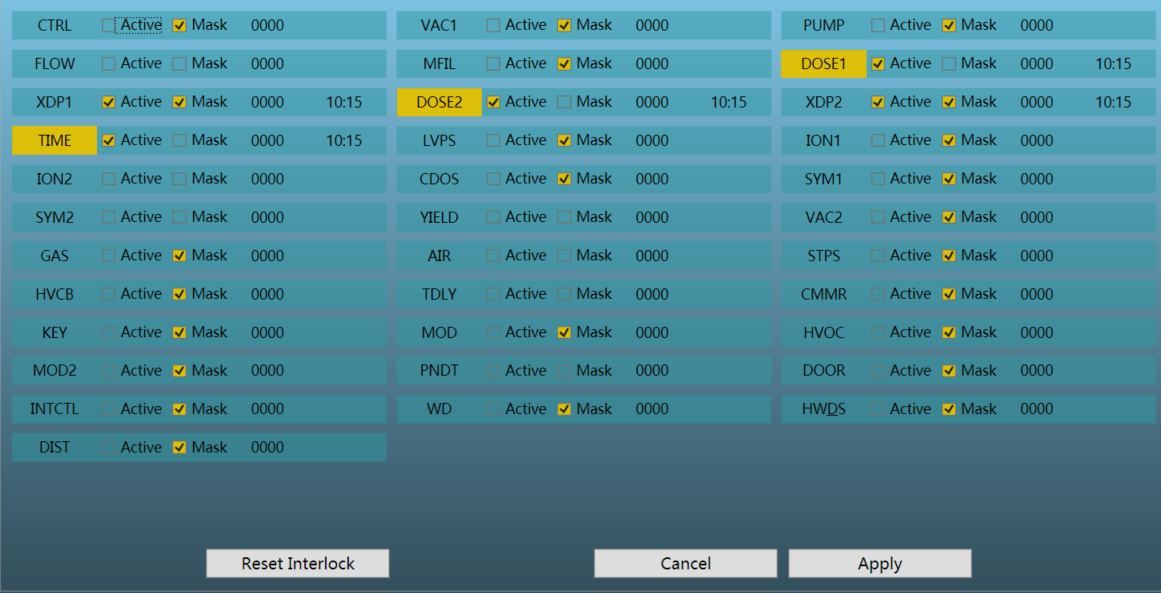
连锁页面

一、页面截图



二、 相对应的连锁表

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| CTRL | 主要 | 自检 |  | VAC1 | 主要 | 真空连锁 |  | PUMP | 主要 | 水泵 |
| FLOW | 主要 | 水流量 |  | MFIL | 主要 | 速调管真空电流 |  | DOSE1 | 剂量连锁 | 治疗结束 |
| XDP1 | 剂量连锁 | 枪靶方向单脉冲剂量超 或剂量率超范围 |  | DOSE2 | 计量连锁 | MU2超范围 |  | XDP2 |  |  |
| TIME | 剂量连锁 | 时间超时 |  | LVPS | 次连锁 | 剂量 |  | ION1 | 次要连锁 | 电离室左右方向电压 |
| ION2 | 次要连锁 | 电离室枪靶方向 电压 |  | CDOS |  |  |  | SYM1 |  |  |
| SYM2 |  |  |  | YIELD |  |  |  | VAC2 | 次要连锁 | 不严重真空漏 |
| GAS | 次要连锁 | 波导气压 |  | AIR | 次要连锁 | 气压 |  | STPS | 次要连锁 | Stand By 电源 |
| HVCB | 次要连锁 | 短路器 |  | TDLY | 次要连锁 | 延时 |  | CMMR |  |  |
| KEY | 次要连锁 | 要吃 |  | MOD | 次要连锁 | 调制器阻抗 |  | HVOC | 次要连锁 | 高压过流 |
| MOD2 | 次要连锁 | 能量板 |  | PNDT | 次要连锁 | 手控盒 |  | DOOR | 次要连锁 | 门 |
| INTCTL |  |  |  | WD |  |  |  | HWDS |  |  |
| DIST |  |  |  |  |  |  |  |  |  |  |

相对应的代码：

//连锁定义

enum ILType

{

ILType\_NULL = -1,

//主联锁

ILType\_CTRL,

ILType\_VAC1,

ILType\_PUMP,

ILType\_FLOW,

ILType\_MFIL,

//剂量联锁

ILType\_DOSE1,

ILType\_XDP1,

ILType\_DOSE2,

ILType\_XDP2,

ILType\_TIME,

//次联锁

ILType\_LVPS,

ILType\_ION1,

ILType\_ION2,

ILType\_CDOS,

ILType\_SYM1,

ILType\_SYM2,

ILType\_YIELD,

ILType\_VAC2,

ILType\_GAS,

ILType\_AIR,

ILType\_STPS,

ILType\_HVCB,

ILType\_TDLY,

ILType\_CMMR,

ILType\_KEY,

ILType\_MOD,

ILType\_HVOC,

ILType\_MOD2,

ILType\_PNDT,

ILType\_DOOR,

ILType\_INTCTL,

ILType\_WD,

ILType\_HW\_DS,

ILType\_DIST,//剂量分布联锁 吸收剂量分布畸变超过10%或辐射探测器吸收剂量分布探测信号指示变化大于10%，在增加吸收剂量达到0.25Gy之前

ILType\_COUNT

};

三、页面属性：

第一列：连锁名称

第二列：屏蔽状态 （bool8 m\_mask; //屏蔽状态）

第三列：活动状态 （bool8 m\_actived; //活动状态）

第四列：CTRL 子码

第五列 ：当前时间

说明：当选中Active活动状态开启，将参数保存到相应文件中去、并且连锁名称变为黄色

并记录勾选时间，选中Mask激活屏蔽状态。

点击 Apply 将各个连锁选中的状态下发到下位机中。

点击Reset 将各个连锁状态复位到初始状态。

四、点击各个按钮所涉及到的代码逻辑。

相应的逻辑代码：

1、相应的数据类型、命令类型：

CommDataType\_NoData = 0, //无数据

CMD\_SetInterLockMask\_V2M,//设置联锁屏蔽标志

CMD\_SetInterLockActive\_V2M,//设置联锁活动状态

CMD\_SaveInterLockSetting\_V2M,//保存联锁设置参数

CMD\_CancelInterLockSetting\_V2M,//取消联锁参数设置

CommDataType\_NoData = 0, //无数据

CommDataType\_InterLockMask,//联锁屏蔽

CommDataType\_InterLockActive,//联锁活动状态

存在于枚举LCommCommand中：

enum LCommCommand

{

CMD\_CommCommand\_NULL = 0,

CMD\_HeartBeat\_V2M\_M2V, //心跳报文。双向命令。//以下为应用程序所有命令

//第1组命令 控制台到控制器的命令

CMD\_SetWorkMode\_V2M, //设定工作模式

CMD\_SetBeamType\_V2M, //设置Beam类型

CMD\_SetEnergyType\_V2M,//设置能量类型

CMD\_SetDoseRate\_V2M,//设置剂量率类型

CMD\_SetTreatmentCMu1\_V2M,//设置剂量

CMD\_SetTreaSettingFlag\_V2M ,//设定参数完毕标志

CMD\_SetInterLockMask\_V2M,//设置联锁屏蔽标志

CMD\_SetInterLockActive\_V2M,//设置联锁活动状态

CMD\_SaveInterLockSetting\_V2M,//保存联锁设置参数

CMD\_CancelInterLockSetting\_V2M,//取消联锁参数设置

CMD\_SetOutputIOBitLevel\_V2M,//设置io电平

CMD\_SetDACChannelValue\_V2C, //设置DA通道的数值 2017-3-27

CMD\_SetADChannelValue2VoltPara\_V2M, //设置ADC数值到电压的计算参数 2017-4-1

CMD\_SetDAChannelVolt2ValuePara\_V2M, //设置DAC通道电压到数值的计算参数 2017-4-1

CMD\_SetCounterSetting\_V2M,//设置计数器

CMD\_ResetInterlockLatch\_V2M,//联锁锁存器复位

CMD\_SetSubscribeCycleEvent\_V2M, //订阅周期性发生的事件

CMD\_SetPlanBeamData\_V2M, //设定计划治疗数据

CMD\_SetMotionControlPara\_V2M, //设定运动控制参数2017-3-20

CMD\_MotionCommand\_V2C,//运动控制命令 2017-3-23

CMD\_MotionAxisIOControl\_V2C,//运动轴的IO控制

CMD\_MotionAxisPostionPara\_V2M,//设置运轴的位置参数 2017-3-24

CMD\_MotionAxisSpeedPara\_V2M,//设置运轴的速度参数 2017-3-24

CMD\_SetMotionAxisMotionMode\_V2M,//设置运动轴的运动控制模式 2017-3-25

CMD\_SetProgramRunSetting\_V2M,//设置程序的运行模式 2017-3-29

CMD\_SetPhysicsSetting\_V2M, //设置物理参数 2017-6-8

//第2组命令 控制器到控制台的命令

DUpdate\_Mainstatus, //更新主状态

DUpdate\_WorkMode,// 更新工作模式

DUpdate\_MessageCode,//更新信息代码

DUpdate\_ControllerBaseData, //更新控制器基本数据

DUpdate\_PlanBeamData\_M2V, // 更新计划治疗数据

DUpdate\_ActualBeamData\_M2V, //显示所有实际治疗数据

DUpdate\_Interlocks\_M2V, //更新联锁

DUpdate\_DIO\_M2V,//更新dio数据

DUpdate\_ADC\_M2V,//更新adc数据

DUpdate\_DAC\_M2V,//更新dac数据 2017-3-29

DUpdate\_CORDERS\_M2V,//更新编码器数据 2017-4-30

DUpdate\_ADChannelValue2VoltParas\_M2V, //更新ADC所有通道数值到电压的计算参数 2017-4-1

DUpdate\_DAChannelVolt2ValueParas\_M2V, //更新DAC所有通道电压到数值的计算参数 2017-4-1

DUpdate\_Counter\_M2V,//更新计数器数据

DUpdate\_MotionControlPara\_V2M,//更新运动轴的控制数据 2017-3-21

DUpdate\_AxisPosCalDatas\_V2M,//更新位置标定数据 2017-3-22

DUpdate\_AxisMathParas\_V2M,//更新轴的数学参数 2017-3-22

DUpdate\_AxisControlData\_V2M,//更新轴控制数据 2017-3-23

DUpdate\_AxisSpeedCalData\_V2M,//更新速度标定数据2017-3-23 pm

DUpdate\_AxisLimitSwitchStatus\_V2M,//更新限位开关状态 2017-3-23 pm

DUpdate\_ProgramRunSetting\_V2M,//更新程序运行模式控制数据 2017-3-29

DUpdate\_UpPhysicsData\_M2V, // 2016-06-09

CMD\_UpLayerSend\_Confirm\_M2V, //上层发送确认。收到命令后发出，仅表示通讯成功。

//以下添加服务应用程序所用命令（略）2016-9-29

CommCommand\_Count //命令个数

}; //注这个数据类型发生修改时，对应的AppComDataDef.cs中的CS\_LCommCommand也要发生修改

//通讯数据类型

enum CommDataType

{

CommDataType\_NoData = 0, //无数据

CommDataType\_WorkMode,//工作模式

CommDataType\_BeamType,//Beam类型

CommDataType\_EnergyType,//能量类型

CommDataType\_DoseRate,//剂量率

CommDataType\_TreatCMu1,//治疗数据CMu1

CommDataType\_TreatSetFlag,//治疗数据设定状态

CommDataType\_InterLockMask,//联锁屏蔽

CommDataType\_InterLockActive,//联锁活动状态

CommDataType\_OutputIOBit,//输出IO电平

CommDataType\_DACChannelData,//某一DAC通道数据

CommDataType\_CordeslData,//某一道的编码器数据 2017-4-26

CommDataType\_ADChannelValue2VoltPara,//ADC数值到电压的计算参数 2017-4-1

CommDataType\_DAChannelVolt2ValuePara,//DAC通道电压到数值的计算参数 2017-4-1

CommDataType\_CounterSetting,//计数器设定

CommDataType\_MotionAxisSettings,//运动轴控制参数

CommDataType\_MotionAxisMotionCommandPara,//运动轴运动控制命令数据 2017-3-24

CommDataType\_MotionAxisIOs,//运动轴IO 2017-3-24

CommDataType\_MotionAxisPosPara,//运动轴的位置计参数 2017-3-25

CommDataType\_MotionAxisSpeedPara,// 运动轴速度参数计算参数 2017-3-24

CommDataType\_MotionControlMode,//运动轴的运动控制模式 2071-3-25

CommDataType\_SetProgramRunSetting,//程序的运行模式控制 2017-3-29

//第2组命令 控制器到控制台的命令

CommDataType\_MotionAxisPosCalData,//运动轴位置标定数据

CommDataType\_MotionAxisMathParas, //某一个运动轴的数学计算参数（包含位速度与位置）

CommDataType\_MotionAxisControlData, //某一个运动轴的控制数据

CommDataType\_MotionAxisSpeedCalData,//更新速度标定数据 207-3-23 pm

CommDataType\_MotionAxisLimitSwitchStatus,//更新限位开关状态 207-3-23 pm

CommDataType\_PlanBeamData, //设置计划治疗数据

CommDataType\_ActualBeamData,//Beam数据, 包括几何数据

CommDataType\_InterLocks,//联锁数据

CommDataType\_SubscribeCycleEvents, //事件订阅

CommDataType\_MainStatus,//主状态

CommDataType\_MessageCodeData,//信息代码

CommDataType\_ControllerBase,//控制器基本数据

CommDataType\_DIOs,//Dio数据

CommDataType\_ADC\_Values,//ADC数据

CommDataType\_ADC\_Volts,//ADC电压值

CommDataType\_DAC\_Values,//DA数值 2017-3-27

CommDataType\_DAC\_Volts,//DA电压值 2017-3-27

CommDataType\_ADValue2VoltParas,//AD所有通道计算参数 2017-4-1

CommDataType\_DAValue2VoltParas,//DA所有通道计算参数 2017-4-1

CommDataType\_PhysicsSetting,//物理参数 2017-6-8

CommDataType\_Counters, //计数器

CommDataType\_Confirm, //确认报文

//上下都用到的数据类型

CommDataType\_Count

}; //注意：CommDataType 这个枚类型与CommDataTypeSize[]数组是有联带关系的，修改其中一个，另一个必须对应的修改

2、选中某一个连锁的屏蔽按钮的代码逻辑：

//屏蔽连锁时所涉及到的代码：

// 对上位机发出的命令做出反应

bool8 LInterLockCtr::ProcessCommCommand(LCommCommand commCommand,

CommDataType dataType,

void \*commData)//2017-4-2

{

bool8 ok=false;

switch( commCommand )

{

case CMD\_SetInterLockMask\_V2M://设置联锁屏蔽标志

if( dataType == CommDataType\_InterLockMask )

{

SetInterLockMask\_V2M( (ILType)((CDT\_InterLockMask\*)commData)->interLockType,

((CDT\_InterLockMask\*)commData)->maskFlag);

}

else

{

//数据类型错误

}

ok = true;

break;

}

//更新数据、针对每一个连锁设置其屏蔽状态。

void LInterLockCtr::SetInterLockMask\_V2M( ILType interLockType, bool8 maskFlag)//设置联锁屏蔽标志

{

m\_pInterLockModel->m\_ppInterLocksTable[interLockType]->m\_mask = maskFlag;

}

class far CDT\_InterLockMask//:public ICommData

{

public:

ENUM16 interLockType;//ILType interLockType;

bool8 maskFlag;

};

3、选中某一个连锁的活动按钮的代码逻辑：

// 设置联锁活动状态：

// 对上位机发出的命令做出反应

case CMD\_SetInterLockActive\_V2M://设置联锁活动状态

if( dataType == CommDataType\_InterLockActive )

{

m\_pThis->SetInterLockActive\_V2M( (ILType)((CDT\_InterLockActive\*)commData)->interLockType,

((CDT\_InterLockActive\*)commData)->activeStatus);

}

else

{

//数据类型错误

}

ok = true;

break;

//更新数据、针对每一个连锁设置其活动状态并保存的模型中去。

void LInterLockCtr::SetInterLockActive\_V2M( ILType interLockType, bool8 activeStatus )//设置联锁活动状态

{

m\_pInterLockModel->m\_ppInterLocksTable[interLockType]->m\_actived = activeStatus;

}

// 数据的存储类型

class far CDT\_InterLockActive//:public ICommData

{

public:

ENUM16 interLockType;//ILType interLockType;

bool8 activeStatus;

};

4、点击Apply的代码逻辑：

//接收到保存命令是的应激反应。

case CMD\_SaveInterLockSetting\_V2M://保存联锁设置参数

if( dataType == CommDataType\_NoData )

{

m\_pThis->SaveInterLockSetting\_V2M();

}

else

{

//数据类型错误

}

ok = true;

break;

// 保存数据

void LInterLockCtr::SaveInterLockSetting\_V2M()//保存联锁设置参数

{

m\_pInterLockModel->SaveCFGFile();

}

//将模型中的数据保存到文件中去。

bool8 LInterLockModel::SaveCFGFile()

{

FILE \*fp = fopen( INTERLOCK\_CFG\_FILE\_PATH,"wb");

if( fp != NULL)

{

for( int16 i = 0; i < ILType\_COUNT; i++ )

{ m\_ppInterLocksTable;

fwrite( m\_ppInterLocksTable[i], sizeof(ILTypes),1,fp); //提示信息，配置文件保存成功!

m\_ppInterLocksTable[i]->m\_class;

m\_ppInterLocksTable[i]->m\_value;

m\_ppInterLocksTable[i]->m\_name;

}

fclose(fp);

return true;

}

else

{

return false;

}

}

5、点击复位按钮的代码逻辑：

//接收到命令的应激反应代码：

case CMD\_ResetInterlockLatch\_V2M: //复位联锁锁存器

if( dataType == CommDataType\_NoData )

{

RestIOInterLock();

}

else

{

//数据类型错误

}

ok = true;

break;

default:

ok = false;

break;

}

return ok;

}

void LInterLockCtr::RestIOInterLock()//复位io所读联锁

{

m\_pHardwaredDriver->m\_pDioBoard->ResetIOInterLock();

m\_piARTModelInf->SendMsgCode\_C2M(MessageObjectTypeSystem,MessageGrade\_Info, MsgCode\_ILCtr\_ResetILLatch );

}

//复位其所涉及到的IO电平

void LDioBoard::ResetIOInterLock() //复位联锁

{

//先置为低电平清除锁存器

for( int16 i=0;i < 100; i++ )

{

setOutputIOBitLevel( OutIOType\_MjFltRet,false);

setOutputIOBitLevel( OutIOType\_BoFltRet,false);

setOutputIOBitLevel( OutIOType\_AxFltRet,false);

setOutputIOBitLevel( OutIOType\_SuFltRet,false);

setOutputIOBitLevel( OutIOType\_DsFltRet,false);

setOutputIOBitLevel( OutIOType\_CrDoseRet,false); // 2016-10-24加入

}

//延时1ms

// LPc8254Clock::Delay( 1 );

//最后置为高电平，恢复锁存器功能

setOutputIOBitLevel( OutIOType\_MjFltRet,true);

setOutputIOBitLevel( OutIOType\_BoFltRet,true);

setOutputIOBitLevel( OutIOType\_AxFltRet,true);

setOutputIOBitLevel( OutIOType\_SuFltRet,true);

setOutputIOBitLevel( OutIOType\_DsFltRet,true);

setOutputIOBitLevel( OutIOType\_CrDoseRet,true);

setIoInterLockLevel( ILType\_FLOW, false ); //2012-12-14添加,流量联锁单独处理

}

void LiARTModelInf::SendMsgCode\_C2M(MessageObjectType type,MessageGrade grade, MsgCodeType code)

{

MessageCodeData codeData;

for( int16 i = 0; i < MessageChildCodeCount; i++ )

{

codeData.m\_childCode[i] = -1;

}

6、页面显示的代码

void LInterLockCtr::ProcessUpdateDataTask()

{

//更新联锁

if( m\_piARTModelInf->m\_pSubscribeCycleEvents->ExistSubscribeCycleEvent( DUpdate\_Interlocks\_M2V ) )

{

UpdateInterLocks\_M2V( m\_pInterLockModel->m\_ppInterLocksTable );

}

}

void LInterLockCtr::UpdateInterLocks\_M2V( ILTypes\*\* m\_ppInterLocksTable ) //上传联锁数据

{

CDT\_InterLocks far\* pData = new CDT\_InterLocks();

for( int16 i = 0; i < ILType\_COUNT; i++ )

{

bool8 act = m\_ppInterLocksTable[i]->m\_actived;

bool8 mask = m\_ppInterLocksTable[i]->m\_mask;

byte childCode = (byte)(m\_ppInterLocksTable[i]->m\_iLCode);

pData->SetInterlock(i, act, mask, childCode );

}

m\_pCommSender->SendMessage2( DUpdate\_Interlocks\_M2V, CommDataType\_InterLocks, pData );

delete pData;

}

void CDT\_InterLocks::SetInterlock( int16 index, bool8 active, bool8 mask, byte cCode )

{

int16 workIndex = index / 8;

int16 bitIndex = index % 8;

if( active )

{

activeSta[ workIndex ] |= (0x01 << bitIndex);

}

else

{

activeSta[ workIndex ] &= ~(0x01 << bitIndex);

}

if( mask )

{

maskSta[ workIndex ] |= (0x01 << bitIndex);

}

else

{

maskSta[ workIndex ] &= ~(0x01 << bitIndex);

}

childCode[ index ] = cCode;

}

class far CDT\_InterLocks //联锁

{

public:

byte activeSta[ ILType\_COUNT/8 + ((ILType\_COUNT%8 > 0)?1:0) ];

byte maskSta[ ILType\_COUNT/8 + ((ILType\_COUNT%8 > 0)?1:0) ];

byte childCode[ ILType\_COUNT ];

public:

CDT\_InterLocks();

void SetInterlock( int16 index, bool8 active, bool8 mask, byte childCode );

void GetInterlock( int16 index, bool8& active, bool8& mask, byte& cCode );

};