IEC 104

客户端开发包接口 API 使用手册

版本：V2.0

修订日期：2024 年 3 月 8 日

1. 开发包简介

IEC104 客户端(Master)接口调用说明文档，主要包括 104 核心命令的接口和调用参数的详细说明。
客户利用该开发包，可以快速，高效的完成 104 客户端的开发，不用考虑 104 标准具体的实现细节，只需要会简单调用并进行二次开发，就可以完成产品的研发。
开发包简单，易用，封装性良好，是开发 104 主站最快捷的方式。

2. 接口描述

2.1. 初始化（不带链路参数）

函数声明	int TS_DLLEXPORT IEC_Init(const char* svrName,const char* cliIp, const char* serIp); //初始化 1-初始化完成 0-失败
功能	初始化过程
参数	const char *svrName：连接从站的名字，作为当前连接标识使用； cliIp: 主站 IP 地址 serIp:从站 IP
返回值	发送成功返回 1，否则为 0;
示例	bool ret = IEC_Init ("test","192.168.3.147","192.168.3.147");

Tip：当初始化成功后，后续命令才能发送成功

2.2. 初始化（带链路参数 与前面初始化，二选一即可）

函数声明	int TS_DLLEXPORT IEC_InitByParam(const char* svrName,const char* cliIp, const char* serIp,LinkParameters *pLink); //初始化 1-初始化完成 0-失败 pLink-链路参数
功能	初始化过程
参数	const char *svrName：连接从站的名字，作为当前连接标识使用； cliIp: 主站 IP 地址 serIp:从站 IP pLink:链路参数
返回值	发送成功返回 1，否则为 0;

示例	LinkParameters Link; int ret = IEC_InitByParam("test","192.168.3.147","192.168.3.147",&Link);

Tip：当初始化成功后，后续命令才能发送成功

链路参数说明:

```
struct LinkParameters{
    IEC_WORD   t0;                  // 网络建立连接超时时间 单位:秒
    IEC_WORD   t1;                  // 发送或测试 APDU 的超时时间 单位:秒
    IEC_WORD   t2;                  //接收方无数据报文时确认的超时时间 S 格式的
                                    //超时时间 单位:秒
    IEC_WORD   t3;                  //通道长期空闲时发送测试帧的超时时间 单位:秒
    IEC_WORD    t4;                 //应用报文确认超时时间 单位:秒
    IEC_WORD   K;                   // 发送方未被确认的 I 格式的 APDU 的最大数目
    IEC_WORD   W;                   // 接收方最多收到未被确认的 I 个数的 APDU 的
                                    //最大数目
                                    // 【接收方最迟收到 W 个 I 就必须要回复确认帧
                                    //默认值为 8；主站使用，这里不使用】
                                    // W 不能够超过 K 的 2/3
    IEC_WORD wStationNo;            //站地址
    //超时时间选择: t0 < 30s; t1 < 20s; t2 < 15s; t2 < t1; t3 < 25s;
    void Init(){
        //给定默认初始值
        t0 = 10;
        t1 = 12;
        t2 = 5;
        t3 = 15;
        t4 = 8;
        K = 12;
        W = 8;
        wStationNo = 1;
    }
};
```

2.3. 注册激活确认回调

函数声明	void TS_DLLEXPORT Reg_Svr_Act_Con_Res(Svr_Act_Con_Resp callback); 获取激活确认
功能	获取激活确认
参数	callback:函数指针
返回值	void
示例	void Svr_Act_Con_Resp_CallBack(const char* svrName, Act_Con_Resp *) {

	//回调处理 resp 参数具体看头文件注释
	}
	//回调注册
	Reg_Svr_Act_Con_Res(Svr_Act_Con_Resp_CallBack);

2.4. 注册激活终止回调

函数声明	void TS_DLLEXPORT Reg_Svr_Act_Term_Res(Svr_Act_Term_Resp callback);
功能	获取激活终止信息
参数	callback:函数指针
返回值	void

示例	void Svr_Act_Term_Resp_CallBack(const char* svrName, Act_Term_Resp*)
	{
	//回调处理 resp 参数具体看头文件注释
	}
	//回调注册
	Reg_Svr_Act_Term_Res(Svr_Act_Term_Resp_CallBack);

2.5. 注册故障事件回调

函数声明	void TS_DLLEXPORT Reg_Svr_CRashEvent_Res(Svr_CRashEvent_Resp callback);
功能	获取激活确认
参数	callback:函数指针
返回值	void

示例	void Svr_CRashEvent_Resp_CallBack(const char* svrName, CRashEvent*){
	//回调处理 resp 参数具体看头文件注释
	}
	//回调注册
	Reg_Svr_CRashEvent_Res(Svr_CRashEvent_Resp_CallBack);

2.6. 发送报文回调

函数声明	void TS_DLLEXPORT Reg_Svr_GetSend_Msg(Svr_GetSend_Msg callBack);
功能	获取发送报文

参数	callback:函数指针
返回值	void

示例	void Svr_GetSend_Msg_CallBack(MsgInfo *msg)
	{
	}
	//回调注册
	Reg_Svr_GetSend_Msg(Svr_GetSend_Msg_CallBack);

2.7. 接收报文回调

函数声明	void TS_DLLEXPORT Reg_Svr_GetRecv_Msg(Svr_GetRecv_Msg callBack);
功能	截取接收报文
参数	callback:函数指针
返回值	void

示例	void Svr_GetRecv_Msg_CallBack(MsgInfo *msg); {
	}
	//回调注册
	Reg_Svr_GetRecv_Msg(Svr_GetRecv_Msg_CallBack);

2.8. 获取当前连接状态

函数声明	int TS_DLLEXPORT GetConnectStatus(const char* svrName);
功能	获取连接状态
参数	const char *svrName：连接从站的名字，作为当前连接标识使用；
返回值	0-断开 1-连接
示例	int ret = GetConnectStatus ("test");

2.9. 注册连接断开回调函数

函数声明	void TS_DLLEXPORT Reg_Svr_LostConct(Svr_lostConct callBack);
功能	获取连接断开状态
参数	callBack:函数指针
返回值	0-断开 1-连接

示例	void Svr_lostConct_CallBack(const char* svrName,const char* ,const char*)
	{
	//回调处理
	}
	//回调注册
	Reg_Svr_LostConct(Svr_lostConct_CallBack);

2.10. 获取请求响应状态

函数声明	int TS_DLLEXPORT CheckConSatusById(const char*servName, int reqId);
功能	根据请求返回的序号，获取响应状态
参数	const char *svrName：连接从站的名字，作为当前连接标识使用； reqId: 请求返回序号
返回值	0-已经回复 1-没有回复

示例	int reqId = 100; //请求返回值 此处直接赋值
	const char* servName = "test";
	int ret = CheckConSatusById(servName, reqId);

2.11. 总召（GI）请求

函数声明	int TS_DLLEXPORT Send_GI(const char* svrName, IEC_BYTE groupNum = 20, int waitResult = 1); //发送Gi
功能	发送 GI 请求
参数	const char *svrName：连接从站的名字，作为当前连接标识使用； groupNum: 20(总召唤)，其他为第 i 组（i 从 20 起），详情看协议 waitResult:1-等待请求响应 0-不等待请求响应 可根据开发情况使用
返回值	当返回值为-1 时，发送失败（当 waitResult = 1 时,为未响应），否则返回当前请求序号
示例	const char* servName = "test"; //...中间省略初始化过程 int reqId = Send_GI(servName);

2.12. 单点信息回调注册

函数声明	void TS_DLLEXPORT Reg_Svr_SPI_Res(Svr_Ctrl_SPI_Resp

	callback);
功能	单点信息获取
参数	callback:函数指针
返回值	void

示例	void Svr_Ctrl_SPI_Resp_CallBack(const char* svrName, Ctrl_SPI_Resp *resp){
	//回调处理 resp 参数具体看头文件注释
	}
	//回调注册
	Reg_Svr_SPI_Res(Svr_Ctrl_SPI_Resp_CallBack);

2.13. 双点信息回调注册

函数声明	void TS_DLLEXPORT Reg_Svr_DPI_Res(Svr_Ctrl_DPI_Resp callback);
功能	获取单点回调信息
参数	callback:回调函数指针
返回值	无
示例	void Svr_Ctrl_DPI_Resp_CallBack(const char* svrName, Ctrl_DPI_Resp pResp*){} Reg_Svr_DPI_Res(Svr_Ctrl_DPI_Resp_CallBack);

2.14. 注册测量信息回调

函数声明	void TS_DLLEXPORT Reg_Svr_MEASURE_Res(Svr_Ctrl_MEASURE_Resp callback);
功能	获取测量信息
参数	callback:回调函数
返回值	void
示例	//回调声明实现 void Svr_Ctrl_MEASURE_Resp_CallBack(const char* svrName, Ctrl_MEASURE_Resp *){} Reg_Svr_DPI_Res(Svr_Ctrl_MEASURE_Resp_CallBack);

2.15. 注册步长位置信息回调

函数声明	void TS_DLLEXPORT Reg_Svr_Ctrl_STEP_Resp(Svr_Ctrl_STEP_Resp callback);
功能	获取步长位置信息

参数	callback:回调函数
返回值	void
示例	//回调声明实现 void Svr_Ctrl_STEP_Resp_CallBack(const char* svrName, Ctrl_STEP_Resp*){} Reg_Svr_Ctrl_STEP_Resp(Svr_Ctrl_STEP_Resp_CallBack);

2.16. 带变位检出的成组单点信息回调注册

函数声明	void TS_DLLEXPORT Reg_Svr_SPI_INFO_StatusChange(Svr_SPI_INFO_StatusChange callback);
功能	获取测量信息
参数	callback:回调函数
返回值	void
示例	//回调声明实现 void Svr_SPI_INFO_StatusChange_CallBack(const char* svrName, SPI_INFO_StatusChange*){} Reg_Svr_SPI_INFO_StatusChange(Svr_Ctrl_MEASURE_Resp_CallBack);

2.17. 注册测量信息回调

函数声明	void TS_DLLEXPORT Reg_Svr_MEASURE_Res(Svr_Ctrl_MEASURE_Resp callback);
功能	获取测量信息
参数	callback:回调函数
返回值	void
示例	//回调声明实现 void Svr_Ctrl_MEASURE_Resp_CallBack(const char* svrName, Ctrl_MEASURE_Resp *){} Reg_Svr_DPI_Res(Svr_Ctrl_MEASURE_Resp_CallBack);

2.18. 注册测量信息回调

函数声明	void TS_DLLEXPORT Reg_Svr_MEASURE_Res(Svr_Ctrl_MEASURE_Resp callback);
功能	获取测量信息
参数	callback:回调函数
返回值	void

示例	//回调声明实现 void Svr_Ctrl_MEASURE_Resp_CallBack(const char* svrName, Ctrl_MEASURE_Resp *){} Reg_Svr_DPI_Res(Svr_Ctrl_MEASURE_Resp_CallBack);

2.19. 电能 Gi 请求

函数声明	int TS_DLLEXPORT Send_ElectGI(const char* svrName, IEC_BYTE groupNum = 5, int waitResult = 1);
功能	发送电能 GI 请求
参数	const char *svrName：连接从站的名字，作为当前连接标识使用； groupNum: 0:未用 1..4 召第 i 组 5 总召唤(默认值) waitResult:1-等待请求响应 0-不等待请求响应 可根据开发情况使用
返回值	当返回值为-1 时，发送失败（当 waitResult = 1 时,为未响应），否则返回当前请求序号
示例	const char* servName = "test"; //...中间省略初始化过程 int reqId = Send_ElectGI(servName);

2.20. 电量信息回调注册

函数声明	void TS_DLLEXPORT Reg_Svr_ELECTRIC_Res(Svr_Ctrl_ELECTTIC_Resp callback);
功能	获取电度信息
参数	callback:回调函数
返回值	void
示例	//回调声明实现 void Svr_Ctrl_ELECTTIC_Resp_CallBack(const char* svrName, Ctrl_ELECTRIC_Resp *){} Reg_Svr_ELECTRIC_Res(Svr_Ctrl_ELECTTIC_Resp_CallBack);

2.21. 发送时间同步

函数声明	int TS_DLLEXPORT Send_ClockSyn(const char* svrName, const char* time, int waitResult = 1);
功能	发送时间同步请求
参数	const char *svrName：连接从站的名字，作为当前连接标识使用； time: 时间（格式固定,如"2000-01-01 08:00:00.000"）

	waitResult:1-等待请求响应 0-不等待请求响应 可根据开发情况使用
返回值	当返回值为-1 时，发送失败（当 waitResult = 1 时,为未响应），否则返回当前请求序号
示例	const char* servName = "test"; const char* time = "2000-01-01 08:00:00.000"; //...中间省略初始化过程 int reqId = Send_ClockSyn(servName,time);

2.22. 注册时间同步信息回调

函数声明	void TS_DLLEXPORT Reg_Svr_ClockSyn_Res(Svr_ClockSyn_Resp callback);
功能	获取事件同步信息
参数	callback:回调函数
返回值	void
示例	//回调声明实现 void Svr_ClockSyn_Resp_CallBack(const char* svrName, ClockSyn_Resp *){} Reg_Svr_ClockSyn_Res(Svr_ClockSyn_Resp_CallBack);

2.23. 发送复位进程

函数声明	int TS_DLLEXPORT Send_ReSetPorcess(const char* svrName,IEC_BYTE qrp = 1, int waitResult = 1);
功能	发送复位进程请求
参数	const char *svrName：连接从站的名字，作为当前连接标识使用； qrp: 1-进程总复位 其他-看协议 qrp 定义 waitResult:1-等待请求响应 0-不等待请求响应 可根据开发情况使用
返回值	当返回值为-1 时，发送失败（当 waitResult = 1 时,为未响应），否则返回当前请求序号
示例	const char* servName = "test"; //...中间省略初始化过程 int reqId = Send_ReSetPorcess(servName);

2.24. 复位进程确认回调

函数声明	void TS_DLLEXPORT

	Reg_Svr_ReSetProcess_Conf(Svr_ReSetProcess_Conf callback);
功能	获取复位进程信息
参数	callback:回调函数
返回值	void
示例	//回调声明实现 void Svr_ReSetProcess_Conf_CallBack(const char* svrName, ReSetProcess_Conf*);{} Reg_Svr_ReSetProcess_Conf(Svr_ReSetProcess_Conf_CallBack);

2.25. 遥控

函数声明	int TS_DLLEXPORT Send_YK(const char* svrName, Ctrol_YK_Param ctrlParam);
功能	发送遥控请求
参数	const char *svrName：连接从站的名字，作为当前连接标识使用； ctrlParam: 遥控参数，需要根据遥控的类型进行配合配置
返回值	当返回值为-1 时，发送失败，否则返回当前请求序号

示例	const char* servName = "test";
	//...中间省略初始化过程
	Ctrol_YK_Param ctrlParam;
	int nTx = 0;
	ctrlParam.Init();
	ctrlParam.ctl_actModel = ENUM_104_COT_ACT;//遥控激活
	ctrlParam.ctl_address = 0x0001;//遥控地址
	ctrlParam.ctl_oper = ENUM_CTRL_EXE_REQ;//遥控执行
	ctrlParam.ctl_Val = ENUM_CTRL_SC_OPEN;//遥控值 分闸
	ctrlParam.ctl_Test = 0;
	ctrlParam.ctl_model = ENUM_CTRL_SC_DEFAULT;
	nTx = Send_YK(serName, ctrlParam);

Tips:此接口调用需要根据参数进行配合，但是比较灵活。也可以调用下面的接口

2.26. 遥控-选择

函数声明	int TS_DLLEXPORT Send_YK_Select(const char* svrName, ENUM_CTRL_MODE ctlMode, IEC_DWORD address, ENUM_104_CTRL_CS ctlVal, IEC_BYTE test = 0, IEC_BYTE negative = 0);
功能	发送遥控选择请求
参数	const char *svrName:连接从站的名字，作为当前连接标识使用；

	ctlMode: 控制的类型，如单点遥控； address: 遥控地址 ctlVal:遥控值 test:是否检修（0-非检修 1-检修） negative:是否为负响应（0-正响应 1-负响应）
返回值	当返回值为-1 时，发送失败，否则返回当前请求序号

示例	const char* servName = "test";
	//...中间省略初始化过程
	Ctrol_YK_Param ctrlParam;
	int nTx = 0;
	ENUM_104_CTRL_CS ctlVal = ENUM_CTRL_SC_CLOSE;
	nTx = Send_YK_Select(serName,ENUM_CTRL_MODE_SC, 0x0001, ctlVal);

2.27. 遥控-取消

函数声明	int TS_DLLEXPORT Send_YK_Cancel(const char* svrName,IEC_BYTE SelOrExe, ENUM_CTRL_MODE ctlMode, IEC_DWORD address, ENUM_104_CTRL_CS ctlVal, IEC_BYTE test = 0, IEC_BYTE negative = 0);
功能	发送遥控取消请求
参数	const char *svrName:连接从站的名字，作为当前连接标识使用； SelOrExe:选择还是执行 (0-选择 1-执行) ctlMode: 控制的类型，如单点遥控； address: 遥控地址 ctlVal:遥控值 test:是否检修（0-非检修 1-检修） negative:是否为负响应（0-正响应 1-负响应）
返回值	当返回值为-1 时，发送失败，否则返回当前请求序号

示例	const char* servName = "test";
	//...中间省略初始化过程
	Ctrol_YK_Param ctrlParam;
	int nTx = 0;
	ENUM_104_CTRL_CS ctlVal = ENUM_CTRL_SC_CLOSE;
	nTx = Send_YK_Cancel(serName, 0, ENUM_CTRL_MODE_SC, 0x0001, ctlVal);

2.28. 遥控-执行

函数声明	int TS_DLLEXPORT Send_YK_Execute(const char* svrName,ENUM_CTRL_MODE ctlMode, IEC_DWORD address, ENUM_104_CTRL_CS ctlVal, IEC_BYTE test = 0, IEC_BYTE negative = 0);
功能	发送遥控执行请求
参数	const char *svrName:连接从站的名字，作为当前连接标识使用； ctlMode: 控制的类型，如单点遥控； address: 遥控地址 ctlVal:遥控值 test:是否检修（0-非检修 1-检修） negative:是否为负响应（0-正响应 1-负响应）
返回值	当返回值为-1 时，发送失败，否则返回当前请求序号

示例	const char* servName = "test";
	//...中间省略初始化过程
	Ctrol_YK_Param ctrlParam;
	int nTx = 0;
	ENUM_104_CTRL_CS ctlVal = ENUM_CTRL_SC_CLOSE;
	nTx = Send_YK_Execute(serName,ENUM_CTRL_MODE_SC, 0x0001, ctlVal);

2.29. 遥调（遥设）

函数声明	int TS_DLLEXPORT Send_YK(const char* svrName, Ctrol_YK_Param ctrlParam);
功能	发送遥调(遥设)请求
参数	const char *svrName:连接从站的名字，作为当前连接标识使用； ctrlParam：设置参数
返回值	当返回值为-1 时，发送失败，否则返回当前请求序号

示例	const char* servName = "test";
	//...中间省略初始化过程
	Ctrol_YT_Param ctrlYTParam;
	ctrlYTParam.Init();
	ctrlYTParam.ctl_address = 0x0704;//设置对象地址
	ctrlYTParam.ctl_actModel = ENUM_104_COT_ACT;//激活
	ctrlYTParam.val_model = ENUM_104_C_SE_NA; //归一化值
	ctrlYTParam.ctl_oper = ENUM_CTRL_EXE_REQ;//执行
	strcpy(ctrlYTParam.ctl_Val, "3.14");//设置值
	Send_YT(serName, ctrlYTParam);

2.30. 注册遥调遥控响应回调

函数声明	void TS_DLLEXPORT Reg_Svr_YTYK_Res(Svr_CtlYK_Resp resp);
功能	获取遥控、遥调的控制信息
参数	callback:回调函数
返回值	void
示例	//回调声明实现 void Svr_CtlYK_Resp_CallBack(const char* svrName, Ctrol_YK_Resp_Param*){} Reg_Svr_YTYK_Res(Svr_CtlYK_Resp_CallBack);

2.31. 获取文件目录

函数声明	int TS_DLLEXPORT Send_GetFileDirect(const char* svrName,string dirName,string startTime, string endTime);
功能	发送获取文件目录请求
参数	const char *svrName:连接从站的名字，作为当前连接标识使用； dirName: 目录名 startTime: 查询起始时间 如: "2000-01-01 08:00:00.000"; endTime:查询终止时间 如: "2000-01-01 08:00:00.000";
返回值	当返回值为-1 时，发送失败，否则返回当前请求序号

示例	const char* servName = "test";
	//...中间省略初始化过程
	int nTx = 0;
	nTx = Send_GetFileDirect(serName, "COMTRADE", "", "");

2.32. 注册读文件目录回调

函数声明	void TS_DLLEXPORT Reg_Svr_GetFileDir_Resp(Svr_GetFileDir_Resp callback);
功能	获取目录信息
参数	callback:回调函数
返回值	void
示例	//回调声明实现 void Svr_GetFileDir_Resp_CallBack(const char* svrName,GetFileDir_Resp *resp){} Reg_Svr_GetFileDir_Resp(Svr_GetFileDir_Resp_CallBack);

2.33. 读文件

函数声明	int TS_DLLEXPORT Send_GetFile(const char* svrName,const char *fileName);
功能	发送获取文件请求
参数	const char *svrName:连接从站的名字，作为当前连接标识使用； fileName: 文件名
返回值	当返回值为-1 时，发送失败，否则返回当前请求序号

示例	const char* servName = "test";
	//...中间省略初始化过程
	int nTx = 0;
	nTx = Send_GetFile(serName, "test.txt");

2.34. 注册读文件回调

函数声明	void TS_DLLEXPORT Reg_GetFileData_Resp(Svr_GetFileData callback);
功能	获取目录信息
参数	callback:回调函数
返回值	void
示例	//回调声明实现 void Svr_GetFileData_CallBack(const char* svrName,const char* dataFile){} Reg_GetFileData_Resp(Svr_GetFileData_CallBack);

2.35. 写文件

函数声明	int TS_DLLEXPORT Send_SetFile(const char* svrName,const char* fileName, const char* dataFile);
功能	发送写文件请求
参数	const char *svrName:连接从站的名字，作为当前连接标识使用； fileName: 文件名 const char* dataFile:需要发送的文件内容
返回值	0-成功 1-激活超时 2-传输超时 3-文件传输失败
示例	const char* servName = "test"; //...中间省略初始化过程 int nTx = 0; const char * dataFile = "abc"; const char *fileName = "test.txt";

	Send_SetFile(serName, fileName, dataFile);

2.36. 切换定值区

函数声明	int TS_DLLEXPORT Send_SelectActiveSG(const char* svrName,IEC_BYTE sgNum, IEC_DWORD address = 0x00, int waitResult = 1);
功能	发送切换定值请求
参数	const char *svrName:连接从站的名字，作为当前连接标识使用； sgNum: 定值区号 const char* dataFile:需要发送的文件内容 waitResult:1-等待请求响应 0-不等待请求响应 可根据开发情况使用
返回值	当返回值为-1 时，发送失败（当 waitResult = 1 时,为未响应），否则返回当前请求序号
示例	const char* servName = "test"; //...中间省略初始化过程 int nTx = 0; Send_SelectActiveSG(serName, 1);

2.37. 注册切换定值回调

函数声明	void TS_DLLEXPORT Reg_Svr_SelectActiveSG_Conf(Svr_SelectActiveSG_Conf callback);
功能	获取目录信息
参数	callback:回调函数
返回值	void
示例	//回调声明实现 void Svr_SelectActiveSG_Conf_CallBack(const char* svrName,IEC_SelectActiveSG_Conf *resp){} Reg_Svr_SelectActiveSG_Conf(Svr_SelectActiveSG_Conf_CallBack);

2.38. 读当前定值区号

函数声明	int TS_DLLEXPORT Send_ReadActiveSG(const char* svrName,IEC_DWORD address = 0x00, int waitResult = 1);
功能	发送读当前定值区号请求
参数	const char *svrName:连接从站的名字，作为当前连接标识使用；

	address:默认给 0 waitResult:1-等待请求响应 0-不等待请求响应 可根据开发情况使用
返回值	当返回值为-1 时，发送失败（当 waitResult = 1 时,为未响应），否则返回当前请求序号
示例	const char* servName = "test"; //...中间省略初始化过程 Send_ReadActiveSG(serName);

2.39. 注册读当前定值区号回调

函数声明	void TS_DLLEXPORT Reg_Svr_SelectActiveSG_Conf(Svr_SelectActiveSG_Conf callback);
功能	获取目录信息
参数	callback:回调函数
返回值	void
示例	//回调声明实现 void Svr_ReadActiveSG_Conf_CallBack(const char* svrName,IEC_ReadActiveSG_Conf *resp){} Reg_Svr_SelectActiveSG_Conf(Svr_SelectActiveSG_Conf_CallBack);

2.40. 读多个参数和定值

函数声明	int TS_DLLEXPORT Send_ReadMoreParam(const char* svrName,IEC_DWORD addressList[],IEC_BYTE addressSize, IEC_BYTE sgNum, int waitResult = 1);
功能	发送读多个参数和定值请求
参数	const char *svrName:连接从站的名字，作为当前连接标识使用； addressList[]:地址列表 addressSize: 地址个数 sgNum:定值区号 waitResult:1-等待请求响应 0-不等待请求响应 可根据开发情况使用
返回值	当返回值为-1 时，发送失败（当 waitResult = 1 时,为未响应），否则返回当前请求序号
示例	const char* servName = "test"; //...中间省略初始化过程 IEC_DWORD addressList[2]; addressList[0] = 0x8001;

	addressList[1] = 0x8020;
	int reqId = 0;
	reqId = Send_ReadMoreParam(serName, addressList, 2, 5);

2.41. 读全部参数和定值

函数声明	int TS_DLLEXPORT Send_ReadAllParam(const char* svrName,IEC_BYTE sgNum, int waitResult = 1);
功能	发送读全部参数和定值请求
参数	const char *svrName:连接从站的名字，作为当前连接标识使用； sgNum:定值区号 waitResult:1-等待请求响应 0-不等待请求响应 可根据开发情况使用
返回值	当返回值为-1 时，发送失败（当 waitResult = 1 时,为未响应），否则返回当前请求序号

示例	const char* servName = "test";
	//...中间省略初始化过程
	
	int reqId = 0;
	reqId = Send_ReadAllParam(serName, 5);

2.42. 读多个/全部参数和定值回调

函数声明	void TS_DLLEXPORT Reg_Svr_IEC_Read_Param_Resp(Svr_IEC_Read_Param_Resp callBack);
功能	获取多个/全部参数和定值信息
参数	callback:回调函数
返回值	void
示例	//回调声明实现 void Svr_IEC_Read_Param_Resp_CallBack(const char* svrName,IEC_Read_Param_Resp *resp){} Reg_Svr_IEC_Read_Param_Resp(Svr_IEC_Read_Param_Resp_CallBack);

2.43. 写多个参数和定值预置

函数声明	int TS_DLLEXPORT Send_WriteMoreParamPreSet(const char* svrName,DATA_VAL* valList[], int size, IEC_BYTE sgNum, int waitResult = 1);

功能	发送写多个参数和定值预置请求
参数	const char *svrName:连接从站的名字，作为当前连接标识使用； valList[]:设置参数列表 size:设置个数 sgNum:定值区号 waitResult:1-等待请求响应 0-不等待请求响应 可根据开发情况使用
返回值	当返回值为-1 时，发送失败（当 waitResult = 1 时,为未响应），否则返回当前请求序号

示例	const char* servName = "test";
	//...中间省略初始化过程
	
	int reqId = 0;
	DATA_VAL* valList[1];
	DATA_VAL data_val;
	data_val.dataType = DATA_BOOLEAN;
	data_val.data_Boolean = 1;
	valList[0] = &data_val;
	Send_WriteMoreParamPreSet(serName, valList, 1, 1);

2.44. 注册写多个/全部参数和定值预置 激活(激活终止)确认回调

函数声明	void TS_DLLEXPORT Reg_Svr_IEC_Write_Param_PreSet_Conf(Svr_IEC_Write_Param_PreSet_Conf callBack);
功能	获取多个/全部参数和定值信息
参数	callback:回调函数
返回值	void
示例	//回调声明实现 void Svr_IEC_Read_Param_Resp_CallBack(const char* svrName,IEC_Read_Param_Resp *resp){} Reg_Svr_IEC_Read_Param_Resp(Svr_IEC_Read_Param_Resp_CallBack);

2.45. 修改多个参数和定值固化或取消

函数声明	int TS_DLLEXPORT Send_ModMoreParamConfOrCancel(const char* svrName,IEC_BYTE ConfOrCancel, IEC_BYTE sgNum, int waitResult = 1);
功能	发送修改多个参数和定值固化或取消请求
参数	const char *svrName:连接从站的名字，作为当前连接标识使用； ConfOrCancel:固定、取消标识 1-固化 0-取消预置 sgNum:定值区号 waitResult:1-等待请求响应 0-不等待请求响应 可根据开发情况使用
返回值	当返回值为-1 时，发送失败（当 waitResult = 1 时,为未响应），否则返回当前请求序号
示例	const char* servName = "test"; //...中间省略初始化过程 Send_ModMoreParamConfOrCancel(serName, 1, 1);

2.46. 注册参数和定值的固定或取消确认回调

函数声明	void TS_DLLEXPORT Reg_Svr_IEC_Mod_Param_confOrCancel_Conf(Svr_IEC_Mod_Param_confOrCancel_Conf callBack);
功能	获取多个/全部参数和定值信息
参数	callback:回调函数
返回值	void
示例	//回调声明实现 void Svr_IEC_Mod_Param_confOrCancel_Conf_CallBack(const char* svrName,IEC_Mod_Param_confOrCancel_Conf *pResp){} Reg_Svr_IEC_Mod_Param_confOrCancel_Conf (Svr_IEC_Mod_Param_confOrCancel_Conf_CallBack);

2.47. 软件升级

函数声明	int TS_DLLEXPORT Send_UpLevel_Soft(const char* svrName,IEC_BYTE startOrEnd,IEC_BYTE isAct, IEC_DWORD address, int waitResult = 1);
功能	发送软件升级请求
参数	const char *svrName:连接从站的名字，作为当前连接标识使用； startOrEnd:软件升级启动/停止（0-启动 1-停止） isAct:激活停止（0-激活 1-停止激活） address:信息对象地址 waitResult:1-等待请求响应 0-不等待请求响应 可根据开发情况使用
返回值	当返回值为-1 时，发送失败（当 waitResult = 1 时,为未响应），否则返回当前请求序号
示例	const char* servName = "test"; //...中间省略初始化过程 Send_UpLevel_Soft(serName, 1, 1 , 0x0000);

2.48. 注册软件升级数据回调

函数声明	void TS_DLLEXPORT Reg_Svr_UpLevel_Soft(Svr_UpLevel_Soft callback);
功能	获取软件升级请求响应信息
参数	callback:回调函数
返回值	void
示例	//回调声明实现 void Svr_UpLevel_Soft_CallBack(const char* svrName,UpLevel_Soft *resp){} Reg_Svr_UpLevel_Soft(Svr_UpLevel_Soft_CallBack);