
文件编号		密级	对外	页数	29
项目编号		版次	1.4	受控编号	

文件名称：编程接口库使用说明书

单 位：珞石（北京）科技有限公司

日 期：2021-11-02

目录

目录	2
1 文档说明	4
1.1 文档目的及内容	4
1.2 文档编号及版本	4
1.3 文档使用对象	4
2 使用方法	4
2.1 开发环境搭建	4
2.2 开发配置	4
2.3 调试运行	5
3 接口函数说明	6
3.1 初始化接口	6
3.1.1 API_Init	6
3.1.2 API_MotorOn	6
3.1.3 API_GetRobotStatus	7
3.1.4 API_StartRobotProgram	8
3.1.5 API_StopRobotProgram	8
3.1.6 API_SetAlarmStatus	9
3.1.7 API_ResetProgramPointer	9
3.1.8 API_Release	10
3.2 控制接口	10
3.2.1 API_PPtomain	10
3.2.2 API_StartRun	11
3.2.3 API_Stop0	12
3.2.4 API_Stop1	12
3.2.5 API_EmgStop	13
3.2.6 API_WriteOutBit	13
3.2.7 API_ReadInBit	14

3.2.8 API_UpdateToolMsg	15
3.3 运动接口	16
3.3.1 API_HomeMove	16
3.3.2 API_MoveLSinglePosition	16
3.3.3 API_MoveLMutiPosition	18
3.3.4 API_MoveAbsjSinglePosition	19
3.3.5 API_MoveAbsjMutiPosition	20
3.3.6 API_MoveLOffSinglePosition	21
3.3.7 API_MoveLOffMutiPosition	22
3.4 查询接口	24
3.4.1 API_GetRobotPosition	24
3.4.2 rokae_last_msg	25
4 数据结构说明	26
4.1 ROKAEJNTARGETPOS	26
4.2 ROKAESDKGETJNTROBOTPOSITION	26
4.3 ROKAEROBTARGETPOS	26
4.4 ROKAESDKMOTIONPARAM	27
4.5 ROKAESDKGETROBROBOTPOSITION	27
4.6 ROKAEROBTARGETPOSOFF	28
4.7 ROKAESDKGETROBOTPOSITION	28
4.8 ROKAESDKIOVALUEPOD	28
5 错误码说明	29

1 文档说明

1.1 文档目的及内容

编程接口库是珞石机器人提供给客户用于二次开发的软件产品，通过编程接口库，客户可以对机器人进行一系列控制和操作。

该使用说明书主要介绍编程接口库的使用方法，以及各接口函数的功能，并为每个接口函数都提供了示例代码。

1.2 文档编号及版本

文档相关信息见表 1-1。

表 1-1 文档相关信息

文档名称	《编程接口库使用说明书》
文档版本	V1.4
软件版本号	3.6.0

1.3 文档使用对象

- 操作员
- 示教员
- 维护工程师

2 使用方法

2.1 开发环境搭建

编程接口库使用 C++ 编写，Visual Studio 2019 编译，64 位或 32 位，整个接口库包括 3 个文件，分别为：

- RobotAPI.dll
- RobotAPI.h
- RobotAPI.lib

请安装 Visual Studio 2019，并同时选择安装所有 C++ 类库。

2.2 开发配置

创建 C++ 工程后，将 RobotAPI.h 复制到某个任意目录下，建议复制到工程目录下，在工程属性-配置属性-C/C++-常规的“附加包含目录”中添加 RobotAPI.h 所在目录，如下图 2-1：

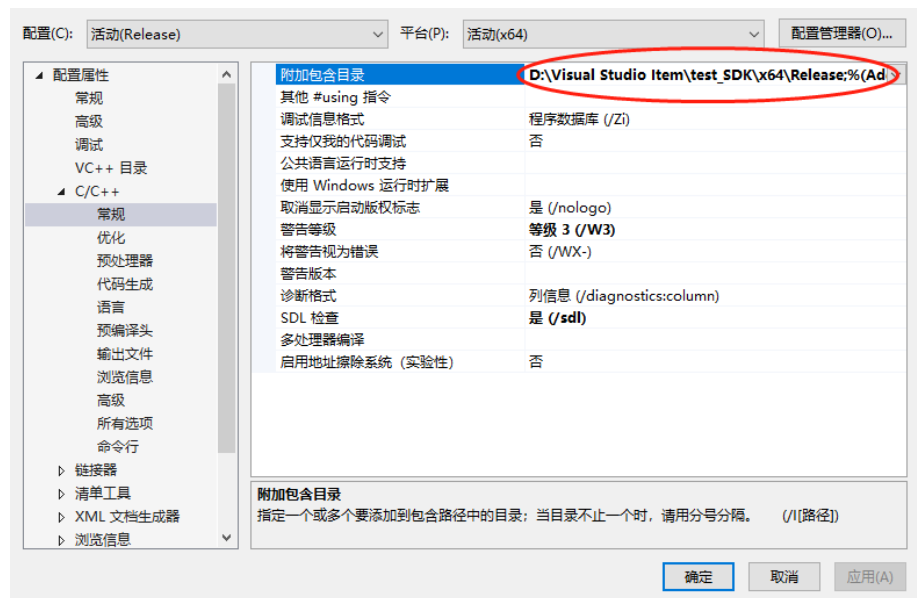


图 2-1 “附加包含目录”中添加 RobotAPI.h 所在目录

在解决方案的资源文件中添加 RobotAPI.lib 文件，如下图 2-2、2-3：

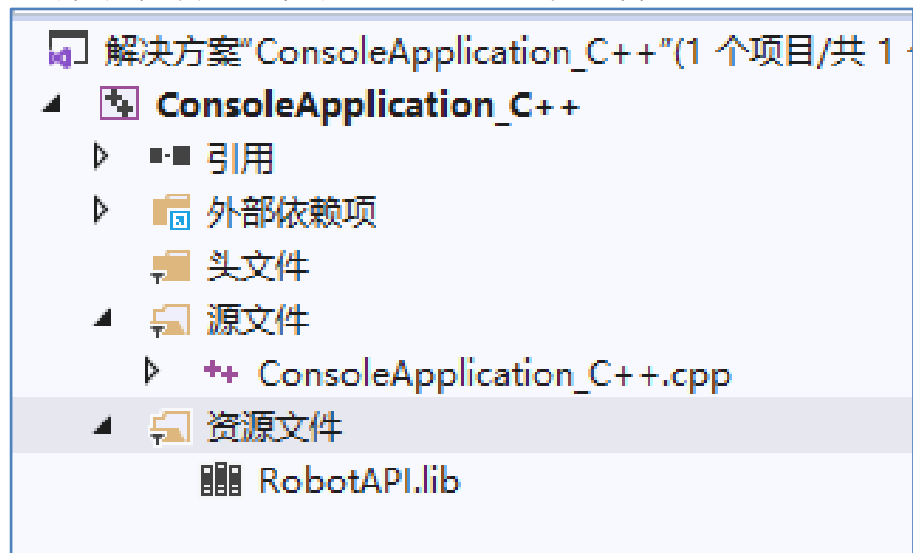


图 2-2 “资源文件”选项

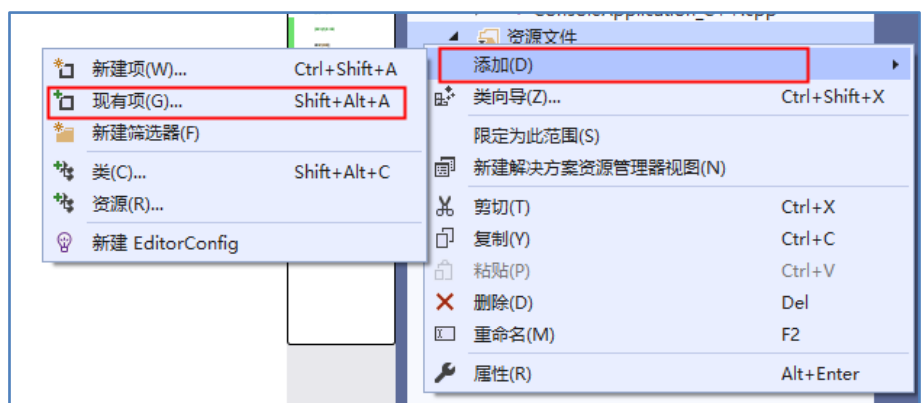


图 2-3 “资源文件”添加 RobotAPI.lib

至此，工程中就可以正常调用 RobotAPI.h 中提供的各接口函数，并能正常编译。

2.3 调试运行

调试或运行前，请将 RobotAPI.dll 复制到运行目录。PC 需要将 IP 地址设置为“192.168.0.xx”网段，且用网线连接到机器人的调试网口，若未在示教器上修改调试网口的 IP 地址，则调试网口的默认地址为“192.168.0.160”。

机器人端需要将 RL 工程文件“__Rokae_SDK_Task__”通过 U 盘拷贝到示教器中，然后加载“__Rokae_SDK_Task__”工程，示教器上的模式选择开关切换到自动模式。

3 接口函数说明

3.1 初始化接口

3.1.1 API_Init

函数声明

```
Int API_Init(const char *ip, unsigned int port)
```

功能

初始化，建立连接等初始化操作

参数

- ip
 - ip 地址，点分十进制形式，例如：“192.168.3.100”
- port
 - 端口号 0 - 65535

返回

0：执行成功

1：执行失败

2：执行超时

65535：无效值

使用方法

```
int ret = 0;
//初始化
ret = API_Init("192.168.0.160", 50502);
if (ret == 0) {    //初始化成功
}
}else {           //初始化失败
}
}
```

◆ 目前端口必须设置为“50502”。

3.1.2 API_MotorOn

函数声明

Int API_MotorOn()

功能

伺服接通，电机上电

返回

0: 执行成功

1: 执行失败

2: 执行超时

65535: 无效值

使用方法

```
int ret = 0;

//上电

ret = API_MotorOn();

if (ret == 0) {    //上电成功

}else {          //上电失败

}
```

注意事项

调用 API_MotorOn() 函数，确认收到函数返回值为 0 后，建议等待至少 100ms，再进行下一个函数的调用操作。

3.1.3 API_GetRobotStatus

函数声明

Int API_GetRobotStatus(char* msg, unsigned int *len)

功能

查询机械手状态。若异常，返回异常报错信息

参数

- msg
 - 状态信息码对应的状态
 - 0: 初始化状态
 - 1: 电机停止状态
 - 2: 电机上电状态
 - 3: 常规停止状态

4: 紧急停止状态

5: 故障状态

■ len

➤ 错误信息长度

返回

0: 执行成功

1: 执行失败

2: 执行超时

65535: 无效值

使用方法

```
int ret;
char msg;
unsigned int len;
//查询机械手状态
ret = API_GetRobotStatus(&msg, &len);
if (ret == 0) {           //查询成功
    std::cout << "msg = " << msg << std::endl;
} else {                 //查询失败
}
```

注意事项

调用 API_GetRobotStatus() 函数前，需要先清除示教器界面报警弹框。

3.1.4 API_StartRobotProgram

函数声明

Int API_StartRobotProgram()

功能

启动机械手程序

返回

int: 错误码-- 0, 正常; 返回值<0, 异常, 具体待定义

使用方法

暂不支持

3.1.5 API_StopRobotProgram

函数声明

Int API_StopRobotProgram()

功能

停止机械手程序

返回

int : 错误码-- 0, 正常; 返回值<0, 异常, 具体待定义

使用方法

暂不支持

3.1.6 API_SetAlarmStatus

函数声明

Int API_SetAlarmStatus(int value)

功能

设置报警和清除报警, 暂时仅支持清除报警功能

参数

- value
 - 0, 清除报警; 1, 报警

返回

0: 执行成功

1: 执行失败

2: 执行超时

65535: 无效值

使用方法

```
int ret = 0;
//清除报警
ret = API_SetAlarmStatus(0);
if (ret == 0) {           //清除成功
}
}else {                  //清除失败
}
}
```

3.1.7 API_ResetProgramPointer

函数声明

Int API_ResetProgramPointer()

功能

复位机械手程序指针

返回

int: 错误码-- 0, 正常; 返回值<0, 异常, 具体待定义

使用方法

暂不支持

3.1.8 API_Release

函数声明

Int API_Release()

功能

释放函数, 执行下电, 断开连接操作

返回

0: 执行成功

1: 执行失败

2: 执行超时

65535: 无效值

使用方法

```
int ret = 0;

//释放函数

ret = API_Release();

if (ret == 0) {    //释放成功

}else {          //释放失败

}
```

3.2 控制接口

3.2.1 API_PPtomain

函数声明

Int API_PPtomain()

功能

程序指针指到 main 函数第一行。

返回

0: 执行成功

1: 执行失败

2: 执行超时

65535: 无效值

使用方法

```
int ret = 0;
ret = API_PPtomain ();

if (ret == 0) {    //程序指针指到 main 成功

}
else {           //程序指针指到 main 失败

}
}
```

注意事项

调用 API_PPtomain()接口须满足条件: 控制器处于自动模式且电机为上电状态和程序处于暂停状态。

3.2.2 API_StartRun

函数声明

Int API_StartRun()

功能

开始执行程序。

返回

0: 执行成功

1: 执行失败

2: 执行超时

65535: 无效值

使用方法

```
int ret = 0;
ret = API_StartRun();

if (ret == 0) {    //程序启动成功

}
else {           //程序启动失败

}
}
```

}

3.2.3 API_Stop0

函数声明

Int API_Stop0()

功能

暂停运行程序，机器人未下电的情况下，可调用 API_StartRun()、按程序启动按钮或

系统 IO 输入启动程序，继续执行未运行结束的点。

返回

0: 执行成功

1: 执行失败

2: 执行超时

65535: 无效值

使用方法

```
int ret = 0;
ret = API_Stop0();
if (ret == 0) {    //程序暂停成功
}
else {            //程序暂停失败
}
}
```

3.2.4 API_Stop1

函数声明

Int API_Stop1()

功能

停止运行程序，未运行结束的点不能继续执行。

返回

0: 执行成功

1: 执行失败

2: 执行超时

65535: 无效值

使用方法

```
int ret = 0;
```

```
ret = API_Stop1();

if (ret == 0) {    //程序停止成功

}else {          //程序停止失败

}

}
```

3.2.5 API_EmgStop

函数声明

```
Int API_EmgStop()
```

功能

急停止机器人并下电。

返回

0: 执行成功

1: 执行失败

2: 执行超时

65535: 无效值

使用方法

```
int ret = 0;
ret = API_EmgStop();

if (ret == 0) {    //急停成功

}else {          //急停失败

}

}
```

注意事项

调用此接口后，机器人会立即下电，并抱闸，若机器人正在高速运行，调用此接口会对抱闸等机械部件带来一定的损耗。此接口仅在紧急停止时使用，不可用于常规停止。

3.2.6 API_WriteOutBit

函数声明

```
Int API_WriteOutBit(RokaeSDKIOValuePOD msg)
```

功能

设置 IO 信号

参数

■ msg

- 设置 DO 信号的参数

返回

0: 执行成功

1: 执行失败

2: 执行超时

65535: 无效值

使用方法

```
int ret = 0;
RokaeSDKIOValuePOD pod;
char name[80] = "do0";
memcpy(pod.pod_name, name, strlen(name) + 1);
pod.pod_value = 1;    //do0 信号的值设置为 1
ret = API_WriteOutBit(pod);
if (ret == 0) {      //设置 IO 成功

}
else {              //设置 IO 失败

}
}
```

3.2.7 API_ReadInBit

函数声明

```
Int API_ReadInBit(char* di_name, int name_len, RokaeSDKIOValuePOD* msg,
unsigned int* len)
```

功能

读取 IO 信号

参数

- di_name
 - 要读取信号的名字
- name_len
 - 信号名称的长度
- msg
 - 获取信号的信息
- len
 - 获取信息的长度

返回

0: 执行成功
1: 执行失败
2: 执行超时
65535: 无效值

使用方法

```
int ret = 0;
char di_name[80] = "di0";
unsigned int len = 0;
RokaeSDKIOValuePOD pod_msg;
ret = API_ReadInBit(di_name, strlen(di_name), &pod_msg, &len);
if (ret == 0) {    //读取 IO 成功
    std::cout << di_name << " = " << pod_msg.pod_value << std::endl;
} else {          //读取 IO 失败
}
```

3.2.8 API_UpdateToolMsg

函数声明

```
Int API_UpdateToolMsg(char* tool_name , int name_len)
```

功能

切换工具。

参数

- tool_name
 - 要切换的工具的名称
- name_len
 - 要切换工具的名称长度

返回

0: 执行成功
1: 执行失败
2: 执行超时
65535: 无效值

使用方法

```
char tool_name[7] = "tool01";
int ret = 0;
```

```
ret = API_UpdateToolMsg(tool_name, 6);  
if (ret == 0) {    //切换工具成功  
  
}else {           //切换工具失败  
  
}
```

注意事项

切换的工具需为系统已有的工具, 否则会切换失败。切换工具前可提前将工具标定好。

切换工具只对后续发送的运动指令生效, 对机器人正在执行的运动指令不生效。

3.3 运动接口

3.3.1 API_HomeMove

函数声明

```
Int API_HomeMove()
```

功能

机械手返回初始位置, 该初始位置预留在机械手内。

返回

0: 执行成功

1: 执行失败

2: 执行超时

65535: 无效值

使用方法

```
int ret = 0;  
//机械手返回初始位置  
  
ret = API_HomeMove();  
  
if (ret == 0) {           //到达初始位置成功  
  
}else {                  //到达初始位置失败  
  
}
```

注意事项

在调用 API_HomeMove()接口前, 需要开启起始点功能。

3.3.2 API_MoveLSinglePosition

函数声明

Int API_MoveLSinglePosition(int iMoveMode, rokae_robotarget_pos rob_pos)

功能

移动单个 robpos, 对应 MoveL/MoveJ 指令

参数

- iMoveMode
 - 运动模式, 0: MoveJ ; 1: MoveL
- rokae_robotarget_pos
 - 执行 MoveL, MoveJ 时的必须参数

返回

0: 执行成功

1: 执行失败

2: 执行超时

65535: 无效值

使用方法

//设置位置、速度和转弯区参数

```
rokae_robotarget_pos rob_pos;
rob_pos.euler_type = 0;
rob_pos.rob_pos.cogx = 365;
rob_pos.rob_pos.cogy = 0;
rob_pos.rob_pos.cogz = 620;
rob_pos.rob_pos.q1 = 0;
rob_pos.rob_pos.q2 = 0;
rob_pos.rob_pos.q3 = 1;
rob_pos.rob_pos.q4 = 0;
rob_pos.rob_pos.a = -150;
rob_pos.rob_pos.b = 0;
rob_pos.rob_pos.c = 180;
rob_pos.motion_param.per = 100;
rob_pos.motion_param.tcp = 7000;
rob_pos.motion_param.iZone = 200;
```

```
int ret = 0;
```

//以 MoveL 的模式运动到设置的位置

```
ret = API_MoveLSinglePosition(1, rob_pos);
```

```
if (ret == 0) {           //运行成功
```

```
    }else {                //运行失败
    }
}
```

3.3.3 API_MoveLMutiPosition

函数声明

```
Int API_MoveLMutiPosition(int iMoveMode, rokae_robtargt_pos* rob_pos, int num)
```

功能

移动多个 robpos, 对应 MoveL/MoveJ 指令

参数

- iMoveMode
 - 运动模式, 0: MoveJ ; 1: MoveL
- rokae_robtargt_pos
 - 执行 MoveL, MoveJ 时的必须参数
- num
 - 点位个数

返回

0: 执行成功

1: 执行失败

2: 执行超时

65535: 无效值

使用方法

//设置 2 个点的位置、速度和转弯区等参数

```
rokae_robtargt_pos murob_pos[2];
for (int i = 0; i < 2; i++)
{
    murob_pos[i].euler_type = 3;
    murob_pos[i].rob_pos.q1 = 0;
    murob_pos[i].rob_pos.q2 = 0;
    murob_pos[i].rob_pos.q3 = 1;
    murob_pos[i].rob_pos.q4 = 0;
    murob_pos[i].rob_pos.a = 180;
    murob_pos[i].rob_pos.b = 0;
    murob_pos[i].rob_pos.c = 180;
    murob_pos[i].motion_param.per = 100;
}
```

```

        murob_pos[i].motion_param.tcp = 800;
        murob_pos[i].motion_param.iZone = 200;
    }

    murob_pos[0].rob_pos.cogx = 365;
    murob_pos[0].rob_pos.cogy = -100;
    murob_pos[0].rob_pos.cogz = 672;

    murob_pos[1].rob_pos.cogx = 365;
    murob_pos[1].rob_pos.cogy = 100;
    murob_pos[1].rob_pos.cogz = 672;

    int ret = 0;
    //以 MoveJ 的模式运动到设置的 2 个点的位置，按数组升序运行点
    ret = API_MoveLMutiPosition(0, murob_pos, 2);
    if (ret == 0) {           //运行成功
    }else {                   //运行失败
    }
}

```

3.3.4 API_MoveAbsjSinglePosition

函数声明

```
Int API_MoveAbsjSinglePosition(rokae_jnttarget_pos jnt_pos)
```

功能

移动单个 jntpos，对应 moveabsj 指令

参数

- rokae_jnttarget_pos
 - 执行 MoveAbsj 时的必须参数

返回

- 0: 执行成功
- 1: 执行失败
- 2: 执行超时
- 65535: 无效值

使用方法

```

//设置关节角度、速度和转弯区参数
rokae_jnttarget_pos jnt_pos;
jnt_pos.j1 = 0;

```

```
jnt_pos.j2 = 0;
jnt_pos.j3 = 0;
jnt_pos.j4 = 0;
jnt_pos.j5 = 90;
jnt_pos.j6 = 0;
jnt_pos.per = 100;
jnt_pos.iZone = 50;
jnt_pos.tcp = 100;

int ret = 0;
//运动到设置的关节角度位置
ret = API_MoveAbsjSinglePosition( jnt_pos);
if (ret == 0) {           //运行成功
}
else {                   //运行失败
}
}
```

3.3.5 API_MoveAbsjMutiPosition

函数声明

```
Int API_MoveAbsjMutiPosition(rokae_jnttarget_pos *jnt_pos, int num)
```

功能

移动多个 jntpos，对应 moveabsj 指令

参数

- rokae_jnttarget_pos
 - 执行 MoveAbsj 时的必须参数
- num
 - 点位个数

返回

0: 执行成功
 1: 执行失败
 2: 执行超时
 65535: 无效值

使用方法

```
//设置 2 个点的关节角度、速度和转弯区等参数
rokae_jnttarget_pos mujnt_pos[2];
```

```

for (int i = 0; i < 2; i++)
{
    mujnt_pos[i].j1 = 0 + i * -20;
    mujnt_pos[i].j2 = 0;
    mujnt_pos[i].j3 = 0;
    mujnt_pos[i].j4 = 0;
    mujnt_pos[i].j5 = 90;
    mujnt_pos[i].j6 = 0;
    mujnt_pos[i].per = 100;
    mujnt_pos[i].iZone = 50;
    mujnt_pos[i].tcp = 100;
}

int ret = 0;

//运动到设置的 2 个关节角度的位置，按数组升序运行点

ret = API_MoveAbsjMutiPosition(mujnt_pos, 2);

if (ret == 0) {           //运行成功

}
else {                   //运行失败

}

```

3.3.6 API_MoveLOffSinglePosition

函数声明

Int API_MoveLOffSinglePosition(int iMoveMode, rokae_robtargt_pos_off rob_pos)

功能

在坐标点的基础上进行偏移

参数

- iMoveMode
 - 运动模式，0: MoveJ； 1: MoveL
- rob_pos
 - 执行 MoveL, MoveJ 时的必须参数

返回

- 0: 执行成功
- 1: 执行失败
- 2: 执行超时
- 65535: 无效值

使用方法

```
//设置位置、速度和转弯区等参数
rokae_robtargt_pos_off rob_pos;
rob_pos.euler_type = 0;
rob_pos.rob_pos.cogx = 365;
rob_pos.rob_pos.cogy = 0;
rob_pos.rob_pos.cogz = 620;
rob_pos.rob_pos.q1 = 0;
rob_pos.rob_pos.q2 = 0;
rob_pos.rob_pos.q3 = 1;
rob_pos.rob_pos.q4 = 0;
rob_pos.rob_pos.a = -150;
rob_pos.rob_pos.b = 0;
rob_pos.rob_pos.c = 180;
rob_pos.motion_param.per = 100;
rob_pos.motion_param.tcp = 7000;
rob_pos.motion_param.iZone = 200;
rob_pos.x_off = 100;
rob_pos.y_off = 0;
rob_pos.z_off = 0;

int ret = 0;

//以 MoveL 的模式运动到设置的位置
ret = API_MoveLSinglePosition(1, rob_pos);
if (ret == 0) {           //运行成功
}
else {                   //运行失败
}
}
```

3.3.7 API_MoveLOffMutiPosition

函数声明

```
Int API_MoveLOffMutiPosition(int iMoveMode,rokae_robtargt_pos_off* rob_pos, int num)
```

功能

在坐标点的基础上进行偏移

参数

- iMoveMode
 - 运动模式, 0: MoveJ ; 1: MoveL
- rob_pos

➤ 执行 MoveL, MoveJ 时的必须参数

■ num

➤ 点位个数

返回

0: 执行成功

1: 执行失败

2: 执行超时

65535: 无效值

使用方法

//设置位置、速度和转弯区等参数

```
rokae_robtargt_pos_off rob_pos[2];
rob_pos[0].base_rob_pos.euler_type = 2;
rob_pos[0].base_rob_pos.rob_pos.cogx = 365;
rob_pos[0].base_rob_pos.rob_pos.cogy = 0;
rob_pos[0].base_rob_pos.rob_pos.cogz = 638;
rob_pos[0].base_rob_pos.rob_pos.q1 = 0;
rob_pos[0].base_rob_pos.rob_pos.q2 = 0;
rob_pos[0].base_rob_pos.rob_pos.q3 = 1;
rob_pos[0].base_rob_pos.rob_pos.q4 = 0;
rob_pos[0].base_rob_pos.rob_pos.a = 180;
rob_pos[0].base_rob_pos.rob_pos.b = 0;
rob_pos[0].base_rob_pos.rob_pos.c = 180;
rob_pos[0].base_rob_pos.motion_param.tcp = 500;
rob_pos[0].base_rob_pos.motion_param.per = 100;
rob_pos[0].base_rob_pos.motion_param.iZone = 200;
rob_pos[0].x_off = 100;
rob_pos[0].y_off = 0;
rob_pos[0].z_off = 0;
```

```
rob_pos[1].base_rob_pos.euler_type = 2;
rob_pos[1].base_rob_pos.rob_pos.cogx = 365;
rob_pos[1].base_rob_pos.rob_pos.cogy = 100;
rob_pos[1].base_rob_pos.rob_pos.cogz = 638;
rob_pos[1].base_rob_pos.rob_pos.q1 = 0;
rob_pos[1].base_rob_pos.rob_pos.q2 = 0;
rob_pos[1].base_rob_pos.rob_pos.q3 = 1;
rob_pos[1].base_rob_pos.rob_pos.q4 = 0;
rob_pos[1].base_rob_pos.rob_pos.a = 180;
rob_pos[1].base_rob_pos.rob_pos.b = 0;
rob_pos[1].base_rob_pos.rob_pos.c = 180;
rob_pos[1].base_rob_pos.motion_param.tcp = 500;
rob_pos[1].base_rob_pos.motion_param.per = 100;
rob_pos[1].base_rob_pos.motion_param.iZone = 200;
rob_pos[1].x_off = 100;
rob_pos[1].y_off = 0;
rob_pos[1].z_off = 0;
```

```
int ret = 0;
```

//以 MoveL 的模式运动到设置的位置, 按数组升序运行点

```
ret = API_MoveLOffMutiPosition(1, rob_pos, 2);
```

```
if (ret == 0) {           //运行成功
```

```
}else {                 //运行失败
```

```
}
```

3.4 查询接口

3.4.1 API_GetRobotPosition

函数声明

```
Int API_GetRobotPosition (rokae_sdk_get_robot_position* msg, unsigned int* len)
```

功能

获取机器人位姿信息

参数

- msg
 - 机器人位姿信息
- len
 - 信息长度

返回

0: 执行成功

1: 执行失败

2: 执行超时

65535: 无效值

使用方法

```
RokaesDKGetRobotPosition position;
unsigned int len = 0;
int ret = 0;
ret = API_GetRobotPosition(&position, &len);

if (ret == 0) {    //获取机器人位姿信息成功

    std::cout << "len = " << len << std::endl;
    std::cout << "J1 = " << position.jnt_pos.j1 << std::endl;
    std::cout << "J2 = " << position.jnt_pos.j2 << std::endl;
    std::cout << "J3 = " << position.jnt_pos.j3 << std::endl;
    std::cout << "J4 = " << position.jnt_pos.j4 << std::endl;
    std::cout << "J5 = " << position.jnt_pos.j5 << std::endl;
    std::cout << "J6 = " << position.jnt_pos.j6 << std::endl;
    std::cout << "X = " << position.rob_pos.cogx << std::endl;
    std::cout << "Y = " << position.rob_pos.cogy << std::endl;
    std::cout << "Z = " << position.rob_pos.cogz << std::endl;
```



```
std::cout << "q1 = " << position.rob_pos.q1 << std::endl;
std::cout << "q2 = " << position.rob_pos.q2 << std::endl;
std::cout << "q3 = " << position.rob_pos.q3 << std::endl;
std::cout << "q4 = " << position.rob_pos.q4 << std::endl;
std::cout << "A = " << position.rob_pos.a << std::endl;
std::cout << "B = " << position.rob_pos.b << std::endl;
std::cout << "C = " << position.rob_pos.c << std::endl;

}else {          //获取机器人位姿信息失败

}
```

注意事项

获取的位姿信息与状态监控的位姿一致

3.4.2 rokae_last_msg

函数声明

```
Int rokae_last_msg(char *str, unsigned int *len)
```

功能

读取信息通用方法，持续保存全局操作的最后一次错误信息

参数

- str
 - 输出信息字符串
- len
 - 输出信息字符串长度

返回

0: 执行成功

1: 执行失败

2: 执行超时

65535: 无效值

使用方法

```
char msg[256];

//读取错误信息

rokae_last_msg(msg, nullptr);

std::cout << "msg = " << msg << std::endl;
```

4 数据结构说明

4.1 RokaJntTargetPos

```
//MoveAbsj 执行时需要的数据
RokaJntTargetPos {
    rokae_sdk_motion_param  motion_param;      //运动参数
    rokae_sdk_get_jnt_robot_position jnt_pos;    //关节数据信息
} rokae_jnttarget_pos; 相关参数说明详见表 4-1。
```

表 4-1 RokaJntTargetPos 相关参数说明

名称	说明
motion_param	运动参数
jnt_pos	关节位置信息

4.2 RokaSDKGetJntRobotPosition

```
//关节位置信息
RokaSDKGetJntRobotPosition {
    double j1;      //机器人 1 轴关节角度
    double j2;      //机器人 2 轴关节角度
    double j3;      //机器人 3 轴关节角度
    double j4;      //机器人 4 轴关节角度
    double j5;      //机器人 5 轴关节角度
    double j6;      //机器人 6 轴关节角度
} rokae_sdk_get_jnt_robot_position; 相关参数说明详见表 4-2。
```

表 4-2 RokaSDKGetJntRobotPosition 相关参数说明

名称	说明
j1-j6	机器人 1 轴-6 轴关节角度

4.3 RokaRobtargetPos

```
//MoveL, MoveJ 执行时需要的数据
RokaRobtargetPos {
    int euler_type;
    rokae_sdk_motion_param  motion_param;
    rokae_sdk_get_rob_robot_position rob_pos;
} rokae_robtarget_pos; 相关参数说明详见表 4-3。
```

表 4-3 RokaRobtargetPos 相关参数说明

名称	说明
euler_type	读取欧拉 jog 顺规类型 :0 为使用欧拉角 XYZ 顺规; 1 为使用欧拉角 ZYX 顺规; 其他为使用四元数规则。(通常为 xyz 规则或四元数规则)
motion_param	运动参数
rob_pos	笛卡尔空间数据信息

4.4 RokaesDKMoTionParam

//运动参数

```
RokaesDKMoTionParam {
    double per;
    double tcp;
    double iZone;
}rokae_sdk_motion_param; 相关参数说明详见表 4-4。
```

表 4-4 RokaesDKMoTionParam 相关参数说明

名称	说明
per	关节速度百分比, 取值范围 (1~100)
tcp	TCP 线速度, 取值范围 (1~7000)
iZone	笛卡尔空间转弯区大小 , 取值范围 (0~200)

4.5 RokaesDKGetRobRobotPosition

//笛卡尔空间数据信息

```
RokaesDKGetRobRobotPosition {
    double cogx;
    double cogy;
    double cogz;
    double q1;
    double q2;
    double q3;
    double q4;
    double a;
    double b;
    double c;
}rokae_sdk_get_rob_robot_position; 相关参数说明详见表 4-5。
```

表 4-5 RokaesDKGetRobRobotPosition 相关参数说明

名称	说明
cogx	X 坐标
cogy	Y 坐标

cogz	z 坐标
q1	四元数: q1
q2	四元数: q2
q3	四元数: q3
q4	四元数: q4
a	欧拉角 A
b	欧拉角 B
c	欧拉角 C

4.6 RokaerobtargetPosOff

// API_MoveLOffSinglePosition()和 API_MoveLOffMutiPosition()函数使用的数据类型

```
RokaerobtargetPosOff {
    rokae_robtarg_pos base_rob_pos;
    double x_off;
    double y_off;
    double z_off;
} rokae_robtarg_pos_off; 相关参数说明详见表 4-6。
```

表 4-6 RokaerobtargetPosOff 相关参数说明

名称	说明
base_rob_pos	基于坐标点信息, 数据结构为 RokaerobtargetPos
x_off	基于 base_rob_pos 在 x 方向的偏移量
y_off	基于 base_rob_pos 在 y 方向的偏移量
z_off	基于 base_rob_pos 在 z 方向的偏移量

4.7 RokaesdkGetRobotPosition

// API_GetRobotPosition ()函数使用的数据类型

```
RokaesdkGetRobotPosition {
    rokae_sdk_get_jnt_robot_position jnt_pos;
    rokae_sdk_get_rob_robot_position rob_pos;
} rokae_sdk_get_robot_position; 相关参数说明详见表 4-7。
```

表 4-7 RokaesdkGetRobotPosition 相关参数说明

名称	说明
jnt_pos	关节位置信息
rob_pos	笛卡尔空间数据信息

4.8 RokaesdkIOValuePOD

// IO 信息数据

```
RokaeSDKIOValuePOD {
    char pod_name;
    int pod_value;
```

};RokaeIntTargetPos 相关参数说明详见表 4-8。

表 4-8 RokaeSDKIOValuePOD 相关参数说明

名称	说明
pod_name	信号对应名字
pod_value	信号对应电平,0: 低电平; 1: 高电平

5 错误码说明

表 5-1 错误码说明

错误码	枚举值	说明
0x0000	RR_SUCCESS	执行成功
0x0001	RR_FAILED	执行失败
0x0002	RR_TIMEOUT	执行超时
0xffff	RR_INVALID	无效值