em\_plc详细设计文档

1. **概述**

em\_plc是为了替代传统PLC设备功能而设计的模块，主要目的是对收到的模式控制命令进行处理，模式控制前需要反馈给数据底座，然后下发控制相应设备的命令。

本文档基于IotPLC设计文档-V1.8.docx文档，对接口和处理逻辑进行细节性补充，为后续的开发工作提供指导依据。

1. **数据定义**

//控制信息，用于下发控制，与plc\_iotdevice共用

typedef struct

{

string szProductKey; //大平台iot中的产品key

string szDeviceKey; //大平台iot中的设备key

string szPropertyCode; //大平台设备中的属性代码

string szValue； //属性值

}MQTTCMD;

//模式信息，与plc\_iotdevice共用

typedef struct

{

string szSystemName; //系统key

int nMode; //子模式号

int nOpt; //操作模式,0-复位；1-触发

long long nTime; //时间戳，毫秒

}MQTTMODE;

//用于保存火灾来源表信息

typedef struct

{

string szFireSource; //火灾来源

int nLevel； //优先级

}FIREINFO;

//用于保存子模式表信息

typedef struct

{

string szDevice; //控制设备名称

string szProduct; //产品key

string szCtrlProp; //控制属性

int nCtrlValue; //控制值

string szExpectProp; //期望属性

int nExpectValue; //期望值

string szInterLock; //闭锁脚本

int nLockTime; //闭锁超时阈值

int nDelayTime; //延迟时间

bool bExecuted; //0-未执行；1-已执行

int nFirstDelayTime; //记录第一次延迟的时间戳，秒

int nFirstLockTime; //记录第一次闭锁检测失败的时间戳，秒

}MODEINFO;

//各个环控系统的运行信息

typedef struct

{

string szSystem; //环控系统，只会是环控系统表的系统代码

MQTTMODE mqMode; //模式，收到的模式控制

unsigned int nLevel; //优先级，0表示正常模式，非0表示火灾来源优先级

int nHandle; //线程号

bool bExitThread; //线程退出标志

int nTimeOut; //整个模式的执行超时阈值，秒

list<MODEINFO> lsCtrlDev; //模式控制的所有设备列表

}RUNNINGINFO;

1. **接口**
2. getSystemCmd（MQTTMODE &info）：获取模式下控的命令。参数：空值的结构体引用info。返回值：不为空值的结构体info。
3. getDeviceInfo（char \* device,char \* property,int & type,string& value）;获取设备信息的属性值，脚本函数需要增加此函数。参数：设备名字串device、属性名字串property、类型的引用type和属性值的引用。返回：属性值类型type，type=1表示AI，type=3表示DI，属性值value。
4. setSystemReturn(vector<MQTTMODE> info);将模式结果反馈给plc\_iotdevice。参数：多个模式信息的vector数组。无返回值。
5. setDeviceProperty（MQTTCMD info）；下发正常的设备属性控制。参数：控制信息结构体info，无返回值。
6. int getHost()：获取当前机器的主备状态：返回值：1-主，0-备。
7. **主要函数**
8. MAINLOOP:主进程函数，无参数。While循环，睡眠时间1秒，实现定时效果。
9. modeCtrl(MQTTMODE info):模式控制处理函数，负责接收到模式控制命令后进行逻辑判断。参数：模式控制信息info。无返回值。
10. fireModeCtrl(MQTTMODE info):处理火灾模式的触发控制。参数：模式控制信息info。无返回值。
11. normalModeCtrl(MQTTMODE info):处理正常模式的触发控制。参数：模式控制信息info。无返回值。
12. resetMode(MQTTMODE info):处理火灾的复位控制。参数：模式控制信息info。无返回值。
13. getCtrlDevice(RUNNINGINFO info):从子模式表中根据系统代码和子模式号获取所有的控制设备以及控制信息。参数：运行信息info。无返回值。
14. modeCtrlThread(void\* argp):线程执行函数。参数: RUNNINGINFO\* 的void\*格式。
15. **详细设计**
16. 初始化，将火灾来源表存储在数组list< FIREINFO >，将环控系统表存储在数组vector <string >中。所有环控系统运行状态保存在list< RUNNINGINFO >里，初始化时所有环控系统的运行信息RUNNINGINFO里，除了系统szSystem的值为环控系统表的系统代码外，所有的参数均为空值。
17. 定时器函数MAINLOOP ()里，首先通过接口getHost()获取当前机器的主备状态，如果不等于1说明不是主机，不做任何处理。如果等于1，说明是主机，然后通过接口getSystemCmd（MQTTMODE &info）获取模式控制命令。先取到info里的时间nTime，如果比当前时间晚10s以上或则比当前时间还早则不做处理。获取到的info里的系统代码szSystemName为空，说明无模式控制命令，此时不做处理；获取到的info里的szSystemName不为空，说明获取到模式控制命令，调用主处理函数modeCtrl(MQTTMODE info)。
18. 在模式控制modeCtrl(MQTTMODE info)函数里，通过参数info里的szSystemName得到需要控制的系统；nMode得到子模式号；nOpt得到操作模式，0代表复位，1代表触发。后面进行逻辑处理：
    1. 判断szSystemName在list< FIREINFO >的szFireSource中是否匹配到，匹配到，说明此次模式控制是火灾模式，nOpt为1走3.2，nOpt为0走3.3；匹配不到，说明不是火灾模式，然后在vector<string >中匹配，匹配到走3.4，匹配不到，日志记录(未匹配到任何模式)，退出函数。
    2. 火灾模式控制，调用fireModeCtrl(MQTTMODE info)。
    3. 火灾复位控制，调用resetMode(MQTTMODE info)。
    4. 正常模式控制，调用normalModeCtrl(MQTTMODE info)。
19. 火灾模式控制fireModeCtrl(MQTTMODE info)
    1. 火灾触发控制，例如：szSystemName=“ISCS”,nMode=1,nOpt=1。
       1. 先由系统代码ISCS和1，去火灾模式系统表取执行超时阈值nTimeOut、所属系统(例如：TK)和影响系统(例如：TVS,K1)。由找到的所属系统去list< RUNNINGINFO >里找到当前火灾模式对应的环控系统szSystem，判断该环控系统的运行模式nLevel，nLevel>0表示该环控系统的运行模式为火灾模式；nLevel=0表示该环控系统的运行模式为正常模式。
       2. 环控系统正在运行火灾模式：根据传入的ISCS从list< FIREINFO >取出对应的level1。
          1. 如果level1<nLevel，将正在执行的模式线程nHandle终止，将火灾模式、所属系统和影响系统的参数填入vector<MQTTMODE>中{szSystemName=”ISCS”,nMode=1,nOpt=1},{szSystemName=”ISCS-TK”,nMode=1,nOpt=1},{szSystemName=”TK”, nMode =0, nOpt =1},{ szSystemName=”TVS”, nMode =0, nOpt =1},{ szSystemName=”K1”, nMode =0, nOpt =1}，通过接口setSystemReturn(vector<MQTTMODE> info)发送给plc\_iotdevice。将RUNNINGINFO的信息换成szSystemName=“ISCS”,nMode=1,nOpt=1，nLevel=level1，重新创建线程。日志记录(优先级高，模式覆盖)，调用getCtrlDevice(RUNNINGINFO)。
          2. 如果level1>nLevel，忽略，日志记录(优先级低，模式忽略)，退出函数。
          3. 如果level1=nLevel，忽略，日志记录(优先级相同，未复位)，退出函数。
       3. 环控系统正在运行正常模式：将正常模式线程nHandle终止，将火灾模式、所属系统和影响系统的参数填入vector<MQTTMODE>中{szSystemName=”ISCS”,nMode=1,nOpt=1},{szSystemName=”ISCS-TK”,nMode=1,nOpt=1},{szSystemName=”TK”, nMode =0, nOpt =1},{ szSystemName=”TVS”, nMode =0, nOpt =1},{ szSystemName=”K1”, nMode =0, nOpt =1}，通过接口setSystemReturn(vector<MQTTMODE> info)发送给plc\_iotdevice。根据传入的ISCS从list< FIREINFO >取出对应的level1，然后将RUNNINGINFO的信息换成szSystemName=“ISCS”,nMode=1,nOpt=1，nLevel=level1，重新创建线程。调用getCtrlDevice(RUNNINGINFO)。
20. 复位控制，例如：szSystemName=“ISCS”,nMode=1,nOpt=0。先由ISCS和1，去火灾模式系统表取所属系统(例如：TK)，将参数szSystemName=“ISCS-TK”,nMode=0,nOpt=0，通过接口setSystemReturn(vector<MQTTMODE> info)发送给plc\_iotdevice。如果此时TK系统正在执行ISCS的1模式，需要终止线程nHandle执行，并将list< RUNNINGINFO >里的TK系统的nLevel置为0。
21. 正常模式控制normalModeCtrl(MQTTMODE info) ，例如：szSystemName=“TK”,nMode=1,nOpt=1
    1. 先由系统代码TK去list< RUNNINGINFO >里找到当前模式对应的环控系统szSystem，判断该环控系统的运行模式nLevel，nLevel>0表示该环控系统的运行模式为火灾模式；nLevel=0表示该环控系统的运行模式为正常模式。
    2. 环控系统正在运行火灾模式：忽略，退出函数。
    3. 环控系统正在运行正常模式：将正在执行的模式线程nHandle终止，将RUNNINGINFO的信息换成szSystemName=“TK”,nMode=1,nOpt=1。然后将参数szSystemName=“TK”,nMode=1,nOpt=1通过接口setSystemReturn(vector<MQTTMODE> info)发送给plc\_iotdevice。将RUNNINGINFO的信息换成szSystemName=“TK”,nMode=1,nOpt=1，nLevel=0，重新创建线程。由szSystemName=“TK”,nMode=1从正常模式系统表中获取执行超时阈值nTimeOut。调用getCtrlDevice(RUNNINGINFO)。
22. 在getCtrlDevice(RUNNINGINFO info)里，通过szSystemName和nMode从子模式表中筛选，将获取到的信息都保存到list<MODEINFO>里，然后将list<MODEINFO>和之前获取到的执行超时阈值nTimeOut保存到RUNNINGINFO中，然后传入线程nHandle里。
23. 开始模式执行，模式执行由线程函数modeCtrlThread(void\* argp)执行。初始所有设备的标志位bExecuted都为0，表示还未执行。记录日志(模式执行开始时间)，开始计时t0。
    1. 循环执行所有设备
       1. 判断模式执行是否超时。（当前时间-t0）>执行超时阈值，执行超时，退出循环，退出线程，日志记录。反之则未超时，继续执行。
       2. 标志位bExecuted为1的直接跳过，进入下一个控制。
       3. 有延时执行则记录当前时间nFirstDelayTime，下次再执行时，t3=当前时间- nFirstDelayTime，如果t3<延时时间nDelayTime，则跳过，进入下一个控制；如果t3>延时时间nDelayTime，则继续往下。
       4. 验证期望属性和期望值。使用MODEINFO里的控制设备szDevice和期望属性szExpectProp，通过接口getDeviceInfo（char \* device,char \* property,int& type,strign& value）获取到期望属性para的当前值value。比较期望值nExpectValue 和当前值value是否一致。
       5. 如果期望属性当前值与子模式表里的期望值一致则不做控制，置标志位1，记录日志，进入下一个控制；如果不一致则继续往下。
       6. 执行闭锁逻辑脚本szInterLock并获取结果。这里要将ehmi的脚本部分移植过来。CLuaScript m\_Script ; scriptfile –脚本文件全路径。int nResult=m\_Script.ExecPFile(scriptfile)
       7. 获取结果nResult为1，则表示联锁条件满足，此时通过多功能网关平台下发控制, 下发参数使用子模式表的控制设备、产品、控制属性和控制值{“device”：“TVF1”, “product”：”43cc7”,“property”:“control”,value:1},置标记位为1，记录日志；获取结果nResult为2，则表示为手动模式，此时不下发控制，直接置标记位为1，记录日志；获取结果nResult为0，则表示联锁条件不满足，不下发控制，此时如果是第一次执行则记录当前时间nFirstLockTime，如果不是第一次控制则要判断是否超时(t4=当前时间- nFirstLockTime，t4>闭锁超时阈值nLockTime，则认为已经执行，置标志位为1，记录日志；t4<闭锁超时阈值nLockTime，则继续后续步骤)。进入下一个控制。
       8. 所有设备的标志位bExecuted都为1，退出循环，则为完成模式控制，退出线程。记录日志(模式执行结束时间)。
24. 进程退出时，清空数组list< FIREINFO >、vector<string >。停止并清除所有线程，然后清空数组list< RUNNINGINFO >。