DNP3.0

主站开发包接口使用手册

版本：V2.0

深圳天勺电力软件有限公司

中国. 深圳

目录

1. 开发包简介

本开发包适用于 DNP3.0 规约，基于主站应用程序程序开发。规约参考 **D**istributed **N**etwork **P**rotocol 系列文件研发。**D**istributed **N**etwork **P**rotocol 由 Harris 公司提出，从 1993 年开始应用。近年来，DNP3.0 规约在我国的应用在逐渐上升，特别是大型综合自动化变电站采用国外的智能设备，大多数都要求采用 DNP 规约。

其中注册的函数，只需要注册一次即可。用户只需要关注了解一些基础以及 API 接口调用，就可以快速掌握 DNP 协议的相关应用，节约了客户从零研发的周期和成本。

开发包兼容 LINUX 以及 windows 系统。在使用源码编译时，需要注意以下几点：

1、 LINUX系统:在szts_DNPDataFile.h中,注释#define DNP30_IS_WINDOWS定义,
如果是windows则打开即可。

2、CMakeLists.txt 是生成动态库的文件，需要重新编译，可参考此文件。

3、在使用时，不想了解代码细节的话，直接看szts_DNP30Master.h中的接口即可。

2. 接口描述

2.1 连接到从站(TCP 模式)

函数声明	int TS_DNP30_DLL_EXPORT TS_DNP30_ConnectToTCPSlave(const char* svrName,const char *slave_IPAddr,const char *local_ip,unsigned short destAddr,unsigned short sourAddr,unsigned int port = 20000,unsigned int timeOut = 10);
功能	连接到从站
参数	const char *svrName：连接从站的名字，作为当前连接标识使用； slave_IPAddr:从站 IP 地址 local_ip:本机 IP destAddr:目的地址 sourAddr:源地址 port:端口号 TCP 模式一般默认为 20000

	timeOut:连接超时时间单位:秒
返回值	连接成功返回 1，否则为 0;

示例	const char *slaveName = "test";
	const char *slaveIp = "192.168.3.131";
	const char *localIp = "192.168.3.131";
	unsigned short destAddr = 4;
	unsigned short sourAddr = 3;
	int retVal = TS_DNP30_ConnectToTCPSlave(slaveName,slaveIp,localIp,destAddr, sourAddr);

Tip：当连接成功后，后续命令才能发送成功

2.2 连接到从站(串口模式)

函数声明	int TS_DNP30_DLL_EXPORT TS_DNP30_ConnectToComSlave(const char* svrName, unsigned short destAddr,unsigned short sourAddr,const char* portNo = "COM2", int baud = 9600, char parity = 'N',int databits = 8, int stopsbits = 1,unsigned int timeOut = 10);
功能	连接到从站
参数	svrName：连接从站的名字，作为当前连接标识使用 destAddr：目的地址 sourAddr：源地址 portNo:串口号，默认为 COM2,如果是 linux 一般为/dev/pts/2 形式 baud:波特率，默认 9600 parity:校验位 databits:数据位，默认 8 位 stopsbits:停止位 timeOut:连接超时时间 单位:秒
返回值	NULL

示例	const char* svrName = = "test";
	//关闭串口
	TS_IEC102_DisConnectCom(const char* svrName)

Tip：当连接成功后，后续命令才能发送成功

2.3 发送链路请求

函数声明	int TS_DNP30_DLL_EXPORT TS_DPN30_Send_LinkReq(const char* svrName,int linkFuncode);
功能	根据 linkFuncode，发送链路请求
参数	svrName：连接从站的名字，作为当前连接标识使用 linkFuncode:链路功能码,可参考 TS_DNP_E_LINK_FUNCOD 枚举值
返回值	返回目的地址
示例	const char* svrName = = "test"; //链路复位 TS_DPN30_Send_LinkReq (svrName, 0x00);

2.4 发送 GI 请求

函数声明	int TS_DNP30_DLL_EXPORT TS_DPN30_SendGIReq(const char* svrName,E_DNP_REQ_CODE giCode);
功能	获取 class0、class1、class2、class3 的值
参数	svrName：连接从站的名字，作为当前连接标识使用 giCode:召唤指定类别的数据
返回值	-1-发送失败 -2-有数据正在发送 其他发送成功，返回传输序号
示例	//读值class0、1、2、3的数据 const char *slaveName = "test"; TS_DPN30_SendGIReq(slaveName, E_DNP_REQ_CODE_CALSS0_1_2_3);

2.5 发送数据采集请求

函数声明	int TS_DNP30_DLL_EXPORT TS_DPN30_SendDataAcq(const char* svrName,TS_DATA_ACQ_Type dataType, TS_DNP_E_Var_Type varType,DNP_DWORD startAddr, DNP_DWORD stopAddr, DNP_BYTE isAllObj = 1);
功能	获取不同类型的数据
参数	svrName：连接从站的名字，作为当前连接标识使用 dataType:采集数据的类型 varType:变体，根据采集类型进行组合（具体看代码后面注释） startAddr:开始地址 stopAddr：结束地址 isAllObj：是否获取所有对象，为 1 时，开始、结束地址无效
返回值	-1-发送失败 -2-有数据正在发送 其他发送成功，返回传输序号

示例	const char *svrName = "test";　　//从站标识 程序自定义
	TS_DATA_ACQ_Type dataType;
	TS_DNP_E_Var_Type varType;
	DNP_DWORD startAddr = 0;
	DNP_DWORD stopAddr = 10;
	DNP_BYTE isAllObj = 1;
	//二进制输入
	dataType = TS_DATA_ACQ_BINARY;
	varType = TS_E_Var_1;
	//循环采集
	TS_DPN30_SendDataAcq(slaveName, dataType,
	varType,startAddr, stopAddr, isAllObj);
	Sleep(1000);
	//所有对象
	isAllObj = 0;
	//循环采集
	TS_DPN30_SendDataAcq(slaveName, dataType,
	varType,startAddr, stopAddr, isAllObj);

2.6 注册链路响应回调

函数声明	void TS_DNP30_DLL_EXPORT Reg_Svr_TS_DNP_Link_Struct_Resp(Svr_TS_DNP_Link_Struct_Resp callBack);
功能	获取链路的回调信息
参数	函数指针
返回值	void

示例	void Svr_TS_DNP_Link_Struct_Resp_CallBack(const char *server_name,
	TS_DNP_Link_Struct *pResp)
	{printf("ts_dnp_resp ts_dnp_resp.ts_link_info.funCode:%d\n",
	pResp->GetLinkInfo()->funCode);
	printf("ts_dnp_resp ts_dnp_resp.ts_link_info.desAddr:%d\n",
	pResp->GetLinkInfo()->desAddr);
	printf("ts_dnp_resp ts_dnp_resp.ts_link_info.sourAddr:%d\n",
	pResp->GetLinkInfo()->sourAddr);
	}Reg_Svr_TS_DNP_Link_Struct_Resp(Svr_TS_DNP_Link_Struct_Resp_CallB
	ack);

2.7 注册模拟量输入信息回调

函数声明	void TS_DNP30_DLL_EXPORT Reg_Svr_TS_DNP_AnalogInput(Svr_TS_DNP_AnalogInput callBack);
功能	获取模拟量输入信息
参数	函数指针
返回值	void

示例	//注册模拟量输入信息
	Void Svr_TS_DNP_AnalogInput_CallBack(const char *svrName,
	TS_DNP_AnalogInput *pResp){
	TS_DNP_AI_TYPE type = pResp->GetAIType();
	printf("voidSvr_TS_DNP_AnalogInput_CallBacktype :%d\n", type);
	switch(type){
	case TS_DNP_AIS_TYPE_INT32_WITH_FLAG:
	//32 位模拟量输入 Data Object 30－Variation 01 Type：Static
	{
	for (int i =0; i < pResp->GetIOCount(); i++){
	printf("[%d-%d]\n",pResp->dataInfo[i].addres,pResp->dataInfo[i].int32);
	}}}}}
	//注册模拟量输入信息
	Reg_Svr_TS_DNP_AnalogInput(Svr_TS_DNP_AnalogInput_CallBack);

2.8 注册模拟量输出信息回调

函数声明	void TS_DNP30_DLL_EXPORT Reg_Svr_TS_DNP_AnalogOutput(Svr_TS_DNP_AnalogOutput callBack);
功能	获取模拟量输出信息
参数	函数指针
返回值	无
示例	//模拟量输出信息 void Svr_TS_DNP_AnalogOutput_CallBack(const char *svrName,

	TS_DNP_AnalogOutput *pResp)
	{
	TS_DNP_AO_TYPE type = pResp->GetAOType();
	printf("CallBack:%d pResp->GetIOCount(): %d\n",
	type,pResp->GetIOCount());
	switch(type){
	case TS_DNP_AOS_TYPE_INT32_WITH_FLAG:
	//Analog Output Status - 16-bit with flag
	{
	for (int i =0; i < pResp->GetIOCount(); i++){
	printf("[%d-%d]\n", pResp->dataInfo[i].addres,
	pResp->dataInfo[i].int32);
	TS_DNP_Ang_Quality *pAngQuality =
	pResp->dataInfo->pAngQuality;
	if(pAngQuality){
	//品质判断//判断品质OnLine
	if(pAngQuality->GetOnLine()){
	printf("OnLine\n");}else{printf("Off Line\n");}}
	}
	}break;}}
	//注册模拟量输出信息
	Reg_Svr_TS_DNP_AnalogOutput(Svr_TS_DNP_AnalogOutput_CallBack);

2.9 注册 二进制输入信息 回调

函数声明	//注册 void TS_DNP30_DLL_EXPORT Reg_Svr_TS_DNP_BinaryInput(Svr_TS_DNP_BinaryInput callBack);
功能	获取二进制输入信息
参数	函数指针
返回值	void

示例	//二进制输入信息回调
	void Svr_TS_DNP_BinaryInput_CallBack(DNP_CPCHAR svrName,
	TS_DNP_BinaryInput *pResp)
	{
	//数据
	if(pResp->GetBIType() == TS_DNP_BIS_TYPE_SINGLE){
	//Single-Bit Binary Input
	printf("-----01 01 ----\n");
	for (int i =0; i < pResp->GetBICount(); i++)
	{
	printf("[%d-%d]", pResp->biInfo[i].addres,
	pResp->biInfo[i].bitStr[i]);

	}
	}
	printf("\n");
	}
	//二进制输入回调注册
	Reg_Svr_TS_DNP_BinaryInput(Svr_TS_DNP_BinaryInput_CallBack);

2.10 注册 二进制输出信息 回调

函数声明	void TS_DNP30_DLL_EXPORT Reg_Svr_TS_DNP_BinaryOutput(Svr_TS_DNP_BinaryOutput callBack);
功能	获取二进制输出信息
参数	函数指针
返回值	NULL

示例	//二进制输出信息回调函数
	void Svr_TS_DNP_BinaryOutput_CallBackFunc(DNP_CPCHAR svrName,
	TS_DNP_BinaryOutput *pResp)
	{
	//数据
	if(pResp->GetBOType() == TS_DNP_BOS_TYPE_WITH_QUASTATUS){
	//二进制输出状态
	printf("-----10 02 ----\n");
	for (int i =0; i < pResp->GetBICount(); i++)
	{
	printf("[%d-%d]", pResp->biInfo[i].addres,
	pResp->biInfo[i].bitStr[i]);
	}
	printf("\n");
	}
	}
	//注册二进制输出信息获取
	Reg_Svr_TS_DNP_BinaryOutput(Svr_TS_DNP_BinaryOutput_CallBackFunc);

2.11 发送遥控请求

函数声明	int TS_DNP30_DLL_EXPORT TS_DPN30_Send_YK(const char*

	svrName,TS_DNP_APP_E_FUNCODE ctrlMode,Ctrl_YK_Cmd* ctrlList[], int arrSize);
功能	发送遥控请求
参数	svrName：连接从站的名字，作为当前连接标识使用 ctrlMode：控制模式 选择或者执行 ctrlList:控制参数数组 arrSize：控制参数（数组）个数
返回值	-1-发送失败 -2-有数据正在发送 其他发送成功，返回传输序号

示例	const char * slaveName = "test"; //从站标识 程序自定
	//义 跟当前连接进行绑定
	TS_DNP_APP_E_FUNCODE ctrlMode = TS_DNP_APP_FC_DIR_OPERATE;
	Ctrl_YK_Cmd* ctrlList[2];
	Ctrl_YK_Cmd cmd1;
	cmd1.Init();
	cmd1.address = 0;
	cmd1.ctrlCode = TS_DNP_CtrlCode_LATCH_OFF;
	Ctrl_YK_Cmd cmd2;
	cmd2.Init();
	cmd2.ctrlCode = TS_DNP_CtrlCode_LATCH_ON;
	cmd2.address = 1;
	int arrSize = 2;
	ctrlList[0] = &cmd1;
	ctrlList[1] = &cmd2;
	TS_DPN30_Send_YK(slaveName,ctrlMode,ctrlList, arrSize);

详情看demo函数YDemo(const char * slaveName)

2.12 注册 遥控信息 回调

函数声明	void TS_DNP30_DLL_EXPORT Reg_Svr_TS_DNP_YK_Resp(Svr_TS_DNP_YK_Resp callBack);
功能	注册遥控信息回调
参数	函数指针
返回值	无

示例	void Svr_TS_DNP_YK_Resp_CallBack(DNP_CPCHAR svrName,
	TS_DNP_YK_Resp *pResp)
	{
	printf("Svr_TS_DNP_YK_Resp_CallBack callbak\n");
	for(int i = 0; i < pResp->GetPointSize(); i++){
	Ctrl_YK_Cmd *pCmd = pResp->pointList[i];
	if(pCmd){
	printf("ctrl status:%d\n",pCmd->ctrlStatus);
	printf("ctrl ctrlCode:%d\n",pCmd->ctrlCode);
	}

	}
	}
	//注册遥控信息回调
	Reg_Svr_TS_DNP_YK_Resp(Svr_TS_DNP_YK_Resp_CallBack);

2.13 发送遥调请求

函数声明	int TS_DNP30_DLL_EXPORT TS_DPN30_Send_YT(const char* svrName,TS_DNP_APP_E_FUNCODE ctrlMode,TS_DNP_YT_DataType dataType,Ctrl_YT_Cmd* ctrlList[] ,int arrSize);
功能	发送遥调请求
参数	svrName：连接从站的名字，作为当前连接标识使用 ctrlMode：控制模式 选择或者执行 ctrlList:遥调参数数组 arrSize：遥调参数（数组）个数
返回值	-1-发送失败 -2-有数据正在发送 其他发送成功，返回传输序号

示例	const char * slaveName = "test";
	//如果 TS_DNP_APP_FC_SELECT TS_DNP_APP_FC_OPERATE配套使用先选择，后执行
	TS_DNP_APP_E_FUNCODE ctrlMode = TS_DNP_APP_FC_DIR_OPERATE;
	//直接执行
	ctrlMode = TS_DNP_APP_FC_SELECT;//选择后 调用此参数
	int arrSize = 1;//实际应用 一般//只选择一个进行控制
	//遥调类型
	TS_DNP_YT_DataType ctrlDataType = TS_DNP_YT_DataType_INT16;
	//16bit 整型
	//ctrlDataType = TS_DNP_YT_DataType_FLOAT64;
	Ctrl_YT_Cmd* ctrlList[1];
	Ctrl_YT_Cmd cmd1;
	cmd1.Init();
	cmd1.address = 1;
	TS_DPN30_Send_YT(slaveName,ctrlMode,ctrlDataType,ctrlList,arrSize);

详情看demo函数YTDemo(const char * slaveName)

2.14 注册遥调信息回调

函数声明	void TS_DNP30_DLL_EXPORT Reg_Svr_TS_DNP_YT_Resp(Svr_TS_DNP_YT_Resp callBack);
功能	获取遥调信息
参数	函数指针

返回值	void

示例	//遥调信息回调
	void Svr_TS_DNP_YT_Resp_CallBack(DNP_CPCHAR svrName, TS_DNP_YT_Resp *pResp)
	{
	printf("Svr_TS_DNP_YT_Resp_CallBack callback\n");
	for(int i = 0; i < pResp->GetPointSize(); i++){
	TS_DNP_Ctrl_YT_Param *pCmd = pResp->pointList[i];
	if(pCmd){
	printf("ctrl status:%d\n",pCmd->ctrlStatus);
	switch(pCmd->dataType){
	case TS_DNP_YT_DataType_INT16: //16-bit integer //16-Bit Analog Output Block (Obj:41, Var:02)
	printf("val int16:%d\n", pCmd->int16);
	break;
	case TS_DNP_YT_DataType_INT32: //32-bit integer
	printf("val int32:%d\n", pCmd->int32);
	break;
	case TS_DNP_YT_DataType_FLOAT32: //32-bit //float
	printf("val float32:%f\n", pCmd->float32);
	break;
	case TS_DNP_YT_DataType_FLOAT64: //64-bit //double
	printf("val float64:%f\n", pCmd->float64);
	break;
	default:
	break;
	}
	}
	}
	}
	//注册遥调信息回调
	Reg_Svr_TS_DNP_YT_Resp(Svr_TS_DNP_YT_Resp_CallBack);

2.15 发送写请求

函数声明	int TS_DNP30_DLL_EXPORT TS_DPN30_Send_Write(const char* svrName, TS_DNP_Write_Request *pWriteData);
功能	发送写请求
参数	svrName：连接从站的名字，作为当前连接标识使用 pWriteData：写参数指针
返回值	-1-发送失败 -2-有数据正在发送 其他发送成功，返回传输序号

示例	void WriteDataDemo(const char * slaveName){
	TS_DNP_Write_Request writeData;
	TS_DNP_QualifierField qualifier;
	TS_DNP_Range range;
	range.Init();
	range.Start = 7;
	range.Stop = 7;
	writeData.Init();
	qualifier.Init();
	qualifier.pRange = ⦥
	qualifier.SetPreFixCode(TS_E_Qualifier_NO_INDEX);
	qualifier.SetRangeCode(TS_DNP_APP_E_QR_START_STOP_BYTE);
	TS_DNP_OBJECT obj;
	obj.Init();
	//Internal Indications demo
	//obj.object = TS_E_GROUP_TYPE_IIN;
	//obj.var = TS_E_Var_1;
	//obj.valType = TS_DNP_VALUE_TYPE_BITSTR;
	//obj.SetBitString("0", 1);
	//obj.pQulifier = &qualifier;
	//setting the time.
	range.size = 1;
	qualifier.SetPreFixCode(TS_E_Qualifier_NO_INDEX);
	qualifier.SetRangeCode(TS_DNP_APP_E_QR_SINGLE_FIELD_BYTE);
	obj.object = TS_E_GROUP_TYPE_TIME_DATE;
	obj.var = TS_E_Var_1;
	obj.valType = TS_DNP_VAL_TYPE_UTC;
	obj.SetUTCStr("2025-05-16 15:32:11.222", 23);
	obj.pQulifier = &qualifier;
	writeData.writeList[0]= &obj;
	writeData.SetWriteSize(1);
	TS_DPN30_Send_Write(slaveName, &writeData);
	}

2.16 退出当前主站连接

函数声明	void TS_DNP30_DLL_EXPORT TS_DNP30_ExitConnect(const char *svrName);
功能	退出当前连接
参数	svrName：连接从站的名字，作为当前连接标识使用

返回值	void

示例	const char *slaveName = "test"; //从站标识 程序自定义
	//跟当前连接进行绑定
	TS_DNP30_ExitConnect(slaveName);

2.17 注册连接断开回调

函数声明	void TS_DNP30_DLL_EXPORT Reg_Svr_TS_DNP_CONNECT_LOST(Svr_TS_DNP_CONNECT_LOST callBack);
功能	获取连接断开信息
参数	函数指针
返回值	void

示例	void Svr_TS_DNP_CONNECT_LOST_CallBack(DNP_CPCHAR svrName,
	DNP_BYTE RES)
	{
	}
	//注册断开连接回调
	Reg_Svr_TS_DNP_CONNECT_LOST(Svr_TS_DNP_CONNECT_LOST_CallBack);

2.18 注册接收报文回调

函数声明	void TS_DNP30_DLL_EXPORT Reg_Svr_TS_DNP_parseReceMsg(Svr_TS_DNP_parseReceMsg callBack);
功能	获取接收
参数	函数指针
返回值	void

示例	//接收报文回调
	void Svr_TS_DNP_parseReceMsg_CallBack(const char *server_name,
	struct TS_DNP_MsgInfo *msgInfo)
	{
	}
	//接收报文
	Reg_Svr_TS_DNP_parseReceMsg(Svr_TS_DNP_parseReceMsg_CallBack);

2.19 注册发送报文回调

函数声明	void TS_DNP30_DLL_EXPORT Reg_Svr_TS_DNP_parseSendMsg(Svr_TS_DNP_parseSendMsg callBack);
功能	获取发送报文

参数	函数指针
返回值	void

示例	//发送报文回调
	void Svr_TS_DNP_parseSendMsg_CallBack(const char *server_name, struct TS_DNP_MsgInfo *msgInfo)
	{
	} //发送报文
	Reg_Svr_TS_DNP_parseSendMsg(Svr_TS_DNP_parseSendMsg_CallBack);

2.20 注册计数器信息回调

函数声明	void TS_DNP30_DLL_EXPORT Reg_Svr_TS_DNP_COUNTER(Svr_TS_DNP_COUNTER callBack);
功能	获取计数器信息
参数	函数指针
返回值	void

示例	void Svr_TS_DNP_COUNTER_CallBack(DNP_CPCHAR svrName, TS_DNP_COUNTER *pResp)
	{
	for (int i =0; i < pResp->GetIOCount(); i++)
	{
	if(pResp->GetCNType() == TS_DNP_CN_TYPE_UINT32_WITH_FLAG){
	QString msg = "Data Object20 ---Variation :01 Type:Static ";
	char buf[100];
	sprintf(buf, "[%d-%d-%s] ", pResp->dataInfo[i].addres, pResp->dataInfo[i].uInt32, pResp->dataInfo[i].pCountQuality->GetBitString());
	msg += buf;
	}else if(pResp->GetCNType() == TS_DNP_CN_TYPE_UINT16_WITH_FLAG){
	QString msg = "Data Object20 ---Variation :02 Type:Static ";
	char buf[100];
	sprintf(buf, "[%d-%d-%s] ", pResp->dataInfo[i].addres, pResp->dataInfo[i].uInt16,
	pResp->dataInfo[i].pCountQuality->GetBitString());
	msg += buf;
	}else if(pResp->GetCNType() ==

```
TS_DNP_CN_TYPE_UINT32_WITHOUT_FLAG){
            QString msg = "Data Object20 ---Variation :05
Type:Static ";
            char buf[100];
            sprintf(buf, "[%d-%d] ",
pResp->dataInfo[i].addres, pResp->dataInfo[i].uInt32);
            msg += buf;
        }else if(pResp->GetCNType() ==
TS_DNP_CN_TYPE_UINT16_WITHOUT_FLAG){
            QString msg = "Data Object20 ---Variation :06
Type:Static ";
            char buf[100];
            sprintf(buf, "[%d-%d] ",
pResp->dataInfo[i].addres, pResp->dataInfo[i].uInt32);
            msg += buf;
        }
    }
}
 //注册计数器信息回调
Reg_Svr_TS_DNP_COUNTER(Svr_TS_DNP_COUNTER_CallBack);
```