck成员设置为1，其他成员与接收到的内容保持不变。

该命令可用于向车载设备发布一个长时间有效的紧急信息发布任务，因为车载设备通过无线网络连接，而无线网络效果较差的话，在较短的有效时间内，车载设备有可能无法接收到紧急信息任务。当需要取消该紧急信息时，先通过CMD\_CANCEL\_TASK命令清除掉之前发布的紧急信息任务，然后再发布一个长时间有效的取消紧急信息的操作任务。

## 3.10 CMD\_TEST命令

该命令用于测试CmdService的运行状态是否正常。

该命令只有一个CMD\_HEAD结构，CmdService接收到该指令后，以一个CMD\_HEAD结构作为答复，其中buAck成员设置为1，其他成员与接收到的内容保持不变。

# 4 测试代码

## 4.1 查询所有设备状态

BOOL QueryStatus(LPCTSTR szServerIP, UINT uWatchPort)

{

BOOL bReturn = false;

int i, iDataSize, iSend, iReceive, iRet;

PBYTE pData = NULL;

char szBuf[4096];

PDEVICE\_STATUS pDS;

PCMD\_HEAD pCmd;

CSyncSocket sock;

CDeviceInfo \* pDeviceInfo;

//init cmd head

pCmd = (PCMD\_HEAD)szBuf;

memset(pCmd, 0, sizeof(CMD\_HEAD));

pCmd->buType = CMD\_QUERY\_STATUS;

pCmd->iDataSize = sizeof(CMD\_HEAD);

strcpy(pCmd->szHead, CMD\_FLAG);

//init socket

if(!sock.Create())

{ RecordLogToFile("QueryCenterStatus: create socket error");

goto ErrorExit;

}

if(!sock.Connect(szServerIP, uWatchPort))

{ RecordLogToFile("QueryCenterStatus: Connect server error");

goto ErrorExit;

}

sock.SetTimeOut(CSyncSocket::TO\_TYPE\_BOTH, 3000);

//send and receive

iSend = sock.Send(pCmd, pCmd->iDataSize);

if(iSend != pCmd->iDataSize)

{ RecordLogToFile("QueryCenterStatus: Send query status error");

goto ErrorExit;

}

Sleep(200);

iReceive = 0;

iRet = sock.Receive(pCmd, sizeof(CMD\_HEAD));

if( iRet == sizeof(CMD\_HEAD) && strcmp(pCmd->szHead, CMD\_FLAG) == 0 && pCmd->buAck

&& (UINT)pCmd->iDataSize == (sizeof(CMD\_HEAD) + pCmd->iNum \* sizeof(DEVICE\_STATUS)) )

{ //prepare buffer

iDataSize = pCmd->iDataSize;

pData = new BYTE[iDataSize];

//copy data

memcpy(pData, pCmd, iRet);

iReceive += iRet;

while(iReceive < iDataSize)

{ iRet = sock.Receive(szBuf, sizeof(szBuf));

if(iRet > 0)

{ memcpy(pData + iReceive, szBuf, iRet);

iReceive += iRet;

}

else

{ break;

}

}

//check

if(iReceive == iDataSize)

{ pCmd = (PCMD\_HEAD)pData;

for(i = 0; i < pCmd->iNum; i ++)

{ pDS = (PDEVICE\_STATUS)(pData + sizeof(CMD\_HEAD) + sizeof(DEVICE\_STATUS) \* i);

//pDS即设备状态结构指针

}

}

else

{ RecordLogToFile("QueryCenterStatus: Receive query status ack error");

goto ErrorExit;

}

}

else

{ RecordLogToFile("QueryCenterStatus: Receive query status ack error");

goto ErrorExit;

}

bReturn = true;

//close and free data

ErrorExit:

if(pData)

{ delete pData;

}

sock.Close();

return bReturn;

}

## 4.2 查询单个设备状态

void QuerySingleStatus(LPCTSTR szServerIP, UINT uWatchPort, LPCTSTR szDeviceID)

{

int iSend, iReceive;

char szBuf[4096];

PDEVICE\_STATUS pDS;

PCMD\_HEAD pCmd = (PCMD\_HEAD)szBuf;

CSyncSocket sock;

//init cmd head

memset(pCmd, 0, sizeof(CMD\_HEAD));

pCmd->buType = CMD\_QUERY\_SINGLE\_STATUS;

pCmd->iDataSize = sizeof(CMD\_HEAD);

strcpy(pCmd->szHead, CMD\_FLAG);

strcpy(pCmd->szDeviceID, szDeviceID);

//init socket

if(!sock.Create())

{ RecordLogToFile("QuerySingleStatus: create socket error");

goto ErrorExit;

}

if(!sock.Connect(szServerIP, uWatchPort))

{ RecordLogToFile("QuerySingleStatus: Connect server error");

goto ErrorExit;

}

sock.SetTimeOut(CSyncSocket::TO\_TYPE\_BOTH, 3000);

//send and receive

iSend = sock.Send(pCmd, pCmd->iDataSize);

if(iSend != pCmd->iDataSize)

{ RecordLogToFile("QuerySingleStatus: Send query status error");

goto ErrorExit;

}

Sleep(50);

iReceive = sock.Receive(szBuf, sizeof(szBuf));

if(iReceive >= sizeof(CMD\_HEAD) && pCmd->iDataSize == iReceive && pCmd->buAck)

{ if(pCmd->iNum)

{ pDS = (PDEVICE\_STATUS)(szBuf + sizeof(CMD\_HEAD));

//pDS即设备状态结构指针

}

}

else

{ RecordLogToFile("QuerySingleStatus: Receive query status ack error");

goto ErrorExit;

}

//close and free data

ErrorExit:

sock.Close();

}

## 4.3 发布操作任务

操作任务的信息预先存储于OperationList，所有的设备信息预先存储于ptrDevice。操作任务ID可以使用MD5算法生成唯一的32字节长度的字符串。

BOOL ExecuteOperation(LPCTSTR szServerIP, UINT uWatchPort, CTypedPtrArray<CPtrArray, CDeviceInfo \*> & ptrDevice, CString & strMsg)

{

COleDateTime tm;

int i, iDataSize, iSend, iReceive;

PBYTE pData = NULL;

PCMD\_HEAD pCmd;

PTASK\_SET pTaskSet;

PTASK\_LIST pTaskList;

CDeviceInfo \* pDeviceInfo;

CSyncSocket sock;

//clear msg

strMsg.Empty();

//get time

tm = COleDateTime::GetCurrentTime();

//prepare data

iDataSize = sizeof(CMD\_HEAD) + sizeof(TASK\_SET) + ptrDevice.GetSize() \* sizeof(TASK\_LIST);

pData = new BYTE[iDataSize];

if(!pData)

{ strMsg.Format("allocate memory error");

goto ErrorExit;

}

//init cmd head

pCmd = (PCMD\_HEAD)pData;

memset(pCmd, 0, iDataSize);

pCmd->buType = CMD\_PUBLIC\_TASK;

pCmd->iDataSize = iDataSize;

strcpy(pCmd->szHead, CMD\_FLAG);

pCmd->iNum = ptrDevice.GetSize();

//init task set

pTaskSet = (PTASK\_SET)(pData + sizeof(CMD\_HEAD));

strcpy(pTaskSet->szTaskID, g\_OperationList.strTaskID);

pTaskSet->iTaskType = g\_OperationList.OperationInfo.iType;

pTaskSet->iEmType = g\_OperationList.OperationInfo.iEmType;

pTaskSet->iFullScreen = g\_OperationList.OperationInfo.bFullScreen;

pTaskSet->iReplaceInfo = (\_\_int8)g\_OperationList.OperationInfo.wReplaceInfo;

pTaskSet->iSoundType = g\_OperationList.OperationInfo.iSoundType;

pTaskSet->buLeftVolume = g\_OperationList.OperationInfo.buLeftVolume;

pTaskSet->buRightVolume = g\_OperationList.OperationInfo.buRightVolume;

strcpy(pTaskSet->szEmergencyInfo, g\_OperationList.OperationInfo.szContent);

pTaskSet->dtTaskTime = tm.operator DATE();

pTaskSet->iValidTime = gParam.sysParam.iCmdValidTime;

pTaskSet->iTimeLength = g\_OperationList.OperationInfo.wTimeLength;

strcpy(pTaskSet->szEditorID, gParam.UserInfo.szUserID);

pTaskSet->buRightLevel = gParam.UserInfo.iRightLevel;

//init task list

for(i = 0; i < pCmd->iNum; i ++)

{ pTaskList = (PTASK\_LIST)(pData + sizeof(CMD\_HEAD) + sizeof(TASK\_SET) + sizeof(TASK\_LIST) \* i);

pDeviceInfo = ptrDevice.GetAt(i);

strcpy(pTaskList->szDeviceCode, pDeviceInfo->szDeviceCode);

pTaskList->iExecResult = CMD\_RESULT\_WAIT\_EXEC;

}

//init socket

if(!sock.Create())

{ strMsg.Format("create socket error");

goto ErrorExit;

}

if(!sock.Connect(szServerIP, uWatchPort))

{ strMsg.Format("Connect server error");

goto ErrorExit;

}

sock.SetTimeOut(CSyncSocket::TO\_TYPE\_BOTH, 3000);

//send and receive

iSend = sock.Send(pCmd, pCmd->iDataSize);

if(iSend != pCmd->iDataSize)

{ strMsg.Format("Send data error");

goto ErrorExit;

}

Sleep(50);

iReceive = sock.Receive(pCmd, iDataSize);

if(!(iReceive == sizeof(CMD\_HEAD) && pCmd->iDataSize == iReceive && pCmd->buAck))

{ strMsg.Format("Receive data error");

goto ErrorExit;

}

//close and free data

sock.Close();

delete pData;

return true;

ErrorExit:

if(pData)

{ delete pData;

}

sock.Close();

return false;

}

操作任务ID生成示例函数：m\_csUniqueID为同步对象CCriticalSection

CString::GetUniqueID()

{

CString strID;

SYSTEMTIME sysTime;

DWORD dwTime;

static ULONGLONG ulIndex = 0;

m\_csUniqueID.Lock();

GetLocalTime(&sysTime);

dwTime = GetTickCount();

strID.Format("%04d-%02d-%02d %2d:%02d:%02d.%d %d %d %I64d %s", sysTime.wYear, sysTime.wMonth, sysTime.wDay,

sysTime.wHour, sysTime.wMinute, sysTime.wSecond, sysTime.wMilliseconds, dwTime, GetCurrentProcessId(),

ulIndex ++, GetLocalIP());

m\_csUniqueID.Unlock();

return MD5Encode32(strID, strID.GetLength());

}

## 4.4 查询操作任务执行结果

BOOL QueryExecuteResult(LPCTSTR szServerIP, UINT uWatchPort, CTypedPtrArray<CPtrArray, CDeviceInfo \*> & ptrDevice, LPCTSTR szTaskID, CString & strMsg)

{

int i, iDataSize, iSend, iReceive;

PBYTE pData = NULL;

PCMD\_HEAD pCmd;

PTASK\_LIST pTaskList;

CDeviceInfo \* pDeviceInfo;

CSyncSocket sock;

//clear msg

strMsg.Empty();

//prepare data

iDataSize = sizeof(CMD\_HEAD) + ptrDevice.GetSize() \* sizeof(TASK\_LIST);

pData = new BYTE[iDataSize];

if(!pData)

{ strMsg.Format("allocate memory error");

goto ErrorExit;

}

//init cmd head

pCmd = (PCMD\_HEAD)pData;

memset(pCmd, 0, sizeof(CMD\_HEAD));

pCmd->buType = CMD\_QUERY\_TASK\_RESULT;

pCmd->iDataSize = sizeof(CMD\_HEAD);

strcpy(pCmd->szHead, CMD\_FLAG);

strcpy(pCmd->szTaskID, szTaskID);

//init socket

if(!sock.Create())

{ strMsg.Format("create socket error");

goto ErrorExit;

}

if(!sock.Connect(szServerIP, uWatchPort))

{ strMsg.Format("Connect server error");

goto ErrorExit;

}

sock.SetTimeOut(CSyncSocket::TO\_TYPE\_BOTH, 3000);

//send

iSend = sock.Send(pCmd, pCmd->iDataSize);

if(iSend != pCmd->iDataSize)

{ strMsg.Format("Send data error");

goto ErrorExit;

}

Sleep(50);

iReceive = sock.Receive(pCmd, iDataSize);

if(!(iReceive == iDataSize && pCmd->iDataSize == iReceive && pCmd->buAck))

{ strMsg.Format("Receive data error");

goto ErrorExit;

}

//parse result

for(i = 0; i < pCmd->iNum; i ++)

{ pTaskList = (PTASK\_LIST)(pData + sizeof(CMD\_HEAD) + sizeof(TASK\_LIST) \* i);

//pTaskList为执行结果

}

//close and free data

sock.Close();

delete pData;

return true;

ErrorExit:

if(pData)

{ delete pData;

}

sock.Close();

return false;

}

## 4.5 发布设备状态

void UpdateStatus(LPCTSTR szServerIP, UINT uWatchPort)

{

int i, iSize, iDataSize, iSend, iReceive;

PBYTE pData = NULL;

PCMD\_HEAD pCmd;

PDEVICE\_STATUS pDS;

COleDateTime tm = COleDateTime::GetCurrentTime();

CSyncSocket sock;

//allocate memory

iDataSize = sizeof(CMD\_HEAD) + sizeof(DEVICE\_STATUS);

pData = new BYTE[iDataSize];

if(!pData)

{ RecordLog("UpdateStatus: allocate memory error");

goto ErrorExit;

}

memset(pData, 0, iDataSize);

//init socket

if(!sock.Create())

{ RecordLog("UpdateStatus: create socket error");

goto ErrorExit;

}

if(!sock.Connect(szServerIP, uWatchPort))

{ RecordLog("UpdateStatus: Connect server error");

goto ErrorExit;

}

sock.SetTimeOut(CSyncSocket::TO\_TYPE\_BOTH, 3000);

//init cmd

pCmd = (PCMD\_HEAD)pData;

pCmd->buType = CMD\_PUBLIC\_STATUS;

strcpy(pCmd->szHead, CMD\_FLAG);

pCmd->iNum = 1;

pCmd->iDataSize = iDataSize;

pDS = (PDEVICE\_STATUS)(pData + sizeof(CMD\_HEAD) + sizeof(DEVICE\_STATUS));

pDS->dtUpdateTime = tm.operator DATE();

//设置pDS成员

//send

iSend = sock.Send(pCmd, pCmd->iDataSize);

if(iSend != pCmd->iDataSize)

{ goto ErrorExit;

}

Sleep(500);

iReceive = sock.Receive(pData, iDataSize);

ErrorExit:

if(pData)

{ delete pData;

}

sock.Close();

}