AUBO 机器人 Windows_Csharp_sdk 学习资料

（仅供参考）

2019-3-18

遨博(江苏)机器人有限公司
江苏省常州市中科创业中心 B 座 301 室

目录

一、 获取 Windows Csharp SDK 包

SDK 版本为 V1.2.2。
http://download.aubo-robotics.cn:28080/aubo/download/dev/sdk/v1.2.2

Directory Listing For /dev/sdk/v1.2.2/ - Up To /dev/sdk

Filename	Size	Last Modified
aubo-python-release-1.2.2.zip	34186.4 kb	Thu, 13 Sep 2018 09:56:46 GMT
auboi5-sdk-for-windows-x86-csharp-1.2.2.rar	855.8 kb	Thu, 13 Sep 2018 09:56:46 GMT
auboi5-sdk-for-windows-x86-x64-v1.2.2.rar	22651.8 kb	Thu, 13 Sep 2018 09:56:46 GMT
linux-cplusplus/		Mon, 10 Dec 2018 02:07:48 GMT

http://www.aubo-robotics.cn

二、 编程环境

VS 2015

三、 打开 SDK 工程

- 解压缩 SDK 包
- 打开 Vs 2015
- Open Project，打开 auboi5-sdk-for-windows-x86-csharp.sln，打开

四、 Windows Csharp SDK 文件构成

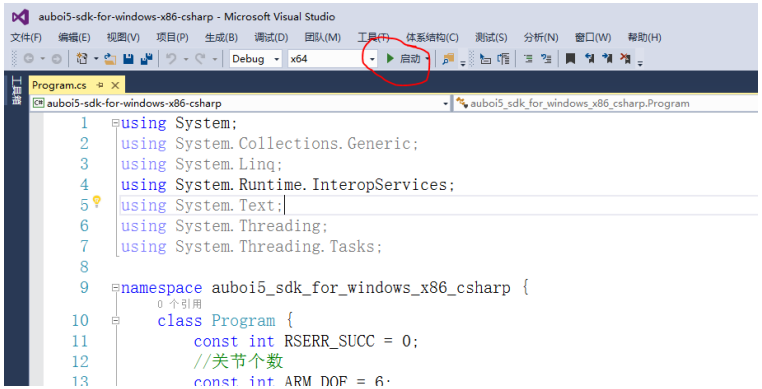
五、 运行 SDK 例子

运行 sdk 例子

1. 打开 Program.cs，修改 IP（机器人 IP）。

```
14          //机械臂IP地址
15          const string robotIP = "192.168.184.129";
16          //机械臂端口号
17          const int serverPort = 8899;
```

2. 点击绿色启动按钮，运行 SDK 工程。



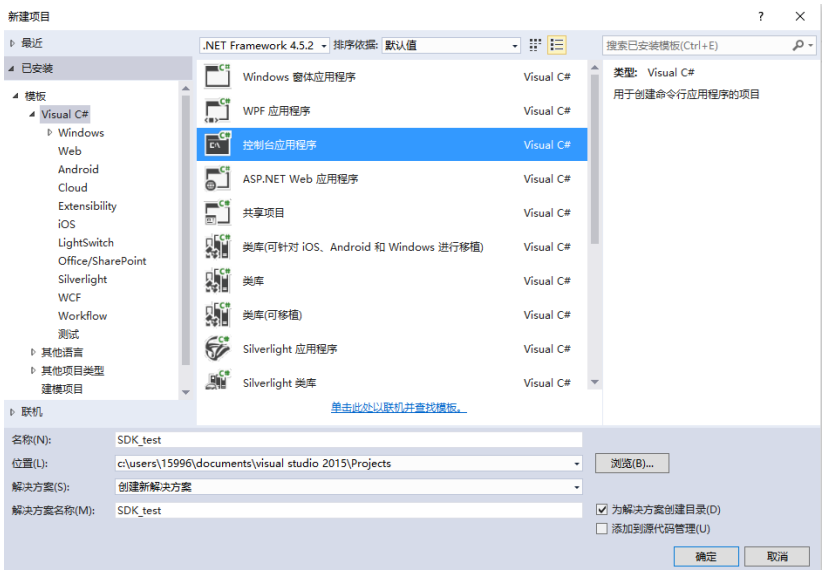
3. 终端打印出信息，机械臂动作。

六、 构建自己的开发程序（基于 VS 2015）

仅供参考。

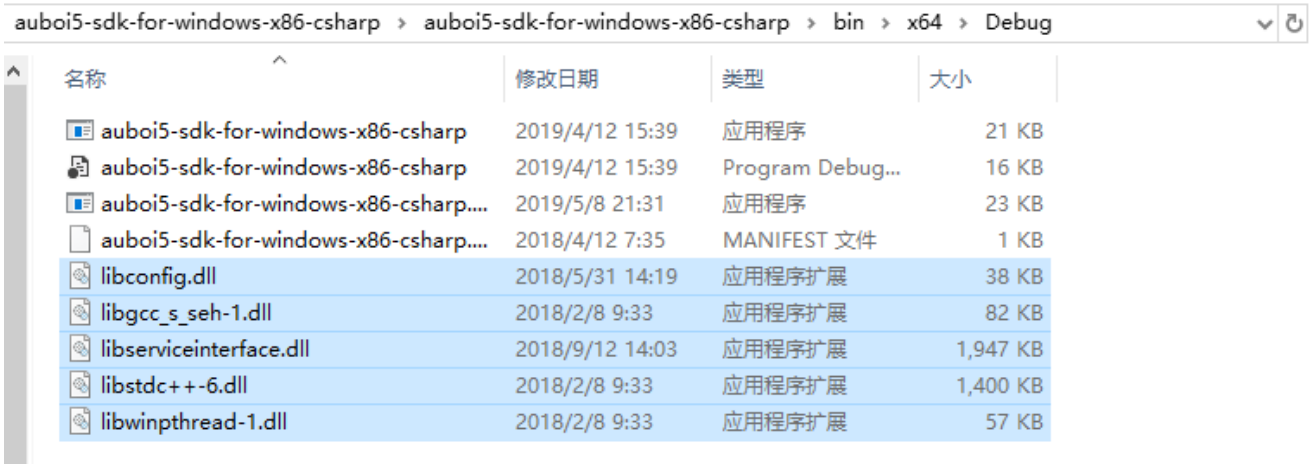
1. 创建新项目

1. 【文件】->【新建】->【项目】->【c#控制台应用程序】->【确定】。本例程的工程名为“SDK_test”。



2. 导入依赖库

2. "\auboi5-sdk-for-windows-x86-csharp\auboi5-sdk-for-windows-x86-csharp\bin\x64\Debug"中的库文件拷贝到 SDK_test 同一路径下。



3. 添加程序内容

3. 向 Program.cs 中添加 `using System.Runtime.InteropServices;`
并将 auboi5-sdk-for-windows-x86-csharp 中程序开头的字段，枚举，结构等复制到程序的开头

```
const int RSERR_SUCC = 0;
//关节个数
const int ARM_DOF = 6;
//机械臂IP地址
const string robotIP = "192.168.184.129";
//机械臂端口号
const int serverPort = 8899;
//M_PI
const double M_PI = 3.14159265358979323846;

//接口板用户DI地址
const int ROBOT_IO_F1 = 30;
//初始化机械臂控制库
[DllImport("libserviceinterface.dll", EntryPoint = "rs_initialize", CharSet = CharSet.Auto, CallingConvention = Calling
0 个引用
public static extern int rs_initialize();

//反初始化机械臂控制库
[DllImport("libserviceinterface.dll", EntryPoint = "rs_uninitialize", CharSet = CharSet.Auto, CallingConvention = Calli
0 个引用
public static extern int rs_uninitialize();

//创建机械臂控制上下文句柄
[DllImport("libserviceinterface.dll", EntryPoint = "rs_create_context", CharSet = CharSet.Auto, CallingConvention = Cal
0 个引用
```

4. 编写 Main 函数中内容如下：

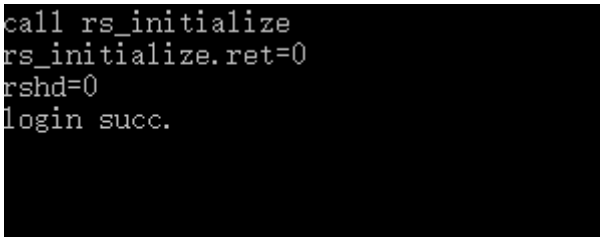
```
static void Main(string[] args)
{
    int result = 0xffff;
    UInt16 rshd = 0xffff;
    Console.Out.WriteLine("call rs_initialize");
    result = rs_initialize();//初始化机械臂控制库
    Console.Out.WriteLine("rs_initialize.ret={0}", result);
    if (RSERR_SUCC == result)
    {
        if (rs_create_context(ref rshd) == RSERR_SUCC)//创建机械臂控制上下文句柄
        {
            Console.Out.WriteLine("rshd={0}", rshd);
            if (rs_login(rshd, robotIP, serverPort) == RSERR_SUCC)//链接机械臂服务器
            {
                Console.Out.WriteLine("login succ.");
                rs_logout(rshd);//断开机械臂服务器链接
            }
            else
                Console.Error.WriteLine("login failed!");
            rs_destory_context(rshd);//注销机械臂控制上下文句柄
        }
        else
        {
            Console.Error.WriteLine("rs_create_context failed!");
        }
        rs_uninitialize();//反初始化机械臂控制库
    }
    else
    {
        Console.Error.WriteLine("rs_initialize failed!");
    }
}
```

4． 编译、运行

5． 按下【F6】生成解决方案

6． 点击绿色启动按钮，运行程序

7． 弹出控制台



七、 Windows Csharp 接口函数示例

机械臂登录与断开

1. 登录 rs_login

int rs_login (UInt16 rshd,string addr, int port);	
函数功能	该函数用于与机械臂服务器建立网络连接，该函数调用成功，是使用其他接口的前提。
参数描述	1、 rshd 上下文控制句柄 2、 addr 机械臂服务器 IP 地址，即控制器的 IP; 3、 port 默认为 8899
返回值	调用成功返回 RSERR_SUCC;错误返回错误号
示例	示例讲解：登录到机器人，成功则打印“login succ”；否则打印“login failed”。

```
int result = 0xffff;
UInt16 rshd = 0xffff;
int ret;
result = rs_initialize();
rs_create_context(ref rshd);
ret = rs_login(rshd, robotIP, serverPort);
if (ret== RSERR_SUCC)
{
    Console.Out.WriteLine("login succ");
}
else
{
    Console.Error.WriteLine("login failed");
}
```

输出	login succ

2. 退出登录 rs_logout

```
int rs_logout (UInt16 rshd);
```

函数功能	退出登录
参数描述	1、 rshd 上下文控制句柄
返回值	调用成功返回 RSERR_SUCC;错误返回错误号
示例	示例讲解：退出登录机器人，成功则打印“logout succ”。

	ret = rs_logout(rshd);
	if (ret== RSERR_SUCC)
	{
	Console.Out.WriteLine("logout succ");
	}

输出	logout succ

3. 获取当前连接状态 rs_get_login_status

int rs_get_login_status(UInt16 rshd, ref bool status);
函数功能	该函数用于查看与机械臂服务器的连接状态
参数描述	1、rshd 上下文控制句柄 2、status 是一个传出参数，网络处于连通状态返回 true;否则返回 false.
返回值	调用成功返回 RSERR_SUCC;错误返回错误号
示例	示例讲解：查看当前登录状态。

	bool status=false;
	ret=rs_get_login_status(rshd,ref status);
	if (ret == RSERR_SUCC)
	{
	Console.Out.WriteLine(status);
	}

输出	True

4. **握手

机械臂上电与断电

5. 机械臂上电 rs_robot_startup

```
int rs_robot_startup (UInt16 rshd, ref ToolDynamicsParam tool, byte colli_class, bool read_pos, bool static_colli_detect, int board_maxacc, ref int state);
```

函数功能	该函数用于启动机械臂，包括上电，松刹车，设置碰撞等级，设置动力学参数等，该函数完成需要的时间比较长，state 表示机械臂启动结果。
参数描述	1、rshd 上下文控制句柄 2、tool：动力学参数，如果末端夹持工具，此参数应该根据具体的来设定；如果末端未夹持工具，将此参数的各项设置为 0。 3、colli_class：碰撞等级 4、read_pose：是否允许读取位置，默认是 true 5、static_colli_detect:默认为 true 6、board_maxacc： 7、state：传出参数，初始化结果，具体参考 ROBOT_SERVICE_STATE 类型。机械臂启动结果只有 result==ROBOT_SERVICE_WORKING 表示机械臂启动成功，否则表示启动失败。

	`enum ROBOT_SERVICE_STATE{`
	`  ROBOT_SERVICE_READY=0,`
	`  ROBOT_SERVICE_STARTING,`
	`  ROBOT_SERVICE_WORKING,`
	`  ROBOT_SERVICE_CLOSING,`
	`  ROBOT_SERVICE_CLOSED,`
	`  ROBOT_SETVICE_FAULT_POWER,`
	`  ROBOT_SETVICE_FAULT_BRAKE,`
	`  ROBOT_SETVICE_FAULT_NO_ROBOT`
	`};`

返回值	调用成功返回 RSERR_SUCC;错误返回错误号
示例	示例讲解：机械臂上电，工具动力学参数为 0，碰撞等级 6。打印启动结果 state。

	`//工具参数`
	`ToolDynamicsParam tool=new ToolDynamicsParam();`
	`tool.payload = 0;`
	`tool.positionX = 0;`

	tool.positionX = 0;
	tool.positionZ = 0;
	tool.toolInertia.xx = 0;
	tool.toolInertia.xy = 0;
	tool.toolInertia.xz = 0;
	tool.toolInertia.yy = 0;
	tool.toolInertia.yz = 0;
	tool.toolInertia.zz = 0;
	//机械臂上电
	int state=0;
	ret=rs_robot_startup(rshd, ref tool, 6, true, true, 1000, ref state);
	if (state==2)
	{
	Console.WriteLine("robot_startup succ " + "state:" + state.ToString());
	}

输出	robot_startup succ state:2

6. 机械臂断电 rs_robot_shutdown

int rs_robot_shutdown (UInt16 rshd);	
函数功能	机械臂断电
参数描述	1、rshd 上下文控制句柄
返回值	调用成功返回 RSERR_SUCC;错误返回错误号

示例	示例讲解：机械臂断电。
	/** 机械臂 Shutdown **/
	ret=rs_robot_shutdown(rshd);
	if (ret== RSERR_SUCC)
	{
	Console.Out.WriteLine("shutdown succ");
	}

输出	shutdown succ

信息推送

7. **使能实时关节状态推送

8. **实时关节信息推送 robotServiceRegisterRealTimeJointStatusCallback

```
int rs_setcallback_realtime_joint_status(UInt16 rshd, [MarshalAs(UnmanagedType.FunctionPtr)] RealTimeJointStatusCallback
RealTimejoint_statusCallback, IntPtr arg);
```

函数功能	该函数用于注册获取“实时关节信息的回调函数”到系统，成功注册后,服务器会通过回调函数实时推送当前的关节状态信息。频率 30ms。
参数描述	1、rshd 上下文控制句柄 2、RealTimejoint_statusCallback 为获取实时路点信息的回调函数指针 3、arg 这个参数系统不做任何处理，只是进行缓存，当系统调用已注册回调函数时该参数会通过回调函数的参数传回。

```
struct JointStatus
{
    int    jointCurrentI;       /**< Current of driver    关节电流*/
    int    jointSpeedMoto;      /**< Speed of driver      关节速度*/
    float  jointPosJ;           /**< Current position in radian    关节角*/
    float  jointCurVol;         /**< Rated voltage of motor. Unit: mV    关节电压*/
    float  jointCurTemp;        /**< Current temprature of joint          当前温度*/
    int    jointTagCurrentI;    /**< Target current of motor               电机目标电流*/
    float  jointTagSpeedMoto;   /**< Target speed of motor                 电机目标速度*/
    float  jointTagPosJ;        /**< Target position of joint in radian  目标关节角  */
    uint16 jointErrorNum;       /**< Joint error of joint num             关节错误码 */
};
```

返回值	调用成功返回 RSERR_SUCC;错误返回错误号
示例	示例讲解：关节状态实时回调。

```
        [DllImport("libserviceinterface.dll", CallingConvention = CallingConvention.Cdecl)]
        public static extern int rs_setcallback_realtime_joint_status(UInt16 rshd, [MarshalAs(UnmanagedType.FunctionPtr)]
        RealTimeJointStatusCallback RealTimejoint_statusCallback, IntPtr arg);

        //定义委托
        [System.Runtime.InteropServices.UnmanagedFunctionPointerAttribute(System.Runtime.InteropServices.CallingConvention.Cdecl)]
        public delegate void RealTimeJointStatusCallback(ref JointStatus rs_jiont_status, IntPtr arg);
```

```
        //回调函数
        static void RealTimejoint_statusCallback(ref JointStatus rs_jiont_status, IntPtr arg)
        {
            Console.Out.WriteLine("-----------------------------------------------------------------------------------------");
            Console.Out.WriteLine("rs_jiont_status.jointCurrentI={0}", rs_jiont_status.jointCurrentI);
            Console.Out.WriteLine("rs_jiont_status.jointSpeedMoto={0}", rs_jiont_status.jointSpeedMoto);
            Console.Out.WriteLine("rs_jiont_status.jointPosJ={0}", rs_jiont_status.jointPosJ);
            Console.Out.WriteLine("rs_jiont_status.jointCurVol={0}", rs_jiont_status.jointCurVol);
            Console.Out.WriteLine("rs_jiont_status.jointCurTemp={0}", rs_jiont_status.jointCurTemp);
            Console.Out.WriteLine("rs_jiont_status.jointTagCurrentI={0}", rs_jiont_status.jointTagCurrentI);
            Console.Out.WriteLine("rs_jiont_status.jointTagSpeedMoto={0}", rs_jiont_status.jointTagSpeedMoto);
            Console.Out.WriteLine("rs_jiont_status.jointTagPosJ={0}", rs_jiont_status.jointTagPosJ);
            Console.Out.WriteLine("rs_jiont_status.jointErrorNum={0}", rs_jiont_status.jointErrorNum);
            Console.Out.WriteLine("-----------------------------------------------------------------------------------------");
        }
        #endregion
        static void Main (string[] args)
        {
            int result = 0xffff;
            UInt16 rshd = 0xffff;
            robotIP = "192.168.184.128";
            int ret;
            result = rs_initialize();
            rs_create_context(ref rshd);
            rs_login(rshd, robotIP, serverPort);
            Thread.Sleep(1000);
            //函数指针实例化
            RealTimeJointStatusCallback RobotjointstatusCallBack = new RealTimeJointStatusCallback(RealTimejoint_statusCallback);
            ret= rs_setcallback_realtime_joint_status(rshd,RobotjointstatusCallBack, IntPtr.Zero);
            Console.Out.WriteLine(ret);
            Thread.Sleep(300);
```

```
            rs_logout(rshd);
```

输出

9. **使能实时路点推送

10.实时路点信息推送 rs_setcallback_realtime_roadpoint

```
void rs_setcallback_realtime_roadpoint (UInt16 rshd, [MarshalAs (UnmanagedType.FunctionPtr)] REALTIME_ROADPOINT_CALLBACK
CurrentPositionCallback, IntPtr arg);
```

函数功能	该函数用于注册获取“实时路点信息的回调函数”到系统，成功注册后,服务器会通过回调函数实时推送机械臂当前路点信息。
参数描述	1、rshd 上下文控制句柄 2、CurrentPositionCallback 为获取实时路点信息的回调函数指针 3、arg 这个参数系统不做任何处理，只是进行缓存，当系统调用已注册回调函数时该参数会通过回调函数的参数传回。
返回值	调用成功返回 RSERR_SUCC;错误返回错误号
示例	示例讲解：路点信息实时回调。

```
        //实时路点回调函数
        [DllImport("libserviceinterface.dll", CallingConvention = CallingConvention.Cdecl)]
        public static extern int rs_setcallback_realtime_roadpoint(UInt16 rshd, [MarshalAs(UnmanagedType.FunctionPtr)]
        REALTIME_ROADPOINT_CALLBACK CurrentPositionCallback, IntPtr arg);
        //定义委托
        [System.Runtime.InteropServices.UnmanagedFunctionPointerAttribute(System.Runtime.InteropServices.CallingConvention.Cdecl)]
        public delegate void REALTIME_ROADPOINT_CALLBACK(ref wayPoint_S waypoint, IntPtr arg);
        //回调函数
        static void CurrentPositionCallback(ref wayPoint_S point, IntPtr arg)
        {
            Console.Out.WriteLine("-------------------------------------------------------------------------------------");
            Console.Out.WriteLine("pos.x={0} y={1} z={2}", point.cartPos.x, point.cartPos.y, point.cartPos.z);
            Console.Out.WriteLine("ori.w={0} x={1} y={2} z={3}", point.orientation.w, point.orientation.x,
                point.orientation.y, point.orientation.z);
            Console.Out.WriteLine("joint1={0} joint2={1} joint3={2}", point.jointpos[0] * 180 / M_PI,
                point.jointpos[1] * 180 / M_PI, point.jointpos[2] * 180 / M_PI);
            Console.Out.WriteLine("joint4={0} joint5={1} joint6={2}", point.jointpos[3] * 180 / M_PI,
                point.jointpos[4] * 180 / M_PI, point.jointpos[5] * 180 / M_PI);
            Console.Out.WriteLine("-------------------------------------------------------------------------------------");
        }
        static void Main (string[] args)
        {
            int result = 0xffff;
            UInt16 rshd = 0xffff;
```

```
robotIP = "192.168.184.128";
int ret;
result = rs_initialize();
rs_create_context(ref rshd);
rs_login(rshd, robotIP, serverPort);
Thread.Sleep(1000);
//函数指针实例化
REALTIME_ROADPOINT_CALLBACK RobotPosCallBack = new REALTIME_ROADPOINT_CALLBACK(CurrentPositionCallback);
ret=rs_setcallback_realtime_roadpoint(rshd, RobotPosCallBack, IntPtr.Zero);
Console.Out.WriteLine(ret);
Thread.Sleep(100);
rs_logout(rshd);

Console.ReadKey();
```

输出

```
0
------------------------------------------------------------------------------------
pos.x=-0.400318951099453 y=-0.121498828356558 z=0.556359716710853
ori.w=1.0873764395212E-07 x=0.707108548951157 y=-0.707105013417502 z=1.13239021253367E-07
joint1=-0.000171915288718036 joint2=-6.8082032179613 joint3=-73.8376985507203
joint4=22.970505037285 joint5=-89.9999820139107 joint6=-0.000458394166018883
------------------------------------------------------------------------------------
------------------------------------------------------------------------------------
pos.x=-0.400318951099453 y=-0.121498828356558 z=0.556359716710853
ori.w=1.0873764395212E-07 x=0.707108548951157 y=-0.707105013417502 z=1.13239021253367E-07
joint1=-0.000171915288718036 joint2=-6.8082032179613 joint3=-73.8376985507203
joint4=22.970505037285 joint5=-89.9999820139107 joint6=-0.000458394166018883
------------------------------------------------------------------------------------
------------------------------------------------------------------------------------
pos.x=-0.400318951099453 y=-0.121498828356558 z=0.556359716710853
ori.w=1.0873764395212E-07 x=0.707108548951157 y=-0.707105013417502 z=1.13239021253367E-07
joint1=-0.000171915288718036 joint2=-6.8082032179613 joint3=-73.8376985507203
joint4=22.970505037285 joint5=-89.9999820139107 joint6=-0.000458394166018883
------------------------------------------------------------------------------------
```

11.**使能实时末端速度推送

12.**实时末端速度推送

函数功能	
参数描述	
返回值	
示例	示例讲解：末端速度实时回调。
输出	

13.机械臂事件实时推送 rs_setcallback_robot_event

int rs_setcallback_robot_event(UInt16 rshd, [MarshalAs(UnmanagedType.FunctionPtr)] ROBOT_EVENT_CALLBACK RobotEventCallback, IntPtr arg);	
函数功能	该函数用于注册获取“的回调函数”到系统，成功注册后,服务器会通过回调函数实时推送事件信息。
参数描述	1、rshd 上下文控制句柄 2、RobotEventCallback 为获取实时路点信息的回调函数指针 3、arg 这个参数系统不做任何处理，只是进行缓存，当系统调用已注册回调函数时该参数会通过回调函数的参数传回。
返回值	调用成功返回 RSERR_SUCC;错误返回错误号
示例	示例讲解：机械臂事件实时回调。

```
        #region 机械臂事件回调函数
        [DllImport("libserviceinterface.dll", CallingConvention = CallingConvention.Cdecl)]
        public static extern int rs_setcallback_robot_event(UInt16 rshd, [MarshalAs(UnmanagedType.FunctionPtr)] ROBOT_EVENT_CALLBACK RobotEventCallback, IntPtr arg);

        //定义委托
        [System.Runtime.InteropServices.UnmanagedFunctionPointerAttribute(System.Runtime.InteropServices.CallingConvention.Cdecl)]
        public delegate void ROBOT_EVENT_CALLBACK(ref RobotEventInfo rs_event, IntPtr arg);

        //回调函数
```

```
        static void RobotEventCallback(ref RobotEventInfo rs_event, IntPtr arg)
        {
            Console.Out.WriteLine("-----------------------------------------------------------------------------------------");
            Console.Out.WriteLine("robot event.type={0}", rs_event.eventType);
            Console.Out.WriteLine("robot event.eventCode={0}", rs_event.eventCode);
            Console.Out.WriteLine("robot event.eventContent={0}", Marshal.PtrToStringAnsi(rs_event.eventContent));
            Console.Out.WriteLine("-----------------------------------------------------------------------------------------");
        }
        #endregion
        static void Main (string[] args)
        {
            int result = 0xffff;
            UInt16 rshd = 0xffff;
            robotIP = "192.168.184.128";
            int ret;
            result = rs_initialize();
            rs_create_context(ref rshd);
            rs_login(rshd, robotIP, serverPort);
            Thread.Sleep(1000);
            //函数指针实例化
            ROBOT_EVENT_CALLBACK RobotEventCallBack = new ROBOT_EVENT_CALLBACK(RobotEventCallback);
            ret= rs_setcallback_robot_event(rshd, RobotEventCallBack, IntPtr.Zero);
            Console.Out.WriteLine(ret);
            Thread.Sleep(3000);
```

```
            rs_logout(rshd);
```

输出	0 -- robot event.type=22 robot event.eventCode=0 robot event.eventContent={"code":0,"text":"At track Target Pos."} -- -- robot event.type=25 robot event.eventCode=1 robot event.eventContent={"code":0,"text":"robot controller state changed."} -- -- robot event.type=22 robot event.eventCode=0 robot event.eventContent={"code":0,"text":"At track Target Pos."} -- -- robot event.type=25 robot event.eventCode=1 robot event.eventContent={"code":0,"text":"robot controller state changed."} --

运动模块

14.初始化运动属性 rs_init_global_move_profile

```
int rs_init_global_move_profile (UInt16 rshd);
```

函数功能	对运动属性进行初始化，将个属性设置为初始值
参数描述	1.关节型运动的最大速度和最大加速度，默认每个关节的最大加速度为 25 度每秒方，默认每个关节的最大速度为 25 度每秒； 2.末端型运动的最大线速度和最大线加速度，默认最大线加速度为 0.03 米每秒方，默认最大线速度为 0.03 米每秒； 3.末端型运动的最大角速度和最大角加速度，默认最大角加速度为 100 度每秒方，默认最大角速度为 100 度每秒； 4.轨迹运动的路点容器，默认容器为空； 5.轨迹运动的交融半径，默认交融半径为 0； 6.轨迹运动中圆轨迹的圈数，默认为 0； 7.运动属性至偏移量属性，默认为无偏移； 8.设置示教运动的坐标系，默认为基座标系；
返回值	调用成功返回 RSERR_SUCC;错误返回错误号

示例	//初始化运动属性 rs_init_global_move_profile (rshd);
输出	

15.设置机械臂关节型运动最大加速度 rs_set_global_joint_maxacc

int rs_set_global_joint_maxacc (UInt16 rshd, double[] max_acc);	
函数功能	设置关节型运动最大加速度 关节型运动包含: * 关节运动; * 示教运动中的关节示教（JOINT1，JOINT2，JOINT3，JOINT4，JOINT5，JOINT6） * 轨迹运动下的（JIONT_CUBICSPLINE，JOINT_UBSPLINEINTP）
参数描述	设置关节型运动的最大加速度,关节型运动的最大加速度是 180°/s2，单位 rad
返回值	调用成功返回 RSERR_SUCC;错误返回错误号

示例	//设置关节运动最大加速度
	double[] joint_maxacc = { 1, 1, 1, 1, 1, 1 };
	rs_set_global_joint_maxacc(rshd, joint_maxacc);

输出	

16.设置机械臂关节型运动最大速度 rs_set_global_joint_maxvelc

int rs_set_global_joint_maxvelc (UInt16 rshd, double[] max_velc);	
函数功能	设置关节型运动最大速度 关节型运动包含: * 关节运动; * 示教运动中的关节示教（JOINT1，JOINT2，JOINT3，JOINT4，JOINT5，JOINT6） * 轨迹运动下的（JIONT_CUBICSPLINE，JOINT_UBSPLINEINTP）
参数描述	设置关节型运动的最大速度,关节型运动的最大速度是 180°/s，角度单位 rad
返回值	调用成功返回 RSERR_SUCC;错误返回错误号
示例	//设置关节运动最大速度

	double[] joint_maxvelc = { 1, 1, 1, 1, 1, 1 }; rs_set_global_joint_maxvelc(rshd, joint_maxvelc);
输出	

17.获取机械臂关节型运动最大加速度 rs_get_global_joint_maxacc

int rs_get_global_joint_maxacc (UInt16 rshd, ref JointVelcAccParam max_acc);	
函数功能	获取关节型运动最大加速度 关节型运动包含： * 关节运动； * 示教运动中的关节示教（JOINT1，JOINT2，JOINT3，JOINT4，JOINT5，JOINT6） * 轨迹运动下的（JIONT_CUBICSPLINE，JOINT_UBSPLINEINTP）
参数描述	1.rshd 上下文控制句柄 2.max_acc 是一个传出参数，表示关节运动最大加速度，单位 rad/s2
返回值	调用成功返回 RSERR_SUCC;错误返回错误号

示例	//获取关节运动最大加速度
	double[] a = { 1, 1, 1, 1.5, 1.5, 1.5 };
	rs_set_global_joint_maxacc(rshd, a);
	JointVelcAccParam joint_maxacc = new JointVelcAccParam();
	rs_get_global_joint_maxacc(rshd, ref joint_maxacc);
	Console.Out.Write("joint_maxacc={");
	for (int i=0;i<6;i++)
	{
	Console.Out.Write(joint_maxacc.jointPara[i]+",");
	}
	Console.Out.WriteLine("}");

输出	login succ joint_maxacc={1,1,1,1.5,1.5,1.5,}

18.获取机械臂关节型运动最大速度 rs_get_global_joint_maxvelc

```
int rs_get_global_joint_maxvelc (UInt16 rshd, ref JointVelcAccParam max_velc);
```

函数功能	获取关节型运动最大速度 关节型运动包含： * 关节运动； * 示教运动中的关节示教（JOINT1，JOINT2，JOINT3，JOINT4，JOINT5，JOINT6） * 轨迹运动下的（JIONT_CUBICSPLINE，JOINT_UBSPLINEINTP）
参数描述	Max_velc 是一个传出参数，表示关节运动最大速度，单位 rad/s
返回值	调用成功返回 RSERR_SUCC;错误返回错误号

示例	//获取关节运动最大速度
	double[] a = { 1, 1, 1, 1.5, 1.5, 1.5 };
	rs_set_global_joint_maxvelc(rshd, a);
	JointVelcAccParam joint_maxvelc = new JointVelcAccParam();
	rs_get_global_joint_maxvelc(rshd, ref joint_maxvelc);
	Console.Out.Write("joint_maxvelc={");
	for (int i=0;i<6;i++)
	{
	Console.Out.Write(joint_maxvelc.jointPara[i]+",");
	}
	Console.Out.WriteLine("}");

输出	login succ joint_maxvelc={1,1,1,1.5,1.5,1.5,}

19.设置末端型运动最大加速度 rs_set_global_end_max_line_acc

```
int rs_set_global_end_max_line_acc (UInt16 rshd, double max_acc);
```

函数功能	设置末端型运动最大加速度 末端型包含： 直线运动（MODEL）； * 示教运动中的位置示教和姿态示教（MOV_X，MOV_Y，MOV_Z，ROT_X，ROT_Y，ROT_Z）； * 轨迹运动下的（ARC_CIR， CARTESIAN_MOVEP， CARTESIAN_CUBICSPLINE， CARTESIAN_UBSPLINEINTP）

参数描述	末端型运动最大加速度，最大 2m/s2。
返回值	调用成功返回 RSERR_SUCC;错误返回错误号

示例	`//设置末端型运动最大加速度`
	`double max_acc = 0.5;`
	`rs_set_global_end_max_line_acc(rshd, max_acc);`

输出	

20.设置末端型运动最大速度 rs_set_global_end_max_line_velc

`int rs_set_global_end_max_line_velc (UInt16 rshd, double max_velc);`	
函数功能	设置末端型运动最大速度 末端型包含：　直线运动（MODEL）; 　　*　　示教运动中的位置示教和姿态示教（MOV_X，MOV_Y，MOV_Z，ROT_X，ROT_Y，ROT_Z）; 　　*　　轨迹运动下的（ARC_CIR，　CARTESIAN_MOVEP，　CARTESIAN_CUBICSPLINE，　CARTESIAN_UBSPLINEINTP）
参数描述	末端型运动最大速度，最大 2m/s。
返回值	调用成功返回 RSERR_SUCC;错误返回错误号

示例	`//设置末端型运动最大速度`
	`double max_velc = 0.5;`
	`rs_set_global_end_max_line_acc(rshd, max_velc);`

输出	

21.获取末端型运动最大加速度 rs_get_global_end_max_line_acc

`int rs_get_global_end_max_line_acc (UInt16 rshd, double max_acc);`	
函数功能	获取末端型运动最大加速度 末端型包含：　直线运动（MODEL）; 　　*　　示教运动中的位置示教和姿态示教（MOV_X，MOV_Y，MOV_Z，ROT_X，ROT_Y，ROT_Z）; 　　*　　轨迹运动下的（ARC_CIR，　CARTESIAN_MOVEP，　CARTESIAN_CUBICSPLINE，　CARTESIAN_UBSPLINEINTP）
参数描述	max_acc 是一个传出参数，末端型运动最大加速度，最大 2m/s2。
返回值	调用成功返回 RSERR_SUCC;错误返回错误号

示例	//获取末端型运动最大加速度
	double max_acc = 0;
	rs_get_global_end_max_line_acc(rshd, ref max_acc);
	Console.Out.Write("max_end_acc="+ max_acc);

输出	login succ max_end_acc=0.03

22.获取末端型运动最大速度 rs_get_global_end_max_line_velc

int rs_get_global_end_max_line_velc (UInt16 rshd, double max_velc);	
函数功能	获取末端型运动最大速度 末端型包含： 直线运动（MODEL）; * 示教运动中的位置示教和姿态示教（MOV_X，MOV_Y，MOV_Z，ROT_X，ROT_Y，ROT_Z）; * 轨迹运动下的（ARC_CIR， CARTESIAN_MOVEP， CARTESIAN_CUBICSPLINE， CARTESIAN_UBSPLINEINTP）
参数描述	max_velc 是一个传出参数，末端型运动最大速度，最大 2m/s。
返回值	调用成功返回 RSERR_SUCC;错误返回错误号

示例	//获取末端型运动最大速度
	double max_velc = 0;
	rs_get_global_end_max_line_velc(rshd, ref max_velc);
	Console.Out.Write("max_end_velc="+ max_velc);

输出	login succ max_end_velc=0.03

23.**设置末端型运动旋转最大角加速度

24.**设置末端型运动旋转最大角速度

25.**获取末端型运动旋转最大角加速度

26.**获取末端型运动旋转最大角速度

27.清空路点容器 rs_remove_all_waypoint

int rs_remove_all_waypoint (UInt16 rshd);	
函数功能	清除路点容器，通常用于新的轨迹运动之前
参数描述	
返回值	调用成功返回 RSERR_SUCC;错误返回错误号
示例	//清空全局路点容器 rs_remove_all_waypoint(rshd);;
输出	

28.添加全局路点 rs_add_waypoint

int rs_add_waypoint (UInt16 rshd, double[] joint_radia);	
函数功能	添加路点，用于轨迹运动
参数描述	关节角形式路点
返回值	调用成功返回 RSERR_SUCC;错误返回错误号
示例	double[] wp1 = { -0.000003, -0.127267, -1.321122, 0.376941, -1.570796, -0.000008 }; rs_add_waypoint(rshd, wp1);

输出	

29.设置轨迹运动中的交融半径 rs_set_blend_radius

```
int rs_set_blend_radius (UInt16 rshd, double radius);
```

函数功能	设置轨迹运动如 movep 的交融半径
参数描述	0.0 ~ 0.05，单位 m
返回值	调用成功返回 RSERR_SUCC;错误返回错误号

示例	`//设置交融半径`
	`double radius=0.01;`
	`rs_set_blend_radius(rshd, radius);`

输出	

30.设置圆运动圈数 rs_set_circular_loop_times

```
int rs_set_circular_loop_times (UInt16 rshd, int times);
```

函数功能	设置圆运动圈数，在轨迹类型为 ARD_CIR 时有效
参数描述	times = 0 时，表示圆弧运动; times > 0 时，表示圆运动，且表示圆的圈数
返回值	调用成功返回 RSERR_SUCC;错误返回错误号
示例	`//设置圆运动圈数` `rs_set_circular_loop_times(rshd, 2);`
输出	

31.设置相对偏移属性 rs_set_relative_offset_on_base

```
int rs_set_relative_offset_on_base (UInt16 rshd, ref MoveRelative relative);
```

函数功能	Base 坐标系下的相对偏移属性设置

参数描述	struct MoveRelative {
	//是否使能偏移
	public byte enable;
	//偏移量 x,y,z
	[MarshalAs (UnmanagedType.ByValArray, SizeConst = 3)]
	public float[] pos;
	//public Pos pos;
	//相对姿态偏移量
	public Ori orientation;
	}

返回值	调用成功返回 RSERR_SUCC;错误返回错误号

示例	示例讲解：在 Base 坐标系下沿 Z 方向偏移 0.1m。
	//相对位移
	MoveRelative relative = new MoveRelative();
	IntPtr pt_relative = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(MoveRelative)));
	relative = (MoveRelative)Marshal.PtrToStructure(pt_relative, typeof(MoveRelative));
	relative.enable = 1;
	relative.orientation.w = 1;
	relative.orientation.x = 0;
	relative.orientation.y = 0;
	relative.orientation.z = 0;
	relative.pos[0] = 0;
	relative.pos[1] = 0;
	relative.pos[2] = -0.1F;
	//移动到坐标系原点
	double[] target0 = { 0, 0, 0, 0, 0, 0 }; //注意这个里面的值是弧度!
	target0[0] = -0.000172 / 180 * M_PI;
	target0[1] = -7.291862 / 180 * M_PI;
	target0[2] = -75.694718 / 180 * M_PI;
	target0[3] = 21.596727 / 180 * M_PI;
	target0[4] = -89.999982 / 180 * M_PI;
	target0[5] = -0.00458 / 180 * M_PI;
	rs_move_joint(rshd, target0, true);
	//相对坐标系原点沿z轴正向运动
	rs_set_relative_offset_on_base(rshd, ref relative);
	rs_move_line(rshd, target0, true);

输出	

32.设置相对偏移属性 rs_set_relative_offset_on_user

`int rs_set_relative_offset_on_user (UInt16 rshd, ref MoveRelative relative, ref CoordCalibrate user_coord);`	
函数功能	User 坐标系或工具坐标系下的相对偏移属性设置
参数描述	relative：相对偏移 userCoord：用户坐标系或者工具坐标系
返回值	调用成功返回 RSERR_SUCC;错误返回错误号
示例 1	示例讲解：在 Base 坐标系下沿 X+方向偏移 0.1m。

```
CoordCalibrate user_coord = new CoordCalibrate();
IntPtr pt_user_coord = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(CoordCalibrate)));
user_coord = (CoordCalibrate)Marshal.PtrToStructure(pt_user_coord, typeof(CoordCalibrate));
//检查用户坐标系参数设置是否合理
user_coord.coordType = BaseCoordinate;
//坐标系标定方法xoxy
user_coord.methods = Origin_AnyPointOnPositiveXAxis_AnyPointOnFirstQuadrantOfXOYPlane;
//为了简化，使用法兰盘中心为工具端
user_coord.toolDesc.cartPos.x = 0;
user_coord.toolDesc.cartPos.y = 0;
user_coord.toolDesc.cartPos.z = 0;
user_coord.toolDesc.orientation.w = 1;
user_coord.toolDesc.orientation.x = 0;
user_coord.toolDesc.orientation.y = 0;
user_coord.toolDesc.orientation.z = 0;
//原点
user_coord.jointPara[0].jointRadian[0] = -0.000172 / 180 * M_PI;
user_coord.jointPara[0].jointRadian[1] = -7.291862 / 180 * M_PI;
user_coord.jointPara[0].jointRadian[2] = -75.694718 / 180 * M_PI;
user_coord.jointPara[0].jointRadian[3] = 21.596727 / 180 * M_PI;
user_coord.jointPara[0].jointRadian[4] = -89.999982 / 180 * M_PI;
user_coord.jointPara[0].jointRadian[5] = -0.00458 / 180 * M_PI;
//x轴正半轴上一点
user_coord.jointPara[1].jointRadian[0] = 11.116932 / 180 * M_PI;
user_coord.jointPara[1].jointRadian[1] = -0.858816 / 180 * M_PI;
user_coord.jointPara[1].jointRadian[2] = -64.008663 / 180 * M_PI;
user_coord.jointPara[1].jointRadian[3] = 26.850182 / 180 * M_PI;
user_coord.jointPara[1].jointRadian[4] = -90.000064 / 180 * M_PI;
```

```
user_coord.jointPara[1].jointRadian[5] = 11.116645 / 180 * M_PI;
//x、y轴平面的第一象限上任意一点
user_coord.jointPara[2].jointRadian[0] = 9.012562 / 180 * M_PI;
user_coord.jointPara[2].jointRadian[1] = 13.378931 / 180 * M_PI;
user_coord.jointPara[2].jointRadian[2] = -47.871256 / 180 * M_PI;
user_coord.jointPara[2].jointRadian[3] = 28.749813 / 180 * M_PI;
user_coord.jointPara[2].jointRadian[4] = -90.000064 / 180 * M_PI;
user_coord.jointPara[2].jointRadian[5] = 9.012275 / 180 * M_PI;
//相对位移
MoveRelative relative = new MoveRelative();
IntPtr pt_relative = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(MoveRelative)));
relative = (MoveRelative)Marshal.PtrToStructure(pt_relative, typeof(MoveRelative));
relative.enable = 1;
relative.orientation.w = 1;
relative.orientation.x = 0;
relative.orientation.y = 0;
relative.orientation.z = 0;
relative.pos[0] = 0;
relative.pos[1] = 0;
relative.pos[2] = -0.1F;
//移动到坐标系原点
double[] target0 = { 0, 0, 0, 0, 0, 0 }; //注意这个里面的值是弧度!
target0[0] = -0.000172 / 180 * M_PI;
target0[1] = -7.291862 / 180 * M_PI;
target0[2] = -75.694718 / 180 * M_PI;
target0[3] = 21.596727 / 180 * M_PI;
target0[4] = -89.999982 / 180 * M_PI;
target0[5] = -0.00458 / 180 * M_PI;
rs_move_joint(rshd, target0, true);
//相对坐标系原点沿z轴正向运动
rs_set_relative_offset_on_user(rshd, ref relative, ref user_coord);
rs_move_line(rshd, target0, true);
```

输出

示例 2

示例讲解：工具坐标系（{0,0,0},{1,0,0,0}）下沿 X+方向相对偏移 0.1m。

```
CoordCalibrate user_coord = new CoordCalibrate();
IntPtr pt_user_coord = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(CoordCalibrate)));
user_coord = (CoordCalibrate)Marshal.PtrToStructure(pt_user_coord, typeof(CoordCalibrate));
//检查用户坐标系参数设置是否合理
user_coord.coordType = EndCoordinate;
//坐标系标定方法xoxy
```

```
                user_coord.methods = Origin_AnyPointOnPositiveXAxis_AnyPointOnFirstQuadrantOfXOYPlane;
                //为了简化，使用法兰盘中心为工具端
                user_coord.toolDesc.cartPos.x = 0;
                user_coord.toolDesc.cartPos.y = 0;
                user_coord.toolDesc.cartPos.z = 0;
                user_coord.toolDesc.orientation.w = 1;
                user_coord.toolDesc.orientation.x = 0;
                user_coord.toolDesc.orientation.y = 0;
                user_coord.toolDesc.orientation.z = 0;
                //原点
                user_coord.jointPara[0].jointRadian[0] = -0.000172 / 180 * M_PI;
                user_coord.jointPara[0].jointRadian[1] = -7.291862 / 180 * M_PI;
                user_coord.jointPara[0].jointRadian[2] = -75.694718 / 180 * M_PI;
                user_coord.jointPara[0].jointRadian[3] = 21.596727 / 180 * M_PI;
                user_coord.jointPara[0].jointRadian[4] = -89.999982 / 180 * M_PI;
                user_coord.jointPara[0].jointRadian[5] = -0.00458 / 180 * M_PI;
                //x轴正半轴上一点
                user_coord.jointPara[1].jointRadian[0] = 11.116932 / 180 * M_PI;
                user_coord.jointPara[1].jointRadian[1] = -0.858816 / 180 * M_PI;
                user_coord.jointPara[1].jointRadian[2] = -64.008663 / 180 * M_PI;
                user_coord.jointPara[1].jointRadian[3] = 26.850182 / 180 * M_PI;
                user_coord.jointPara[1].jointRadian[4] = -90.000064 / 180 * M_PI;
                user_coord.jointPara[1].jointRadian[5] = 11.116645 / 180 * M_PI;
                //x、y轴平面的第一象限上任意一点
                user_coord.jointPara[2].jointRadian[0] = 9.012562 / 180 * M_PI;
                user_coord.jointPara[2].jointRadian[1] = 13.378931 / 180 * M_PI;
                user_coord.jointPara[2].jointRadian[2] = -47.871256 / 180 * M_PI;
                user_coord.jointPara[2].jointRadian[3] = 28.749813 / 180 * M_PI;
                user_coord.jointPara[2].jointRadian[4] = -90.000064 / 180 * M_PI;
                user_coord.jointPara[2].jointRadian[5] = 9.012275 / 180 * M_PI;
                //相对位移
                MoveRelative relative = new MoveRelative();
                IntPtr pt_relative = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(MoveRelative)));
                relative = (MoveRelative)Marshal.PtrToStructure(pt_relative, typeof(MoveRelative));
                relative.enable = 1;
                relative.orientation.w = 1;
                relative.orientation.x = 0;
                relative.orientation.y = 0;
                relative.orientation.z = 0;
                relative.pos[0] = 0;
                relative.pos[1] = 0;
```

```
relative.pos[2] = -0.1F;
//移动到坐标系原点
double[] target0 = { 0, 0, 0, 0, 0, 0 }; //注意这个里面的值是弧度!
target0[0] = -0.000172 / 180 * M_PI;
target0[1] = -7.291862 / 180 * M_PI;
target0[2] = -75.694718 / 180 * M_PI;
target0[3] = 21.596727 / 180 * M_PI;
target0[4] = -89.999982 / 180 * M_PI;
target0[5] = -0.00458 / 180 * M_PI;
rs_move_joint(rshd, target0, true);
//相对坐标系原点沿z轴正向运动
rs_set_relative_offset_on_user(rshd, ref relative, ref user_coord);
rs_move_line(rshd, target0, true);
```

输出

示例 3

示例讲解：User 坐标系（p1）下沿 X+方向偏移 0.1m。

```
CoordCalibrate user_coord = new CoordCalibrate();
IntPtr pt_user_coord = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(CoordCalibrate)));
user_coord = (CoordCalibrate)Marshal.PtrToStructure(pt_user_coord, typeof(CoordCalibrate));
//检查用户坐标系参数设置是否合理
user_coord.coordType = WorldCoordinate;
//坐标系标定方法xoxy
user_coord.methods = Origin_AnyPointOnPositiveXAxis_AnyPointOnFirstQuadrantOfXOYPlane;
//为了简化，使用法兰盘中心为工具端
user_coord.toolDesc.cartPos.x = 0;
user_coord.toolDesc.cartPos.y = 0;
user_coord.toolDesc.cartPos.z = 0;
user_coord.toolDesc.orientation.w = 1;
user_coord.toolDesc.orientation.x = 0;
user_coord.toolDesc.orientation.y = 0;
user_coord.toolDesc.orientation.z = 0;
//原点
user_coord.jointPara[0].jointRadian[0] = -0.000172 / 180 * M_PI;
user_coord.jointPara[0].jointRadian[1] = -7.291862 / 180 * M_PI;
user_coord.jointPara[0].jointRadian[2] = -75.694718 / 180 * M_PI;
user_coord.jointPara[0].jointRadian[3] = 21.596727 / 180 * M_PI;
user_coord.jointPara[0].jointRadian[4] = -89.999982 / 180 * M_PI;
user_coord.jointPara[0].jointRadian[5] = -0.00458 / 180 * M_PI;
//x轴正半轴上一点
user_coord.jointPara[1].jointRadian[0] = 11.116932 / 180 * M_PI;
user_coord.jointPara[1].jointRadian[1] = -0.858816 / 180 * M_PI;
```

	user_coord.jointPara[1].jointRadian[2] = -64.008663 / 180 * M_PI;
	user_coord.jointPara[1].jointRadian[3] = 26.850182 / 180 * M_PI;
	user_coord.jointPara[1].jointRadian[4] = -90.000064 / 180 * M_PI;
	user_coord.jointPara[1].jointRadian[5] = 11.116645 / 180 * M_PI;
	//x、y轴平面的第一象限上任意一点
	user_coord.jointPara[2].jointRadian[0] = 9.012562 / 180 * M_PI;
	user_coord.jointPara[2].jointRadian[1] = 13.378931 / 180 * M_PI;
	user_coord.jointPara[2].jointRadian[2] = -47.871256 / 180 * M_PI;
	user_coord.jointPara[2].jointRadian[3] = 28.749813 / 180 * M_PI;
	user_coord.jointPara[2].jointRadian[4] = -90.000064 / 180 * M_PI;
	user_coord.jointPara[2].jointRadian[5] = 9.012275 / 180 * M_PI;
	//相对位移
	MoveRelative relative = new MoveRelative();
	IntPtr pt_relative = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(MoveRelative)));
	relative = (MoveRelative)Marshal.PtrToStructure(pt_relative, typeof(MoveRelative));
	relative.enable = 1;
	relative.orientation.w = 1;
	relative.orientation.x = 0;
	relative.orientation.y = 0;
	relative.orientation.z = 0;
	relative.pos[0] = 0;
	relative.pos[1] = 0;
	relative.pos[2] = -0.1F;
	//移动到坐标系原点
	double[] target0 = { 0, 0, 0, 0, 0, 0 }; //注意这个里面的值是弧度!
	target0[0] = -0.000172 / 180 * M_PI;
	target0[1] = -7.291862 / 180 * M_PI;
	target0[2] = -75.694718 / 180 * M_PI;
	target0[3] = 21.596727 / 180 * M_PI;
	target0[4] = -89.999982 / 180 * M_PI;
	target0[5] = -0.00458 / 180 * M_PI;
	rs_move_joint(rshd, target0, true);
	//相对坐标系原点沿z轴正向运动
	rs_set_relative_offset_on_user(rshd, ref relative, ref user_coord);
	rs_move_line(rshd, target0, true);

输出	

33.设置无提前到位 rs_set_no_arrival_ahead

int rs_set_no_arrival_ahead (UInt16 rshd);	
函数功能	设置无提前到位
参数描述	
返回值	调用成功返回 RSERR_SUCC;错误返回错误号

示例	double[] wp1 = {0,0,0,0,0,0 };
	wp1[0] = 0 * M_PI / 180;
	wp1[1] = -7.29 * M_PI / 180;
	wp1[2] = -75.69 * M_PI / 180;
	wp1[3] = 21.59 * M_PI / 180;
	wp1[4] =-90 * M_PI / 180;
	wp1[5] = 0 * M_PI / 180;
	double[] wp2 = { 0, 0, 0, 0, 0, 0 };
	wp2[0] = 0 * M_PI / 180;
	wp2[1] = -11 * M_PI / 180;
	wp2[2] = -100 * M_PI / 180;
	wp2[3] =0.7 * M_PI / 180;
	wp2[4] = -90 * M_PI / 180;
	wp2[5] = 0 * M_PI / 180;
	double[] wp3 = { 0, 0, 0, 0, 0, 0 };
	wp3[0] = 17 * M_PI / 180;
	wp3[1] = -3 * M_PI / 180;
	wp3[2] = -93 * M_PI / 180;
	wp3[3] = -0.3 * M_PI / 180;
	wp3[4] = -90 * M_PI / 180;
	wp3[5] = 17 * M_PI / 180;
	//关节运动
	rs_move_joint(rshd, wp1, true);
	
	/*设置无提前到位*/
	rs_set_no_arrival_ahead(rshd);
	/*运动测试*/
	for (int i = 0; i < 2; i++)
	{
	rs_move_joint(rshd, wp2, true);
	rs_move_joint(rshd, wp3, true);
	rs_move_joint(rshd, wp1, true);

	} }
输出	

34.设置提前到位距离模式 rs_set_arrival_ahead_distance

```
int rs_set_arrival_ahead_distance (UInt16 rshd, double distance);
```

函数功能	设置提前到位距离模式
参数描述	distance：距离。单位 m
返回值	调用成功返回 RSERR_SUCC;错误返回错误号

示例	double[] wp1 = {0,0,0,0,0,0 };
	wp1[0] = 0 * M_PI / 180;
	wp1[1] = -7.29 * M_PI / 180;
	wp1[2] = -75.69 * M_PI / 180;
	wp1[3] = 21.59 * M_PI / 180;
	wp1[4] =-90 * M_PI / 180;
	wp1[5] = 0 * M_PI / 180;
	double[] wp2 = { 0, 0, 0, 0, 0, 0 };
	wp2[0] = 0 * M_PI / 180;
	wp2[1] = -11 * M_PI / 180;
	wp2[2] = -100 * M_PI / 180;
	wp2[3] =0.7 * M_PI / 180;
	wp2[4] = -90 * M_PI / 180;
	wp2[5] = 0 * M_PI / 180;
	double[] wp3 = { 0, 0, 0, 0, 0, 0 };
	wp3[0] = 17 * M_PI / 180;
	wp3[1] = -3 * M_PI / 180;
	wp3[2] = -93 * M_PI / 180;
	wp3[3] = -0.3 * M_PI / 180;
	wp3[4] = -90 * M_PI / 180;
	wp3[5] = 17 * M_PI / 180;
	//关节运动
	rs_move_joint(rshd, wp1, true);
	
	/*设置提前到位距离模式*/
	rs_set_arrival_ahead_distance(rshd, 0.05);
	/*运动测试*/

	for (int i = 0; i < 2; i++)
	{
	rs_move_joint(rshd, wp2, true);
	rs_move_joint(rshd, wp3, true);
	rs_move_joint(rshd, wp1, true);
	}

输出	

35.设置提前到位时间模式 rs_set_arrival_ahead_time

int rs_set_arrival_ahead_time (UInt16 rshd, double sec);	
函数功能	设置提前到位时间模式
参数描述	second：时间。单位 s
返回值	调用成功返回 RSERR_SUCC;错误返回错误号

示例	double[] wp1 = {0,0,0,0,0,0 };
	wp1[0] = 0 * M_PI / 180;
	wp1[1] = -7.29 * M_PI / 180;
	wp1[2] = -75.69 * M_PI / 180;
	wp1[3] = 21.59 * M_PI / 180;
	wp1[4] =-90 * M_PI / 180;
	wp1[5] = 0 * M_PI / 180;
	double[] wp2 = { 0, 0, 0, 0, 0, 0 };
	wp2[0] = 0 * M_PI / 180;
	wp2[1] = -11 * M_PI / 180;
	wp2[2] = -100 * M_PI / 180;
	wp2[3] =0.7 * M_PI / 180;
	wp2[4] = -90 * M_PI / 180;
	wp2[5] = 0 * M_PI / 180;
	double[] wp3 = { 0, 0, 0, 0, 0, 0 };
	wp3[0] = 17 * M_PI / 180;
	wp3[1] = -3 * M_PI / 180;
	wp3[2] = -93 * M_PI / 180;
	wp3[3] = -0.3 * M_PI / 180;
	wp3[4] = -90 * M_PI / 180;
	wp3[5] = 17 * M_PI / 180;
	//关节运动
	rs_move_joint(rshd, wp1, true);

	/*设置提前到位时间模式*/
	rs_set_arrival_ahead_time(rshd, 1);
	/*运动测试*/
	for (int i = 0; i < 2; i++)
	{
	rs_move_joint(rshd, wp2, true);
	rs_move_joint(rshd, wp3, true);
	rs_move_joint(rshd, wp1, true);
	}

输出	

36.设置提前到位交融半径模式 rs_set_arrival_ahead_blend

int rs_set_arrival_ahead_blend(UInt16 rshd, double radius);	
函数功能	设置提前到位时间模式
参数描述	second：时间。单位 s
返回值	调用成功返回 RSERR_SUCC;错误返回错误号

示例	double[] wp1 = {0,0,0,0,0,0 };
	wp1[0] = 0 * M_PI / 180;
	wp1[1] = -7.29 * M_PI / 180;
	wp1[2] = -75.69 * M_PI / 180;
	wp1[3] = 21.59 * M_PI / 180;
	wp1[4] =-90 * M_PI / 180;
	wp1[5] = 0 * M_PI / 180;
	double[] wp2 = { 0, 0, 0, 0, 0, 0 };
	wp2[0] = 0 * M_PI / 180;
	wp2[1] = -11 * M_PI / 180;
	wp2[2] = -100 * M_PI / 180;
	wp2[3] =0.7 * M_PI / 180;
	wp2[4] = -90 * M_PI / 180;
	wp2[5] = 0 * M_PI / 180;
	double[] wp3 = { 0, 0, 0, 0, 0, 0 };
	wp3[0] = 17 * M_PI / 180;
	wp3[1] = -3 * M_PI / 180;
	wp3[2] = -93 * M_PI / 180;
	wp3[3] = -0.3 * M_PI / 180;

	wp3[4] = -90 * M_PI / 180;
	wp3[5] = 17 * M_PI / 180;
	//关节运动
	rs_move_joint(rshd, wp1, true);
	
	/*设置距离模式下交融半径距离*/
	rs_set_arrival_ahead_blend(rshd, 0.05);
	/*运动测试*/
	for (int i = 0; i < 2; i++)
	{
	rs_move_joint(rshd, wp2, true);
	rs_move_joint(rshd, wp3, true);
	rs_move_joint(rshd, wp1, true);
	}

输出	

37.设置示教坐标系 rs_set_teach_coord

int rs_set_teach_coord(UInt16 rshd, ref CoordCalibrate teach_coord);	
函数功能	设置示教运动的坐标系
参数描述	teach_coord: 示教运动的坐标系
返回值	调用成功返回 RSERR_SUCC;错误返回错误号

示例	/**** 设置 User 坐标系(p1) *****/
	CoordCalibrate user_coord = new CoordCalibrate();
	IntPtr pt_user_coord = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(CoordCalibrate)));
	user_coord = (CoordCalibrate)Marshal.PtrToStructure(pt_user_coord, typeof(CoordCalibrate));
	//坐标系类型
	user_coord.coordType = EndCoordinate;
	//坐标系标定方法xoxy
	user_coord.methods = Origin_AnyPointOnPositiveXAxis_AnyPointOnFirstQuadrantOfXOYPlane;
	//为了简化，使用法兰盘中心为工具端
	user_coord.toolDesc.cartPos.x = 0;
	user_coord.toolDesc.cartPos.y = 0;
	user_coord.toolDesc.cartPos.z = 0;
	user_coord.toolDesc.orientation.w = 1;

	user_coord.toolDesc.orientation.x = 0;
	user_coord.toolDesc.orientation.y = 0;
	user_coord.toolDesc.orientation.z = 0;
	//原点
	user_coord.jointPara[0].jointRadian[0] = -0.000172 / 180 * M_PI;
	user_coord.jointPara[0].jointRadian[1] = -7.291862 / 180 * M_PI;
	user_coord.jointPara[0].jointRadian[2] = -75.694718 / 180 * M_PI;
	user_coord.jointPara[0].jointRadian[3] = 21.596727 / 180 * M_PI;
	user_coord.jointPara[0].jointRadian[4] = -89.999982 / 180 * M_PI;
	user_coord.jointPara[0].jointRadian[5] = -0.00458 / 180 * M_PI;
	//x轴正半轴上一点
	user_coord.jointPara[1].jointRadian[0] = 11.116932 / 180 * M_PI;
	user_coord.jointPara[1].jointRadian[1] = -0.858816 / 180 * M_PI;
	user_coord.jointPara[1].jointRadian[2] = -64.008663 / 180 * M_PI;
	user_coord.jointPara[1].jointRadian[3] = 26.850182 / 180 * M_PI;
	user_coord.jointPara[1].jointRadian[4] = -90.000064 / 180 * M_PI;
	user_coord.jointPara[1].jointRadian[5] = 11.116645 / 180 * M_PI;
	//x、y轴平面的第一象限上任意一点
	user_coord.jointPara[2].jointRadian[0] = 9.012562 / 180 * M_PI;
	user_coord.jointPara[2].jointRadian[1] = 13.378931 / 180 * M_PI;
	user_coord.jointPara[2].jointRadian[2] = -47.871256 / 180 * M_PI;
	user_coord.jointPara[2].jointRadian[3] = 28.749813 / 180 * M_PI;
	user_coord.jointPara[2].jointRadian[4] = -90.000064 / 180 * M_PI;
	user_coord.jointPara[2].jointRadian[5] = 9.012275 / 180 * M_PI;
	rs_set_teach_coord(rshd, ref user_coord);

输出	

38.关节运动 rs_move_joint

```
int rs_move_joint (UInt16 rshd, double[] joint_radia, bool isblock);
```

函数功能	关节（轴动）运动至目标点位
参数描述	**joint_radia**：以关节角表示路点。 **IsBolck==true** 代表阻塞，机械臂运动直到到达目标位置或者出现故障后返回。 **IsBolck==false** 代表非阻塞，立即返回，运动指令发送成功就返回，函数返回后机械臂开始运动。
返回值	调用成功返回 RSERR_SUCC;错误返回错误号

示例	double[] wp1 = { 0, 0, 0, 0, 0, 0 };
	wp1[0] = 0 * M_PI / 180;
	wp1[1] = -7.29 * M_PI / 180;
	wp1[2] = -75.69 * M_PI / 180;
	wp1[3] = 21.59 * M_PI / 180;
	wp1[4] = -90 * M_PI / 180;
	wp1[5] = 0 * M_PI / 180;
	double[] wp2 = { 0, 0, 0, 0, 0, 0 };
	wp2[0] = 0 * M_PI / 180;
	wp2[1] = -11 * M_PI / 180;
	wp2[2] = -100 * M_PI / 180;
	wp2[3] = 0.7 * M_PI / 180;
	wp2[4] = -90 * M_PI / 180;
	wp2[5] = 0 * M_PI / 180;
	double[] wp3 = { 0, 0, 0, 0, 0, 0 };
	wp3[0] = 17 * M_PI / 180;
	wp3[1] = -3 * M_PI / 180;
	wp3[2] = -93 * M_PI / 180;
	wp3[3] = -0.3 * M_PI / 180;
	wp3[4] = -90 * M_PI / 180;
	wp3[5] = 17 * M_PI / 180;
	rs_move_joint(rshd, wp1, true);
	rs_move_joint(rshd, wp2, true);
	rs_move_joint(rshd, wp3, true);

输出	

39.直线运动 rs_move_line

int rs_move_line (UInt16 rshd, double[] joint_radia, bool isblock);	
函数功能	直线运动至目标点位
参数描述	joint_radia：以关节角表示路点。 **IsBolck==true** 代表阻塞，机械臂运动直到到达目标位置或者出现故障后返回。 **IsBolck==false** 代表非阻塞，立即返回，运动指令发送成功就返回，函数返回后机械臂开始运动。
返回值	调用成功返回 RSERR_SUCC;错误返回错误号
示例	double[] wp1 = { 0, 0, 0, 0, 0, 0 }; wp1[0] = 0 * M_PI / 180;

	wp1[1] = -7.29 * M_PI / 180;
	wp1[2] = -75.69 * M_PI / 180;
	wp1[3] = 21.59 * M_PI / 180;
	wp1[4] = -90 * M_PI / 180;
	wp1[5] = 0 * M_PI / 180;
	double[] wp2 = { 0, 0, 0, 0, 0, 0 };
	wp2[0] = 0 * M_PI / 180;
	wp2[1] = -11 * M_PI / 180;
	wp2[2] = -100 * M_PI / 180;
	wp2[3] = 0.7 * M_PI / 180;
	wp2[4] = -90 * M_PI / 180;
	wp2[5] = 0 * M_PI / 180;
	double[] wp3 = { 0, 0, 0, 0, 0, 0 };
	wp3[0] = 17 * M_PI / 180;
	wp3[1] = -3 * M_PI / 180;
	wp3[2] = -93 * M_PI / 180;
	wp3[3] = -0.3 * M_PI / 180;
	wp3[4] = -90 * M_PI / 180;
	wp3[5] = 17 * M_PI / 180;
	rs_move_line(rshd, wp1, true);
	rs_move_line(rshd, wp2, true);
	rs_move_line(rshd, wp3, true);

输出	

40.旋转运动 rs_move_rotate

int rs_move_rotate (UInt16 rshd, ref CoordCalibrate user_coord, ref MoveRotateAxis rotate_axis, double rotate_angle, bool isblock);	
函数功能	保持当前位置变换姿态做旋转运动
参数描述	参数 1：user_coord 坐标系 参数 2：rotate_axis。转轴[x,y,z]　　当绕 X 转，[1,0,0] 参数 3：rotate_angle。转角，单位 rad。 参数 4：是否阻塞。 IsBolck==true　代表阻塞，机械臂运动直到到达目标位置或者出现故障后返回。 IsBolck==false 代表非阻塞，立即返回，运动指令发送成功就返回，函数返回后机械臂开始运动。
返回值	调用成功返回 RSERR_SUCC;错误返回错误号

示例 1	示例讲解：flange_center 绕 Base 坐标系 X 轴旋转 30°。
	
	//初始化运动属性
	robotService.robotServiceInitGlobalMoveProfile();
	
	double wp1[6] = {};
	wp1[0] = 60.443151*M_PI/180;
	wp1[1] = 42.275463*M_PI/180;
	wp1[2] = -97.679737*M_PI/180;
	wp1[3] = -49.990510*M_PI/180;
	wp1[4] = -90.007372*M_PI/180;
	wp1[5] = 62.567046*M_PI/180;
	
	//关节运动
	robotService.robotServiceJointMove(wp1,true);
	
	//设置末端型运动最大加速度
	double m_set_end_lineacc = 0.2;
	robotService.robotServiceSetGlobalMoveEndMaxLineAcc(m_set_end_lineacc);
	//设置末端型运动最大速度
	double m_set_end_linevelc = 0.2;
	robotService.robotServiceSetGlobalMoveEndMaxLineVelc(m_set_end_linevelc);
	
	//设置 Base 坐标系
	aubo_robot_namespace::CoordCalibrateByJointAngleAndTool userCoord;
	userCoord.coordType = aubo_robot_namespace::BaseCoordinate;
	
	//rotate 旋转运动
	double rotateAxis[3] = {0,0,1};
	double rotateAngle1 = 30*M_PI/180;
	double rotateAngle2 = -30*M_PI/180;
	for(int i = 0; i < 2; i++)
	{
	robotService.robotServiceRotateMove(userCoord,rotateAxis,rotateAngle1,true);
	sleep(1);
	robotService.robotServiceRotateMove(userCoord,rotateAxis,rotateAngle2,true);
	sleep(1);
	}

输出	
示例 2	示例讲解：flange_center 绕 User 坐标系（p1）的 Z 轴旋转 30°。

```
            CoordCalibrate user_coord = new CoordCalibrate();
            IntPtr pt_user_coord = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(CoordCalibrate)));
            user_coord = (CoordCalibrate)Marshal.PtrToStructure(pt_user_coord, typeof(CoordCalibrate));
            //检查用户坐标系参数设置是否合理
            user_coord.coordType = WorldCoordinate;
            //坐标系标定方法xoxy
            user_coord.methods = Origin_AnyPointOnPositiveXAxis_AnyPointOnFirstQuadrantOfXOYPlane;
            //为了简化，使用法兰盘中心为工具端
            user_coord.toolDesc.cartPos.x = 0;
            user_coord.toolDesc.cartPos.y = 0;
            user_coord.toolDesc.cartPos.z = 0;
            user_coord.toolDesc.orientation.w = 1;
            user_coord.toolDesc.orientation.x = 0;
            user_coord.toolDesc.orientation.y = 0;
            user_coord.toolDesc.orientation.z = 0;
            //原点
            user_coord.jointPara[0].jointRadian[0] = -0.000172 / 180 * M_PI;
            user_coord.jointPara[0].jointRadian[1] = -7.291862 / 180 * M_PI;
            user_coord.jointPara[0].jointRadian[2] = -75.694718 / 180 * M_PI;
            user_coord.jointPara[0].jointRadian[3] = 21.596727 / 180 * M_PI;
            user_coord.jointPara[0].jointRadian[4] = -89.999982 / 180 * M_PI;
            user_coord.jointPara[0].jointRadian[5] = -0.00458 / 180 * M_PI;
            //x轴正半轴上一点
            user_coord.jointPara[1].jointRadian[0] = 11.116932 / 180 * M_PI;
            user_coord.jointPara[1].jointRadian[1] = -0.858816 / 180 * M_PI;
            user_coord.jointPara[1].jointRadian[2] = -64.008663 / 180 * M_PI;
            user_coord.jointPara[1].jointRadian[3] = 26.850182 / 180 * M_PI;
            user_coord.jointPara[1].jointRadian[4] = -90.000064 / 180 * M_PI;
            user_coord.jointPara[1].jointRadian[5] = 11.116645 / 180 * M_PI;
            //x、y轴平面的第一象限上任意一点
            user_coord.jointPara[2].jointRadian[0] = 9.012562 / 180 * M_PI;
            user_coord.jointPara[2].jointRadian[1] = 13.378931 / 180 * M_PI;
            user_coord.jointPara[2].jointRadian[2] = -47.871256 / 180 * M_PI;
            user_coord.jointPara[2].jointRadian[3] = 28.749813 / 180 * M_PI;
            user_coord.jointPara[2].jointRadian[4] = -90.000064 / 180 * M_PI;
            user_coord.jointPara[2].jointRadian[5] = 9.012275 / 180 * M_PI;
            MoveRotateAxis rotate_axis = new MoveRotateAxis();
            IntPtr pt_rotate_axis = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(MoveRotateAxis)));
            rotate_axis = (MoveRotateAxis)Marshal.PtrToStructure(pt_rotate_axis, typeof(MoveRotateAxis));
            rotate_axis.rotateAxis[0] = 1; //x
            rotate_axis.rotateAxis[1] = 0; //y
```

```
rotate_axis.rotateAxis[2] = 0; //z沿z轴旋转
double[] wp1 = { 0, 0, 0, 0, 0, 0 };
wp1[0] = 0 * M_PI / 180;
wp1[1] = -7.29 * M_PI / 180;
wp1[2] = -75.69 * M_PI / 180;
wp1[3] = 21.59 * M_PI / 180;
wp1[4] = -90 * M_PI / 180;
wp1[5] = 0 * M_PI / 180;
rs_move_joint(rshd, wp1, true);
rs_move_rotate(rshd, ref user_coord, ref rotate_axis, 0.5, true);
rs_move_joint(rshd, wp1, true);
```

输出

示例 3

示例讲解：工具坐标系（tcp2）绕 User 坐标系（p1）的 Z 轴旋转 30°。

```
CoordCalibrate user_coord = new CoordCalibrate();
IntPtr pt_user_coord = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(CoordCalibrate)));
user_coord = (CoordCalibrate)Marshal.PtrToStructure(pt_user_coord, typeof(CoordCalibrate));
//检查用户坐标系参数设置是否合理
user_coord.coordType = WorldCoordinate;
//坐标系标定方法xoxy
user_coord.methods = Origin_AnyPointOnPositiveXAxis_AnyPointOnFirstQuadrantOfXOYPlane;
//为了简化，使用法兰盘中心为工具端
user_coord.toolDesc.cartPos.x = 0;
user_coord.toolDesc.cartPos.y = 0;
user_coord.toolDesc.cartPos.z = 0;
user_coord.toolDesc.orientation.w = 1;
user_coord.toolDesc.orientation.x = 0;
user_coord.toolDesc.orientation.y = 0;
user_coord.toolDesc.orientation.z = 0;
//原点
user_coord.jointPara[0].jointRadian[0] = -0.000172 / 180 * M_PI;
user_coord.jointPara[0].jointRadian[1] = -7.291862 / 180 * M_PI;
user_coord.jointPara[0].jointRadian[2] = -75.694718 / 180 * M_PI;
user_coord.jointPara[0].jointRadian[3] = 21.596727 / 180 * M_PI;
user_coord.jointPara[0].jointRadian[4] = -89.999982 / 180 * M_PI;
user_coord.jointPara[0].jointRadian[5] = -0.00458 / 180 * M_PI;
//x轴正半轴上一点
user_coord.jointPara[1].jointRadian[0] = 11.116932 / 180 * M_PI;
user_coord.jointPara[1].jointRadian[1] = -0.858816 / 180 * M_PI;
user_coord.jointPara[1].jointRadian[2] = -64.008663 / 180 * M_PI;
user_coord.jointPara[1].jointRadian[3] = 26.850182 / 180 * M_PI;
```

```
user_coord.jointPara[1].jointRadian[4] = -90.000064 / 180 * M_PI;
user_coord.jointPara[1].jointRadian[5] = 11.116645 / 180 * M_PI;
//x、y轴平面的第一象限上任意一点
user_coord.jointPara[2].jointRadian[0] = 9.012562 / 180 * M_PI;
user_coord.jointPara[2].jointRadian[1] = 13.378931 / 180 * M_PI;
user_coord.jointPara[2].jointRadian[2] = -47.871256 / 180 * M_PI;
user_coord.jointPara[2].jointRadian[3] = 28.749813 / 180 * M_PI;
user_coord.jointPara[2].jointRadian[4] = -90.000064 / 180 * M_PI;
user_coord.jointPara[2].jointRadian[5] = 9.012275 / 180 * M_PI;
MoveRotateAxis rotate_axis = new MoveRotateAxis();
IntPtr pt_rotate_axis = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(MoveRotateAxis)));
rotate_axis = (MoveRotateAxis)Marshal.PtrToStructure(pt_rotate_axis, typeof(MoveRotateAxis));
rotate_axis.rotateAxis[0] = 1; //x
rotate_axis.rotateAxis[1] = 0; //y
rotate_axis.rotateAxis[2] = 0; //z沿z轴旋转
//工具参数
ToolInEndDesc tool=new ToolInEndDesc();
tool.cartPos.x = 0;
tool.cartPos.x = 0;
tool.cartPos.x = 0.1;
tool.orientation.w = 1;
tool.orientation.x = 0;
tool.orientation.y = 0;
tool.orientation.z = 0;
rs_set_tool_kinematics_param(rshd, ref tool);
double[] wp1 = { 0, 0, 0, 0, 0, 0 };
wp1[0] = 0 * M_PI / 180;
wp1[1] = -7.29 * M_PI / 180;
wp1[2] = -75.69 * M_PI / 180;
wp1[3] = 21.59 * M_PI / 180;
wp1[4] = -90 * M_PI / 180;
wp1[5] = 0 * M_PI / 180;
rs_move_joint(rshd, wp1, true);
rs_move_rotate(rshd, ref user_coord, ref rotate_axis, 0.5, true);
rs_move_joint(rshd, wp1, true);
```

输出	
示例 4	示例讲解：绕工具坐标系（tcp2）的 X 轴旋转 10°。

41.轨迹运动 rs_move_track

函数功能	轨迹运动
参数描述	1、rshd 上下文控制句柄 2、sub_move_mode 2：Arc_Cir 3：MoveP 3、IsBolck ： 是否阻塞。
	IsBolck==true 代表阻塞，机械臂运动直到到达目标位置或者出现故障后返回。 IsBolck==false 代表非阻塞，立即返回，运动指令发送成功就返回，函数返回后机械臂开始运动。
返回值	调用成功返回 RSERR_SUCC;错误返回错误号
示例 1	示例讲解：圆运动。

	double[] wp1 = { -0.000003, -0.127267, -1.321122, 0.376941, -1.570796, -0.000008 };
	double[] wp2 = { -0.000003, -0.230606, -1.402912, 0.398490, -1.570796, -0.000008 };
	double[] wp3 = { -0.083254, -0.250952, -1.417539, 0.404209, -1.570796, -0.083259 };
	rs_move_joint(rshd, wp1, true);
	rs_remove_all_waypoint(rshd);
	rs_add_waypoint(rshd, wp1);
	rs_add_waypoint(rshd, wp2);
	rs_add_waypoint(rshd, wp3);
	rs_set_circular_loop_times(rshd, 2);
	rs_move_track(rshd, 2, true);

输出	
示例 2	示例讲解：圆弧运动。

	double[] wp1 = { -0.000003, -0.127267, -1.321122, 0.376941, -1.570796, -0.000008 };
	double[] wp2 = { -0.000003, -0.230606, -1.402912, 0.398490, -1.570796, -0.000008 };
	double[] wp3 = { -0.083254, -0.250952, -1.417539, 0.404209, -1.570796, -0.083259 };
	rs_move_joint(rshd, wp1, true);
	rs_remove_all_waypoint(rshd);
	rs_add_waypoint(rshd, wp1);
	rs_add_waypoint(rshd, wp2);
	rs_add_waypoint(rshd, wp3);
	rs_set_circular_loop_times(rshd, 0);
	rs_move_track(rshd, 2, true);

输出	

示例 3	示例讲解：MoveP

	double[] wp1 = { -0.000003, -0.127267, -1.321122, 0.376941, -1.570796, -0.000008 };
	double[] wp2 = { -0.000003, -0.230606, -1.402912, 0.398490, -1.570796, -0.000008 };
	double[] wp3 = { -0.083254, -0.250952, -1.417539, 0.404209, -1.570796, -0.083259 };
	rs_move_joint(rshd, wp1, true);
	rs_remove_all_waypoint(rshd);
	rs_add_waypoint(rshd, wp1);
	rs_add_waypoint(rshd, wp2);
	rs_add_waypoint(rshd, wp3);
	rs_set_blend_radius(rshd, 0.01);
	rs_move_track(rshd, 3, true);

输出	

42.**保持当前姿态通过相对位移直线运动至目标位置

43.**保持当前姿态通过相对位移关节运动至目标位置

44. 直线方式运动至给定位置 rs_move_line_to

int rs_move_line_to(UInt16 rshd,ref Pos target, ref ToolInEndDesc tool, bool isblock);	
函数功能	保持当前姿态通过直线运动的方式运动到目标位置
参数描述	参数 1：工具坐标系在 Base 坐标系下的位置参数 参数 2：工具坐标系 参数：IsBolck 是否阻塞。 IsBolck==true 代表阻塞，机械臂运动直到到达目标位置或者出现故障后返回。 IsBolck==false 代表非阻塞，立即返回，运动指令发送成功就返回，函数返回后机械臂开始运动。
返回值	调用成功返回 RSERR_SUCC;错误返回错误号
示例 1	示例讲解：flange_center 在 Base 下的某个位置。

	Pos target=new Pos();
	target.x = -0.4;
	target.y = -0.12;

	target.z = 0.4;
	ToolInEndDesc tool=new ToolInEndDesc();
	tool.cartPos.x = 0;
	tool.cartPos.y = 0;
	tool.cartPos.z = 0;
	tool.orientation.w = 1;
	tool.orientation.x = 0;
	tool.orientation.y = 0;
	tool.orientation.z = 0;
	double[] wp1 = { -0.000003, -0.127267, -1.321122, 0.376941, -1.570796, -0.000008 };
	rs_move_joint(rshd, wp1, true);
	rs_move_line_to(rshd, ref target, ref tool,true);

输出	

45.轴动方式运动至给定位置 rs_move_joint_to

int rs_move_joint_to(UInt16 rshd,ref Pos target, ref ToolInEndDesc tool, bool isblock);	
函数功能	保持当前姿态通过轴动运动的方式运动到目标位置
参数描述	参数 1：工具坐标系在 Base 坐标系下的位置参数 参数 2：工具坐标系 参数：IsBolck 是否阻塞。 IsBolck==true 代表阻塞，机械臂运动直到到达目标位置或者出现故障后返回。 IsBolck==false 代表非阻塞，立即返回，运动指令发送成功就返回，函数返回后机械臂开始运动。
返回值	调用成功返回 RSERR_SUCC;错误返回错误号

示例	示例讲解：轴动运动至 flange_center 在 Base 下的位置。
	Pos target=new Pos();
	target.x = -0.4;
	target.y = -0.12;
	target.z = 0.4;
	ToolInEndDesc tool=new ToolInEndDesc();
	tool.cartPos.x = 0;
	tool.cartPos.y = 0;
	tool.cartPos.z = 0;
	tool.orientation.w = 1;
	tool.orientation.x = 0;
	tool.orientation.y = 0;

	tool.orientation.z = 0;
	double[] wp1 = { -0.000003, -0.127267, -1.321122, 0.376941, -1.570796, -0.000008 };
	rs_move_joint(rshd, wp1, true);
	rs_move_joint_to(rshd, ref target, ref tool,true);

输出	

46.示教运动开始 rs_teach_move_start

int rs_teach_move_start(UInt16 rshd, teach_mode mode, bool dir);	
函数功能	开始示教

参数描述	参数 1：mode 示教模式。
	示教关节:JOINT1, JOINT2, JOINT3, JOINT4, JOINT5, JOINT6,
	位置示教:MOV_X, MOV_Y, MOV_Z
	姿态示教:ROT_X, ROT_Y, ROT_Z;
	enum teach_mode
	{
	NO_TEACH = 0,
	JOINT1,
	JOINT2,
	JOINT3,
	JOINT4,
	JOINT5,
	JOINT6,
	MOV_X,
	MOV_Y,
	MOV_Z,
	ROT_X,
	ROT_Y,
	ROT_Z
	};
	参数 2：direction 方向，方向正方向 true 反方向 false

返回值	调用成功返回 RSERR_SUCC;错误返回错误号

示例 1	示例讲解：关节示教
	teach_mode j5 = teach_mode.JOINT5;
	rs_teach_move_start(rshd, j5, true);
	Thread.Sleep(1000);

	rs_teach_move_stop(rshd);
	rs_teach_move_start(rshd, j5, false);
	Thread.Sleep(1000);
	rs_teach_move_stop(rshd);

输出	

47.示教停止 rs_teach_move_stop

int rs_teach_move_stop(UInt16 rshd);	
函数功能	示教停止
参数描述	
返回值	调用成功返回 RSERR_SUCC;错误返回错误号

示例	//结束示教
	rs_teach_move_stop(rshd)

输出	

48.机械臂运动快速停止 rs_move_fast_stop

int rs_move_fast_stop (UInt16 rshd);	
函数功能	机械臂快速停止
参数描述	注意：robotMoveFastStop 需要在与 move 不同的线程中调用。且 robotMoveFastStop 调用后需要停止 move 线程，否则会接着执行下一个 move 指令。
返回值	调用成功返回 RSERR_SUCC;错误返回错误号
示例 1	示例讲解：机械臂快速停止。 /*机械臂快速停止*/ rs_move_fast_stop (rshd);
输出	

49.机械臂运动停止 rs_move_stop

int rs_move_stop (UInt16 rshd);	
函数功能	机械臂快速停止
参数描述	
返回值	调用成功返回 RSERR_SUCC;错误返回错误号
示例 1	示例讲解：机械臂快速停止。 /*机械臂快速停止*/ rs_move_stop (rshd);
输出	

50.机械臂运动暂停 rs_move_pause

int rs_move_pause (UInt16 rshd);	
函数功能	机械臂快速停止
参数描述	
返回值	调用成功返回 RSERR_SUCC;错误返回错误号
示例 1	示例讲解：机械臂快速停止。 /*机械臂快速停止*/ rs_move_pause (rshd);
输出	

51.机械臂运动暂停恢复 rs_move_continue

int rs_move_continue (UInt16 rshd);	
函数功能	机械臂快速停止

参数描述	
返回值	调用成功返回 RSERR_SUCC;错误返回错误号

示例 1	示例讲解：机械臂快速停止。
	`/*机械臂快速停止*/`
	`rs_move_continue (rshd);`

输出	

离线轨迹运动

52.**添加离线轨迹路点

53.**清空离线轨迹路点

54.**启动离线轨迹运行

55.**停止离线轨迹运行

TCP 转 CAN 透传

56.进入 tcp 转 can 透传模式 rs_enter_tcp2canbus_mode

`int rs_enter_tcp2canbus_mode(UInt16 rshd);`	
函数功能	进入 can 透传模式
参数描述	
返回值	调用成功返回 RSERR_SUCC;错误返回错误号
示例	`rs_enter_tcp2canbus_mode(rshd);`
输出	

57.退出 tcp 转 can 透传模式 rs_leave_tcp2canbus_mode

int rs_leave_tcp2canbus_mode(UInt16 rshd);	
函数功能	退出 can 透传模式
参数描述	
返回值	调用成功返回 RSERR_SUCC;错误返回错误号
示例	rs_leave_tcp2canbus_mode(rshd);
输出	

58.发送坐标数据到关节 can 总线 rs_set_waypoint_to_canbus

int rs_set_waypoint_to_canbus(UInt16 rshd, double[] joint_radia);	
函数功能	通过透传将关节角度信息发送至驱动器
参数描述	joint_radia：六个关节角，单位 rad
返回值	调用成功返回 RSERR_SUCC;错误返回错误号

示例	double[] wp1 = { -0.000003, -0.127267, -1.321122, 0.376941, -1.570796, -0.000008 };
	rs_move_joint(rshd, wp1, true);
	rs_enter_tcp2canbus_mode(rshd);
	for (int i = 0; i < 1000; i++)
	{
	wp1[4] = wp1[4] + 0.0001;
	rs_set_waypoint_to_canbus(rshd, wp1);
	Thread.Sleep(5);
	}
	rs_leave_tcp2canbus_mode(rshd);
	wp1[4] = -1.570796;
	rs_move_joint(rshd, wp1, true);

输出	

工具接口

59.正解 rs_forward_kin

int rs_forward_kin(UInt16 rshd, double[] joint_radia, ref wayPoint_S waypoint);	
函数功能	此函数为正解函数，已知关节角求对应位置的位置和姿态
参数描述	joint_radia:六个关节的关节角，输入参数，单位 rad。 waypoint：输出参数。
返回值	调用成功返回 RSERR_SUCC;错误返回错误号
示例	/*求正解 FK*/

```
            wayPoint_S waypoint = new wayPoint_S();
            //正解测试
            double[] temp_joint = {-4.358721 / 180 * M_PI,
                                   -6.4477722 / 180 * M_PI,
                                   -68.298347 / 180 * M_PI,
                                   19.191783 / 180 * M_PI,
                                   -81.194208 / 180 * M_PI,
                                   -4.366669 / 180 * M_PI
                                  };
            //正解
            rs_forward_kin(rshd, temp_joint, ref waypoint);
            //打印路点信息
            PrintWaypoint(waypoint);
```

输出	login succ ---------- pos.x=-0.410589723123057 y=-0.104988613726078 z=0.605506377396694 ori.w=0.108786228080563 x=0.7071546150522 y=-0.698578947087587 z=-0.00923913829190883 joint1=-4.358721 joint2=-6.4477722 joint3=-68.298347 joint4=19.191783 joint5=-81.194208 joint6=-4.366669 ---------- login out

60.逆解 rs_inverse_kin

int rs_inverse_kin(UInt16 rshd, double[] joint_radia, ref Pos pos, ref Ori ori, ref wayPoint_S waypoint);	
函数功能	此函数为机械臂逆解函数，根据位置信息(x,y,z)和对应位置的参考姿态(w,x,y,z)得到对应位置的关节角信息
参数描述	joint_radia:参考点六个关节的关节角，输入参数，单位 rad。 pos：目标路点的位置，输入参数，单位 m。 ori：目标路点的参考姿态，输入参数，单位四元数。 waypoint：输出参数。
返回值	调用成功返回 RSERR_SUCC;错误返回错误号

示例	/*求逆解 IK*/
	wayPoint_S waypoint = new wayPoint_S();
	Pos pos = new Pos();
	Ori ori = new Ori();
	//逆解测试
	pos.x = -0.484595;
	pos.y = 0.030930;
	pos.z = 0.341558;
	
	//参考当前机械臂姿态
	rs_get_current_waypoint(rshd, ref waypoint);
	ori = waypoint.orientation;
	
	//逆解

	rs_inverse_kin(rshd, waypoint.jointpos, ref pos, ref ori, ref waypoint); PrintWaypoint(waypoint);
输出	login succ --- pos.x=-0.484595 y=0.03093 z=0.341558 ori.w=2.68754845708537E-06 x=0.707108548946948 y=-0.707105013412322 z=-2.4656075235861E-06 joint1=-18.1423398499353 joint2=-0.226547669035047 joint3=-101.101661214304 joint4=-10.8751135452693 joint5=-89.9998528925999 joint6=-18.142626328389 --- login out

61.**工具标定

62.检查用户坐标系是否合理 rs_check_user_coord

int rs_check_user_coord(UInt16 rshd, ref CoordCalibrate user_coord);	
函数功能	检查提供的参数是否标定出一个坐标系
参数描述	user_coord:坐标系
返回值	调用成功返回 RSERR_SUCC;错误返回错误号

示例	CoordCalibrate user_coord = new CoordCalibrate();
	IntPtr pt_user_coord = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(CoordCalibrate)));
	user_coord = (CoordCalibrate)Marshal.PtrToStructure(pt_user_coord, typeof(CoordCalibrate));
	//坐标系类型
	user_coord.coordType = WorldCoordinate;
	//坐标系标定方法xoxy
	user_coord.methods = Origin_AnyPointOnPositiveXAxis_AnyPointOnFirstQuadrantOfXOYPlane;
	//为了简化，使用法兰盘中心为工具端
	user_coord.toolDesc.cartPos.x = 0;
	user_coord.toolDesc.cartPos.y = 0;
	user_coord.toolDesc.cartPos.z = 0;
	user_coord.toolDesc.orientation.w = 1;
	user_coord.toolDesc.orientation.x = 0;
	user_coord.toolDesc.orientation.y = 0;
	user_coord.toolDesc.orientation.z = 0;

```
//原点
user_coord.jointPara[0].jointRadian[0] = -0.000172 / 180 * M_PI;
user_coord.jointPara[0].jointRadian[1] = -7.291862 / 180 * M_PI;
user_coord.jointPara[0].jointRadian[2] = -75.694718 / 180 * M_PI;
user_coord.jointPara[0].jointRadian[3] = 21.596727 / 180 * M_PI;
user_coord.jointPara[0].jointRadian[4] = -89.999982 / 180 * M_PI;
user_coord.jointPara[0].jointRadian[5] = -0.00458 / 180 * M_PI;
//x轴正半轴上一点
user_coord.jointPara[1].jointRadian[0] = 11.116932 / 180 * M_PI;
user_coord.jointPara[1].jointRadian[1] = -0.858816 / 180 * M_PI;
user_coord.jointPara[1].jointRadian[2] = -64.008663 / 180 * M_PI;
user_coord.jointPara[1].jointRadian[3] = 26.850182 / 180 * M_PI;
user_coord.jointPara[1].jointRadian[4] = -90.000064 / 180 * M_PI;
user_coord.jointPara[1].jointRadian[5] = 11.116645 / 180 * M_PI;
//x、y轴平面的第一象限上任意一点
user_coord.jointPara[2].jointRadian[0] = 9.012562 / 180 * M_PI;
user_coord.jointPara[2].jointRadian[1] = 13.378931 / 180 * M_PI;
user_coord.jointPara[2].jointRadian[2] = -47.871256 / 180 * M_PI;
user_coord.jointPara[2].jointRadian[3] = 28.749813 / 180 * M_PI;
user_coord.jointPara[2].jointRadian[4] = -90.000064 / 180 * M_PI;
user_coord.jointPara[2].jointRadian[5] = 9.012275 / 180 * M_PI;
//检查用户坐标系参数设置是否合理
if (RSERR_SUCC == rs_check_user_coord(rshd, ref user_coord))
    Console.Out.WriteLine("user coord check ok.");
else
    Console.Out.WriteLine("user coord check failed!");
```

输出

```
login succ
user coord check ok.
login out
```

63.**用户坐标系标定

64.Base 坐标系转 User 坐标系 rs_base_to_user

```
int rs_base_to_user(UInt16 rshd, ref Pos pos_onbase, ref Ori ori_onbase, ref CoordCalibrate user_coord, ref ToolInEndDesc tool_pos, ref Pos
```

pos_onuser, ref Ori ori_onuser);	
函数功能	flange_center 在 Base 下坐标转工具坐标系在 User 下的坐标。
参数描述	参数 1：pos_onbase，flange_center 在 Base 下的位置参数 参数 2：ori_onbase，flange_center 在 Base 下的姿态参数 参数 3：user_coord，参考坐标系 参数 4：tool_pos，工具参数 参数 5：pos_onuser，输出参数，工具在 User 下的位置参数 参数 5：ori_onuser ，输出参数，工具在 User 下的姿态参数
返回值	调用成功返回 RSERR_SUCC;错误返回错误号

示例 1	示例讲解：flange_center 在 Base 下坐标转 Tool(tcp1)在 Base 下坐标。
	CoordCalibrate user_coord = new CoordCalibrate();
	IntPtr pt_user_coord = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(CoordCalibrate)));
	user_coord = (CoordCalibrate)Marshal.PtrToStructure(pt_user_coord, typeof(CoordCalibrate));
	//坐标系类型
	user_coord.coordType = BaseCoordinate;
	//坐标系标定方法xoxy
	user_coord.methods = Origin_AnyPointOnPositiveXAxis_AnyPointOnFirstQuadrantOfXOYPlane;
	//为了简化，使用法兰盘中心为工具端
	user_coord.toolDesc.cartPos.x = 0;
	user_coord.toolDesc.cartPos.y = 0;
	user_coord.toolDesc.cartPos.z = 0;
	user_coord.toolDesc.orientation.w = 1;
	user_coord.toolDesc.orientation.x = 0;
	user_coord.toolDesc.orientation.y = 0;
	user_coord.toolDesc.orientation.z = 0;
	//原点
	user_coord.jointPara[0].jointRadian[0] = -0.000172 / 180 * M_PI;
	user_coord.jointPara[0].jointRadian[1] = -7.291862 / 180 * M_PI;
	user_coord.jointPara[0].jointRadian[2] = -75.694718 / 180 * M_PI;
	user_coord.jointPara[0].jointRadian[3] = 21.596727 / 180 * M_PI;
	user_coord.jointPara[0].jointRadian[4] = -89.999982 / 180 * M_PI;
	user_coord.jointPara[0].jointRadian[5] = -0.00458 / 180 * M_PI;
	//x轴正半轴上一点
	user_coord.jointPara[1].jointRadian[0] = 11.116932 / 180 * M_PI;
	user_coord.jointPara[1].jointRadian[1] = -0.858816 / 180 * M_PI;
	user_coord.jointPara[1].jointRadian[2] = -64.008663 / 180 * M_PI;
	user_coord.jointPara[1].jointRadian[3] = 26.850182 / 180 * M_PI;
	user_coord.jointPara[1].jointRadian[4] = -90.000064 / 180 * M_PI;
	user_coord.jointPara[1].jointRadian[5] = 11.116645 / 180 * M_PI;

```
            //x、y轴平面的第一象限上任意一点
            user_coord.jointPara[2].jointRadian[0] = 9.012562 / 180 * M_PI;
            user_coord.jointPara[2].jointRadian[1] = 13.378931 / 180 * M_PI;
            user_coord.jointPara[2].jointRadian[2] = -47.871256 / 180 * M_PI;
            user_coord.jointPara[2].jointRadian[3] = 28.749813 / 180 * M_PI;
            user_coord.jointPara[2].jointRadian[4] = -90.000064 / 180 * M_PI;
            user_coord.jointPara[2].jointRadian[5] = 9.012275 / 180 * M_PI;
            wayPoint_S waypoint = new wayPoint_S();
            Rpy rpy = new Rpy();
            //基座坐标系转用户坐标系
            Ori ori_onbase = new Ori();
            Pos pos_onbase = new Pos();
            ToolInEndDesc tool_pos = new ToolInEndDesc();
            Pos pos_onuser = new Pos();
            Ori ori_onuser = new Ori();
            //首先获取当前位置
            rs_get_current_waypoint(rshd, ref waypoint);
            //打印路点信息
            PrintWaypoint(waypoint);
            ori_onbase = waypoint.orientation;
            pos_onbase = waypoint.cartPos;
            tool_pos.cartPos.x = 0;
            tool_pos.cartPos.y = 0;
            tool_pos.cartPos.z = 0.1;
            tool_pos.orientation.w = 1;
            tool_pos.orientation.x = 0;
            tool_pos.orientation.y = 0;
            tool_pos.orientation.z = 0;
            //基座坐标系转用户坐标系
            rs_base_to_user(rshd, ref pos_onbase, ref ori_onbase, ref user_coord, ref tool_pos, ref pos_onuser, ref ori_onuser);
            rs_quaternion_to_rpy(rshd, ref ori_onuser, ref rpy);
            Console.WriteLine("onuser rpy.rx={0}\nrpy.ry={1}\nrpy.rz={2}\n",
                rpy.rx * 180 / M_PI, rpy.ry * 180 / M_PI, rpy.rz * 180 / M_PI);
            Console.WriteLine("onuser pos.x={0}\npos.y={1}\npos.z={2}\n",
            pos_onuser.x, pos_onuser.y, pos_onuser.z);
```

输出

```
login succ
--------------------------------------------------------------------------------
pos.x=-0.400319025929111 y=-0.121498828551575 z=0.547598446946785
ori.w=2.68754845708537E-06 x=0.707108548946948 y=-0.707105013412322 z=-2.4656075235861E-06
joint1=-0.000171887344619049 joint2=-7.29186214658142 joint3=-75.6947177030201
joint4=21.5967268914056 joint5=-89.9999820139107 joint6=-0.000458366234947395
--------------------------------------------------------------------------------
onuser rpy.rx=179.999572753906
rpy.ry=-1.79824910446769E-05
rpy.rz=-89.9997100830078

onuser pos.x=-0.40031975469534
pos.y=-0.121498859940585
pos.z=0.447598446949445

login out
```

示例 2

示例讲解：flange_center 在 Base 下坐标转 flange_center 在 User(p1)下坐标。

```
CoordCalibrate user_coord = new CoordCalibrate();
IntPtr pt_user_coord = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(CoordCalibrate)));
user_coord = (CoordCalibrate)Marshal.PtrToStructure(pt_user_coord, typeof(CoordCalibrate));
//坐标系类型
user_coord.coordType = WorldCoordinate;
//坐标系标定方法xoxy
user_coord.methods = Origin_AnyPointOnPositiveXAxis_AnyPointOnFirstQuadrantOfXOYPlane;
//为了简化，使用法兰盘中心为工具端
user_coord.toolDesc.cartPos.x = 0;
user_coord.toolDesc.cartPos.y = 0;
user_coord.toolDesc.cartPos.z = 0;
user_coord.toolDesc.orientation.w = 1;
user_coord.toolDesc.orientation.x = 0;
user_coord.toolDesc.orientation.y = 0;
user_coord.toolDesc.orientation.z = 0;
//原点
user_coord.jointPara[0].jointRadian[0] = -0.000172 / 180 * M_PI;
user_coord.jointPara[0].jointRadian[1] = -7.291862 / 180 * M_PI;
user_coord.jointPara[0].jointRadian[2] = -75.694718 / 180 * M_PI;
user_coord.jointPara[0].jointRadian[3] = 21.596727 / 180 * M_PI;
user_coord.jointPara[0].jointRadian[4] = -89.999982 / 180 * M_PI;
user_coord.jointPara[0].jointRadian[5] = -0.00458 / 180 * M_PI;
//x轴正半轴上一点
```

```
                user_coord.jointPara[1].jointRadian[0] = 11.116932 / 180 * M_PI;
                user_coord.jointPara[1].jointRadian[1] = -0.858816 / 180 * M_PI;
                user_coord.jointPara[1].jointRadian[2] = -64.008663 / 180 * M_PI;
                user_coord.jointPara[1].jointRadian[3] = 26.850182 / 180 * M_PI;
                user_coord.jointPara[1].jointRadian[4] = -90.000064 / 180 * M_PI;
                user_coord.jointPara[1].jointRadian[5] = 11.116645 / 180 * M_PI;
                //x、y轴平面的第一象限上任意一点
                user_coord.jointPara[2].jointRadian[0] = 9.012562 / 180 * M_PI;
                user_coord.jointPara[2].jointRadian[1] = 13.378931 / 180 * M_PI;
                user_coord.jointPara[2].jointRadian[2] = -47.871256 / 180 * M_PI;
                user_coord.jointPara[2].jointRadian[3] = 28.749813 / 180 * M_PI;
                user_coord.jointPara[2].jointRadian[4] = -90.000064 / 180 * M_PI;
                user_coord.jointPara[2].jointRadian[5] = 9.012275 / 180 * M_PI;
                wayPoint_S waypoint = new wayPoint_S();
                Rpy rpy = new Rpy();
                //基座坐标系转用户坐标系
                Ori ori_onbase = new Ori();
                Pos pos_onbase = new Pos();
                ToolInEndDesc tool_pos = new ToolInEndDesc();
                Pos pos_onuser = new Pos();
                Ori ori_onuser = new Ori();
                //首先获取当前位置
                rs_get_current_waypoint(rshd, ref waypoint);
                //打印路点信息
                PrintWaypoint(waypoint);
                ori_onbase = waypoint.orientation;
                pos_onbase = waypoint.cartPos;
                tool_pos.cartPos.x = 0;
                tool_pos.cartPos.y = 0;
                tool_pos.cartPos.z = 0;
                tool_pos.orientation.w = 1;
                tool_pos.orientation.x = 0;
                tool_pos.orientation.y = 0;
                tool_pos.orientation.z = 0;
                //基座坐标系转用户坐标系
                rs_base_to_user(rshd, ref pos_onbase, ref ori_onbase, ref user_coord, ref tool_pos, ref pos_onuser, ref ori_onuser);
                rs_quaternion_to_rpy(rshd, ref ori_onuser, ref rpy);
                Console.WriteLine("onuser rpy.rx={0}\nrpy.ry={1}\nrpy.rz={2}\n",
                    rpy.rx * 180 / M_PI, rpy.ry * 180 / M_PI, rpy.rz * 180 / M_PI);
                Console.WriteLine("onuser pos.x={0}\npos.y={1}\npos.z={2}\n",
                    pos_onuser.x, pos_onuser.y, pos_onuser.z);
```

输出

```
login succ
--------------------------------------------------------------------------------

pos.x=-0.400319025929111 y=-0.121498828551575 z=0.547598446946785
ori.w=2.68754845708537E-06 x=0.707108548946948 y=-0.707105013412322 z=-2.4656075235861E-06
joint1=-0.000171887344619049 joint2=-7.29186214658142 joint3=-75.6947177030201
joint4=21.5967268914056 joint5=-89.9999820139107 joint6=-0.000458366234947395
--------------------------------------------------------------------------------

onuser rpy.rx=-0.000166148151038215
rpy.ry=-23.181079864502
rpy.rz=5.00048081448767E-05

onuser pos.x=2.10465850303265E-09
pos.y=-1.43978029498726E-09
pos.z=-2.97338476062947E-09

login out
```

示例 3

示例讲解：flange_center 在 Base 下坐标转 Tool(tcp1)在 User(p1)下坐标。

```
CoordCalibrate user_coord = new CoordCalibrate();
IntPtr pt_user_coord = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(CoordCalibrate)));
user_coord = (CoordCalibrate)Marshal.PtrToStructure(pt_user_coord, typeof(CoordCalibrate));
//坐标系类型
user_coord.coordType = WorldCoordinate;
//坐标系标定方法xoxy
user_coord.methods = Origin_AnyPointOnPositiveXAxis_AnyPointOnFirstQuadrantOfXOYPlane;
//为了简化，使用法兰盘中心为工具端
user_coord.toolDesc.cartPos.x = 0;
user_coord.toolDesc.cartPos.y = 0;
user_coord.toolDesc.cartPos.z = 0;
user_coord.toolDesc.orientation.w = 1;
user_coord.toolDesc.orientation.x = 0;
user_coord.toolDesc.orientation.y = 0;
user_coord.toolDesc.orientation.z = 0;
//原点
user_coord.jointPara[0].jointRadian[0] = -0.000172 / 180 * M_PI;
user_coord.jointPara[0].jointRadian[1] = -7.291862 / 180 * M_PI;
user_coord.jointPara[0].jointRadian[2] = -75.694718 / 180 * M_PI;
user_coord.jointPara[0].jointRadian[3] = 21.596727 / 180 * M_PI;
user_coord.jointPara[0].jointRadian[4] = -89.999982 / 180 * M_PI;
user_coord.jointPara[0].jointRadian[5] = -0.00458 / 180 * M_PI;
//x轴正半轴上一点
```

```
                user_coord.jointPara[1].jointRadian[0] = 11.116932 / 180 * M_PI;
                user_coord.jointPara[1].jointRadian[1] = -0.858816 / 180 * M_PI;
                user_coord.jointPara[1].jointRadian[2] = -64.008663 / 180 * M_PI;
                user_coord.jointPara[1].jointRadian[3] = 26.850182 / 180 * M_PI;
                user_coord.jointPara[1].jointRadian[4] = -90.000064 / 180 * M_PI;
                user_coord.jointPara[1].jointRadian[5] = 11.116645 / 180 * M_PI;
                //x、y轴平面的第一象限上任意一点
                user_coord.jointPara[2].jointRadian[0] = 9.012562 / 180 * M_PI;
                user_coord.jointPara[2].jointRadian[1] = 13.378931 / 180 * M_PI;
                user_coord.jointPara[2].jointRadian[2] = -47.871256 / 180 * M_PI;
                user_coord.jointPara[2].jointRadian[3] = 28.749813 / 180 * M_PI;
                user_coord.jointPara[2].jointRadian[4] = -90.000064 / 180 * M_PI;
                user_coord.jointPara[2].jointRadian[5] = 9.012275 / 180 * M_PI;
                wayPoint_S waypoint = new wayPoint_S();
                Rpy rpy = new Rpy();
                //基座坐标系转用户坐标系
                Ori ori_onbase = new Ori();
                Pos pos_onbase = new Pos();
                ToolInEndDesc tool_pos = new ToolInEndDesc();
                Pos pos_onuser = new Pos();
                Ori ori_onuser = new Ori();
                //首先获取当前位置
                rs_get_current_waypoint(rshd, ref waypoint);
                //打印路点信息
                PrintWaypoint(waypoint);
                ori_onbase = waypoint.orientation;
                pos_onbase = waypoint.cartPos;
                tool_pos.cartPos.x = 0;
                tool_pos.cartPos.y = 0;
                tool_pos.cartPos.z = 0.1;
                tool_pos.orientation.w = 1;
                tool_pos.orientation.x = 0;
                tool_pos.orientation.y = 0;
                tool_pos.orientation.z = 0;
                //基座坐标系转用户坐标系
                rs_base_to_user(rshd, ref pos_onbase, ref ori_onbase, ref user_coord, ref tool_pos, ref pos_onuser, ref ori_onuser);
                rs_quaternion_to_rpy(rshd, ref ori_onuser, ref rpy);
                Console.WriteLine("onuser rpy.rx={0}\nrpy.ry={1}\nrpy.rz={2}\n",
                    rpy.rx * 180 / M_PI, rpy.ry * 180 / M_PI, rpy.rz * 180 / M_PI);
                Console.WriteLine("onuser pos.x={0}\npos.y={1}\npos.z={2}\n",
                    pos_onuser.x, pos_onuser.y, pos_onuser.z);
```

输出	login succ ---------- pos.x=-0.400319025929111 y=-0.121498828551575 z=0.547598446946785 ori.w=2.68754845708537E-06 x=0.707108548946948 y=-0.707105013412322 z=-2.4656075235861E-06 joint1=-0.000171887344619049 joint2=-7.29186214658142 joint3=-75.6947177030201 joint4=21.5967268914056 joint5=-89.9999820139107 joint6=-0.000458366234947395 ---------- onuser rpy.rx=-0.000166148151038215 rpy.ry=-23.181079864502 rpy.rz=5.00048081448767E-05 onuser pos.x=-0.0393638347085351 pos.y=2.54184722714434E-07 pos.z=0.0919265348232652 login out

65.F_B 坐标系转 T_B 坐标 rs_base_to_base_additional_tool

```
int rs_base_to_base_additional_tool(UInt16 rshd, ref Pos flange_center_pos_onbase, ref Ori flange_center_ori_onbase, ref ToolInEndDesc tool_pos, ref Pos tool_end_pos_onbase, ref Ori tool_end_ori_onbase);
```

函数功能	flange_center 在 Base 下坐标转工具坐标系在 Base 下的坐标。
参数描述	参数 1：flange_center_pos_onbase，flange_center 在 Base 下的位置参数 参数 2：flange_center_ori_onbase，flange_center 在 Base 下的姿态参数 参数 3：tool_pos，工具参数 参数 4：tool_end_pos_onbase，输出参数，工具在 Baser 下的位置参数 参数 5：tool_end_ori_onbase ，输出参数，工具在 Base 下的姿态参数
返回值	调用成功返回 RSERR_SUCC;错误返回错误号
示例	示例讲解：flange_center 在 Base 下坐标系转 Tool(tcp1)在 Base 下坐标。

```
    /*F_B 转 T_B*/
```

```
            Rpy rpy = new Rpy();
            Ori flange_center_ori_onbase = new Ori();
            Pos flange_center_pos_onbase = new Pos();
            Pos tool_end_pos_onbase = new Pos();
```

```
            Ori tool_end_ori_onbase = new Ori();
            ToolInEndDesc tool_pos = new ToolInEndDesc();
            flange_center_pos_onbase.x= -0.199632;
            flange_center_pos_onbase.y = -0.598316;
            flange_center_pos_onbase.z = 0.018606;
            rpy.rx= 179.977173 * M_PI / 180;
            rpy.ry = -0.027977 * M_PI / 180;
            rpy.rz = -92.123917 * M_PI / 180;
            rs_rpy_to_quaternion(rshd, ref rpy, ref flange_center_ori_onbase);
            tool_pos.cartPos.x = 0;
            tool_pos.cartPos.y = 0;
            tool_pos.cartPos.z = 0.1;
            tool_pos.orientation.w = 1;
            tool_pos.orientation.x = 0;
            tool_pos.orientation.y = 0;
            tool_pos.orientation.z = 0;
            //F_B坐标系转T_B坐标
            rs_base_to_base_additional_tool(rshd, ref flange_center_pos_onbase, ref flange_center_ori_onbase, ref tool_pos, ref
tool_end_pos_onbase, ref tool_end_ori_onbase);
            rs_quaternion_to_rpy(rshd, ref tool_end_ori_onbase, ref rpy);
            Console.WriteLine("posx:"+tool_end_pos_onbase.x);
            Console.WriteLine("posy:"+tool_end_pos_onbase.y);
            Console.WriteLine("posz:"+tool_end_pos_onbase.z);
            Console.WriteLine("orix:" + rpy.rx*180/M_PI);
            Console.WriteLine("oriy:" + rpy.ry * 180 / M_PI);
            Console.WriteLine("oriz:" + rpy.rz * 180 / M_PI);
```

输出

```
login succ
posx:-0.199673613285725
posy:-0.598363319351112
posz:-0.0813939801460653
orix:179.977172851562
oriy:-0.0279769971966743
oriz:-92.1239242553711
login out
```

66.User 坐标系转 Base 坐标系 rs_user_to_base

```
int rs_user_to_base(UInt16 rshd, ref Pos pos_onuser, ref Ori ori_onuser, ref CoordCalibrate user_coord, ref ToolInEndDesc tool_pos, ref Pos pos_onbase, ref Ori ori_onbase);
```

函数功能	工具坐标系在 User 下的坐标转 flange_center 在 Base 下坐标。
参数描述	参数 1：pos_onuser，工具在 User 下的位置参数 参数 2：ori_onuser，工具在 User 下的姿态参数 参数 3：user_coord，参考坐标系 参数 4：tool_pos，工具参数 参数 5：pos_onbase，输出参数，flange_center 在 Base 下的位置参数 参数 5：ori_onbase ，输出参数，flange_center 在 Base 下的姿态参数
返回值	调用成功返回 RSERR_SUCC;错误返回错误号
示例 1	示例讲解：tcp1 在 Base 下坐标转 flange_center 在 Base 下坐标。

```
CoordCalibrate user_coord = new CoordCalibrate();
IntPtr pt_user_coord = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(CoordCalibrate)));
user_coord = (CoordCalibrate)Marshal.PtrToStructure(pt_user_coord, typeof(CoordCalibrate));
//坐标系类型
user_coord.coordType = BaseCoordinate;
//坐标系标定方法xoxy
user_coord.methods = Origin_AnyPointOnPositiveXAxis_AnyPointOnFirstQuadrantOfXOYPlane;
//为了简化，使用法兰盘中心为工具端
user_coord.toolDesc.cartPos.x = 0;
user_coord.toolDesc.cartPos.y = 0;
user_coord.toolDesc.cartPos.z = 0;
user_coord.toolDesc.orientation.w = 1;
user_coord.toolDesc.orientation.x = 0;
user_coord.toolDesc.orientation.y = 0;
user_coord.toolDesc.orientation.z = 0;
//原点
user_coord.jointPara[0].jointRadian[0] = -0.000172 / 180 * M_PI;
user_coord.jointPara[0].jointRadian[1] = -7.291862 / 180 * M_PI;
user_coord.jointPara[0].jointRadian[2] = -75.694718 / 180 * M_PI;
user_coord.jointPara[0].jointRadian[3] = 21.596727 / 180 * M_PI;
user_coord.jointPara[0].jointRadian[4] = -89.999982 / 180 * M_PI;
user_coord.jointPara[0].jointRadian[5] = -0.00458 / 180 * M_PI;
//x轴正半轴上一点
user_coord.jointPara[1].jointRadian[0] = 11.116932 / 180 * M_PI;
```

```
            user_coord.jointPara[1].jointRadian[1] = -0.858816 / 180 * M_PI;
            user_coord.jointPara[1].jointRadian[2] = -64.008663 / 180 * M_PI;
            user_coord.jointPara[1].jointRadian[3] = 26.850182 / 180 * M_PI;
            user_coord.jointPara[1].jointRadian[4] = -90.000064 / 180 * M_PI;
            user_coord.jointPara[1].jointRadian[5] = 11.116645 / 180 * M_PI;
            //x、y轴平面的第一象限上任意一点
            user_coord.jointPara[2].jointRadian[0] = 9.012562 / 180 * M_PI;
            user_coord.jointPara[2].jointRadian[1] = 13.378931 / 180 * M_PI;
            user_coord.jointPara[2].jointRadian[2] = -47.871256 / 180 * M_PI;
            user_coord.jointPara[2].jointRadian[3] = 28.749813 / 180 * M_PI;
            user_coord.jointPara[2].jointRadian[4] = -90.000064 / 180 * M_PI;
            user_coord.jointPara[2].jointRadian[5] = 9.012275 / 180 * M_PI;
            Rpy rpy = new Rpy();
            //基座坐标系转用户坐标系
            Ori ori_onbase = new Ori();
            Pos pos_onbase = new Pos();
            ToolInEndDesc tool_pos = new ToolInEndDesc();
            Pos pos_onuser = new Pos();
            Ori ori_onuser = new Ori();
            pos_onuser.x = 0;
            pos_onuser.y = 0;
            pos_onuser.z = 0;
            rpy.rx = 180/180*M_PI;
            rpy.ry = 0 / 180 * M_PI;
            rpy.rz = -90 / 180 * M_PI;
            rs_rpy_to_quaternion(rshd, ref rpy, ref ori_onuser);
            tool_pos.cartPos.x = 0;
            tool_pos.cartPos.y = 0;
            tool_pos.cartPos.z = 0.1;
            tool_pos.orientation.w = 1;
            tool_pos.orientation.x = 0;
            tool_pos.orientation.y = 0;
            tool_pos.orientation.z = 0;
            //用户坐标系转基座坐标系
            rs_user_to_base(rshd, ref pos_onuser, ref ori_onuser, ref user_coord, ref tool_pos, ref pos_onbase, ref ori_onbase);
            rs_quaternion_to_rpy(rshd, ref ori_onbase, ref rpy);
            Console.WriteLine("onuser rpy.rx={0}\nrpy.ry={1}\nrpy.rz={2}\n",
                rpy.rx * 180 / M_PI, rpy.ry * 180 / M_PI, rpy.rz * 180 / M_PI);
            Console.WriteLine("onuser pos.x={0}\npos.y={1}\npos.z={2}\n",
                pos_onbase.x, pos_onbase.y, pos_onbase.z);
```

输出

```
login succ
onuser rpy.rx=-179.999984741211
rpy.ry=0
rpy.rz=0

onuser pos.x=0
pos.y=-8.74227800037288E-09
pos.z=0.0999999999999996

login out
```

示例 2

示例讲解：flange_center 在 p1 下的坐标转 flange_center 在 Base 下坐标。

```
            CoordCalibrate user_coord = new CoordCalibrate();
            IntPtr pt_user_coord = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(CoordCalibrate)));
            user_coord = (CoordCalibrate)Marshal.PtrToStructure(pt_user_coord, typeof(CoordCalibrate));
            //坐标系类型
            user_coord.coordType = WorldCoordinate;
            //坐标系标定方法xoxy
            user_coord.methods = Origin_AnyPointOnPositiveXAxis_AnyPointOnFirstQuadrantOfXOYPlane;
            //为了简化，使用法兰盘中心为工具端
            user_coord.toolDesc.cartPos.x = 0;
            user_coord.toolDesc.cartPos.y = 0;
            user_coord.toolDesc.cartPos.z = 0;
            user_coord.toolDesc.orientation.w = 1;
            user_coord.toolDesc.orientation.x = 0;
            user_coord.toolDesc.orientation.y = 0;
            user_coord.toolDesc.orientation.z = 0;
            //原点
            user_coord.jointPara[0].jointRadian[0] = -0.000172 / 180 * M_PI;
            user_coord.jointPara[0].jointRadian[1] = -7.291862 / 180 * M_PI;
            user_coord.jointPara[0].jointRadian[2] = -75.694718 / 180 * M_PI;
            user_coord.jointPara[0].jointRadian[3] = 21.596727 / 180 * M_PI;
            user_coord.jointPara[0].jointRadian[4] = -89.999982 / 180 * M_PI;
            user_coord.jointPara[0].jointRadian[5] = -0.00458 / 180 * M_PI;
            //x轴正半轴上一点
            user_coord.jointPara[1].jointRadian[0] = 11.116932 / 180 * M_PI;
            user_coord.jointPara[1].jointRadian[1] = -0.858816 / 180 * M_PI;
            user_coord.jointPara[1].jointRadian[2] = -64.008663 / 180 * M_PI;
            user_coord.jointPara[1].jointRadian[3] = 26.850182 / 180 * M_PI;
            user_coord.jointPara[1].jointRadian[4] = -90.000064 / 180 * M_PI;
            user_coord.jointPara[1].jointRadian[5] = 11.116645 / 180 * M_PI;
```

```
            //x、y轴平面的第一象限上任意一点
            user_coord.jointPara[2].jointRadian[0] = 9.012562 / 180 * M_PI;
            user_coord.jointPara[2].jointRadian[1] = 13.378931 / 180 * M_PI;
            user_coord.jointPara[2].jointRadian[2] = -47.871256 / 180 * M_PI;
            user_coord.jointPara[2].jointRadian[3] = 28.749813 / 180 * M_PI;
            user_coord.jointPara[2].jointRadian[4] = -90.000064 / 180 * M_PI;
            user_coord.jointPara[2].jointRadian[5] = 9.012275 / 180 * M_PI;
            Rpy rpy = new Rpy();
            //基座坐标系转用户坐标系
            Ori ori_onbase = new Ori();
            Pos pos_onbase = new Pos();
            ToolInEndDesc tool_pos = new ToolInEndDesc();
            Pos pos_onuser = new Pos();
            Ori ori_onuser = new Ori();
            pos_onuser.x = 0;
            pos_onuser.y = 0;
            pos_onuser.z = 0;
            rpy.rx = 180/180*M_PI;
            rpy.ry = 0 / 180 * M_PI;
            rpy.rz = -90 / 180 * M_PI;
            rs_rpy_to_quaternion(rshd, ref rpy, ref ori_onuser);
            tool_pos.cartPos.x = 0;
            tool_pos.cartPos.y = 0;
            tool_pos.cartPos.z = 0;
            tool_pos.orientation.w = 1;
            tool_pos.orientation.x = 0;
            tool_pos.orientation.y = 0;
            tool_pos.orientation.z = 0;
            //用户坐标系转基座坐标系
            rs_user_to_base(rshd, ref pos_onuser, ref ori_onuser, ref user_coord, ref tool_pos, ref pos_onbase, ref ori_onbase);
            rs_quaternion_to_rpy(rshd, ref ori_onbase, ref rpy);
            Console.WriteLine("onuser rpy.rx={0}\nrpy.ry={1}\nrpy.rz={2}\n",
                rpy.rx * 180 / M_PI, rpy.ry * 180 / M_PI, rpy.rz * 180 / M_PI);
            Console.WriteLine("onuser pos.x={0}\npos.y={1}\npos.z={2}\n",
                pos_onbase.x, pos_onbase.y, pos_onbase.z);
```

输出

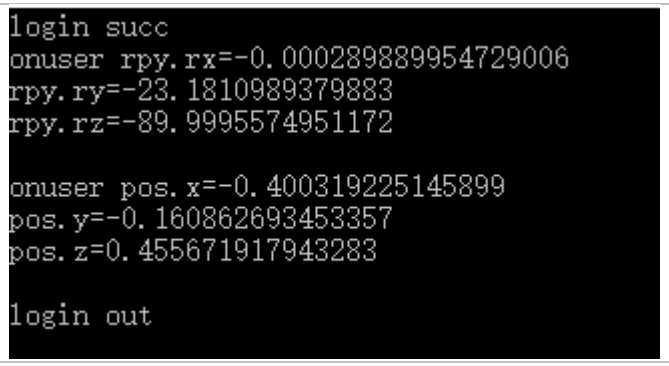
```
login succ
onuser rpy.rx=-0.000289889954729006
rpy.ry=-23.1810989379883
rpy.rz=-89.9995574951172

onuser pos.x=-0.400319027368912
pos.y=-0.121498827787284
pos.z=0.547598443384987

login out
```

示例 3

示例讲解：tcp1 在 p1 下坐标转 flange_center 在 Base 下坐标。

```
CoordCalibrate user_coord = new CoordCalibrate();
IntPtr pt_user_coord = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(CoordCalibrate)));
user_coord = (CoordCalibrate)Marshal.PtrToStructure(pt_user_coord, typeof(CoordCalibrate));
//坐标系类型
user_coord.coordType = WorldCoordinate;
//坐标系标定方法xoxy
user_coord.methods = Origin_AnyPointOnPositiveXAxis_AnyPointOnFirstQuadrantOfXOYPlane;
//为了简化，使用法兰盘中心为工具端
user_coord.toolDesc.cartPos.x = 0;
user_coord.toolDesc.cartPos.y = 0;
user_coord.toolDesc.cartPos.z = 0;
user_coord.toolDesc.orientation.w = 1;
user_coord.toolDesc.orientation.x = 0;
user_coord.toolDesc.orientation.y = 0;
user_coord.toolDesc.orientation.z = 0;
//原点
user_coord.jointPara[0].jointRadian[0] = -0.000172 / 180 * M_PI;
user_coord.jointPara[0].jointRadian[1] = -7.291862 / 180 * M_PI;
user_coord.jointPara[0].jointRadian[2] = -75.694718 / 180 * M_PI;
user_coord.jointPara[0].jointRadian[3] = 21.596727 / 180 * M_PI;
user_coord.jointPara[0].jointRadian[4] = -89.999982 / 180 * M_PI;
user_coord.jointPara[0].jointRadian[5] = -0.00458 / 180 * M_PI;
//x轴正半轴上一点
user_coord.jointPara[1].jointRadian[0] = 11.116932 / 180 * M_PI;
user_coord.jointPara[1].jointRadian[1] = -0.858816 / 180 * M_PI;
user_coord.jointPara[1].jointRadian[2] = -64.008663 / 180 * M_PI;
user_coord.jointPara[1].jointRadian[3] = 26.850182 / 180 * M_PI;
user_coord.jointPara[1].jointRadian[4] = -90.000064 / 180 * M_PI;
```

```
            user_coord.jointPara[1].jointRadian[5] = 11.116645 / 180 * M_PI;
            //x、y轴平面的第一象限上任意一点
            user_coord.jointPara[2].jointRadian[0] = 9.012562 / 180 * M_PI;
            user_coord.jointPara[2].jointRadian[1] = 13.378931 / 180 * M_PI;
            user_coord.jointPara[2].jointRadian[2] = -47.871256 / 180 * M_PI;
            user_coord.jointPara[2].jointRadian[3] = 28.749813 / 180 * M_PI;
            user_coord.jointPara[2].jointRadian[4] = -90.000064 / 180 * M_PI;
            user_coord.jointPara[2].jointRadian[5] = 9.012275 / 180 * M_PI;
            Rpy rpy = new Rpy();
            //基座坐标系转用户坐标系
            Ori ori_onbase = new Ori();
            Pos pos_onbase = new Pos();
            ToolInEndDesc tool_pos = new ToolInEndDesc();
            Pos pos_onuser = new Pos();
            Ori ori_onuser = new Ori();
            pos_onuser.x = 0;
            pos_onuser.y = 0;
            pos_onuser.z = 0;
            rpy.rx = 180/180*M_PI;
            rpy.ry = 0 / 180 * M_PI;
            rpy.rz = -90 / 180 * M_PI;
            rs_rpy_to_quaternion(rshd, ref rpy, ref ori_onuser);
            tool_pos.cartPos.x = 0;
            tool_pos.cartPos.y = 0;
            tool_pos.cartPos.z = 0.1;
            tool_pos.orientation.w = 1;
            tool_pos.orientation.x = 0;
            tool_pos.orientation.y = 0;
            tool_pos.orientation.z = 0;
            //用户坐标系转基座坐标系
            rs_user_to_base(rshd, ref pos_onuser, ref ori_onuser, ref user_coord, ref tool_pos, ref pos_onbase, ref ori_onbase);
            rs_quaternion_to_rpy(rshd, ref ori_onbase, ref rpy);
            Console.WriteLine("onuser rpy.rx={0}\nrpy.ry={1}\nrpy.rz={2}\n",
                rpy.rx * 180 / M_PI, rpy.ry * 180 / M_PI, rpy.rz * 180 / M_PI);
            Console.WriteLine("onuser pos.x={0}\npos.y={1}\npos.z={2}\n",
                pos_onbase.x, pos_onbase.y, pos_onbase.z);
```

输出	

67.**User 坐标系下位置参数转 Base 坐标系

68.**flange 姿态转 Tool 姿态

69.**Tool 姿态转 flange 姿态

70.**根据位置获取目标路点信息

71.四元数转欧拉角 rs_rpy_to_quaternion

```
int rs_rpy_to_quaternion(UInt16 rshd, ref Rpy rpy, ref Ori ori);
```

函数功能	四元素转欧拉角
参数描述	ori：四元数，输入参数 rpy：欧拉角，输出参数，单位 rad
返回值	调用成功返回 RSERR_SUCC;错误返回错误号
示例	wayPoint_S waypoint = new wayPoint_S(); Rpy rpy = new Rpy(); Ori ori = new Ori();

	rs_get_current_waypoint(rshd, ref waypoint);
	ori = waypoint.orientation;
	//四元数转欧拉角
	ori = waypoint.orientation;
	rs_quaternion_to_rpy(rshd, ref ori, ref rpy);
	System.Console.Out.WriteLine("rpy.rx={0}\nrpy.ry={1}\nrpy.rz={2}\n",
	rpy.rx * 180 / M_PI, rpy.ry * 180 / M_PI, rpy.rz * 180 / M_PI);

输出	login succ rpy.rx=179.999572753906 rpy.ry=-1.79824910446769E-05 rpy.rz=-89.9997100830078 login out

72.欧拉角转四元数 rs_quaternion_to_rpy

int rs_quaternion_to_rpy(UInt16 rshd, ref Ori ori, ref Rpy rpy);	
函数功能	欧拉角转四元数
参数描述	rpy：欧拉角，输入参数，单位 rad ori：四元数，输出参数
返回值	调用成功返回 RSERR_SUCC;错误返回错误号

示例	Rpy rpy = new Rpy();
	Ori ori = new Ori();
	rpy.rx = 180 /180 * M_PI;
	rpy.rx = 0 / 180 * M_PI;
	rpy.rx = -90/ 180 * M_PI;
	//欧拉角转四元数
	rs_rpy_to_quaternion(rshd, ref rpy, ref ori);
	Console.Out.WriteLine("ori.w={0} x={1} y={2} z={3}",
	ori.w, ori.x, ori.y, ori.z);

输出	login succ ori.w=1 x=0 y=0 z=0 login out

73.**根据错误号返回错误信息

机械臂控制接口

74.**机械臂控制

75.**返回电源状态

76.**机械臂释放刹车

末端工具接口

77.设置无工具动力学参数 rs_set_none_tool_dynamics_param

int rs_set_none_tool_dynamics_param(UInt16 rshd);	
函数功能	设置无工具的动力学参数
参数描述	
返回值	调用成功返回 RSERR_SUCC;错误返回错误号
示例	/*设置无工具动力学参数*/ rs_set_none_tool_dynamics_param(rshd);
输出	

78.设置工具的动力学参数 rs_set_tool_dynamics_param

int rs_set_tool_dynamics_param(UInt16 rshd, ref ToolDynamicsParam tool);	
函数功能	设置工具的动力学参数
参数描述	tool：工具动力学参数
返回值	调用成功返回 RSERR_SUCC;错误返回错误号

示例	/*设置工具动力学参数*/
	ToolDynamicsParam tool = new ToolDynamicsParam();
	tool.payload = 1;
	tool.positionX = 0;
	tool.positionY = 0;
	tool.positionZ = 0.1;
	rs_set_tool_dynamics_param(rshd, ref tool);

输出	

79.**获取工具的动力学参数 rs_get_tool_dynamics_param

int rs_get_tool_dynamics_param(UInt16 rshd, ref ToolDynamicsParam tool);	
函数功能	获取工具的动力学参数
参数描述	tool：工具动力学参数，输出参数
返回值	调用成功返回 RSERR_SUCC;错误返回错误号
示例	/*获取工具动力学参数*/
输出	

80.设置无工具运动学参数 rs_set_none_tool_kinematics_param

int rs_set_none_tool_kinematics_param(UInt16 rshd);	
函数功能	设置无工具运动学参数
参数描述	
返回值	调用成功返回 RSERR_SUCC;错误返回错误号
示例	/*设置无工具运动学参数*/ rs_set_none_tool_kinematics_param(rshd);
输出	

81.设置工具的运动学参数 rs_set_tool_kinematics_param

int rs_set_tool_kinematics_param(UInt16 rshd, ref ToolInEndDesc tool);	
函数功能	设置工具的运动学参数
参数描述	tool：工具运动学参数
返回值	调用成功返回 RSERR_SUCC;错误返回错误号

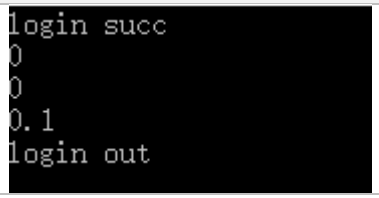
示例	/*设置工具运动学参数*/
	ToolInEndDesc tool = new ToolInEndDesc();
	tool.cartPos.x = 0;
	tool.cartPos.y = 0;
	tool.cartPos.z = 0.1;
	tool.orientation.w = 1;
	tool.orientation.x = 0;
	tool.orientation.y = 0;
	tool.orientation.z = 0;
	rs_set_tool_kinematics_param(rshd, ref tool);

输出	

82.获取工具的运动学参数 rs_get_tool_kinematics_param

int rs_get_tool_kinematics_param(UInt16 rshd, ref ToolInEndDesc tool);	
函数功能	获取工具的运动学参数
参数描述	tool：工具运动学参数，输出参数
返回值	调用成功返回 RSERR_SUCC;错误返回错误号

示例	/*获取工具运动学参数*/
	ToolInEndDesc tool = new ToolInEndDesc();
	tool.cartPos.x = 0;
	tool.cartPos.y = 0;
	tool.cartPos.z = 0.1;
	tool.orientation.w = 1;
	tool.orientation.x = 0;
	tool.orientation.y = 0;
	tool.orientation.z = 0;
	rs_set_tool_kinematics_param(rshd, ref tool);
	rs_get_tool_kinematics_param(rshd, ref tool);
	Console.WriteLine(tool.cartPos.x);
	Console.WriteLine(tool.cartPos.y);
	Console.WriteLine(tool.cartPos.z);

输出	

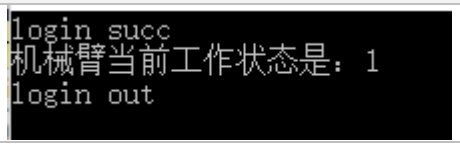
机械臂相关属性获取与设置

83.获取机械臂当前工作模式 rs_get_work_mode

int rs_get_work_mode(UInt16 rshd, ref int state);	
函数功能	获取机械臂当前工作模式

参数描述	传出参数，表示机械臂当前模式。0：仿真模式；1：真实模式。
返回值	调用成功返回 RSERR_SUCC;错误返回错误号

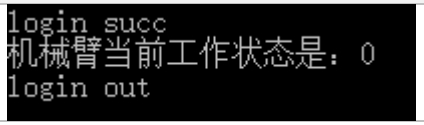
示例	*//获取机械臂当前工作模式*
	`int state=0;`
	`rs_get_work_mode(rshd, ref state);`
	`Console.WriteLine("机械臂当前工作状态是："+state);`

输出	

84.设置机械臂当前工作模式 rs_set_work_mode

`int rs_set_work_mode(UInt16 rshd, int state);`	
函数功能	设置当前机械臂模式　仿真或真实
参数描述	机械臂工作模式。0：仿真模式；1：真实模式。
返回值	调用成功返回 RSERR_SUCC;错误返回错误号

示例	*//设置机械臂当前工作模式*
	`int state=0;`
	`rs_set_work_mode(rshd,state);`
	`rs_get_work_mode(rshd, ref state);`
	`Console.WriteLine("机械臂当前工作状态是："+state);`

输出	

85.**获取重力分量 rs_get_gravity_component

int rs_get_gravity_component(UInt16 rshd, ref RobotGravityComponent gravity);	
函数功能	获取重力分量
参数描述	gravityComponent：重力分量，输出参数 需连接真实机器人
返回值	调用成功返回 RSERR_SUCC;错误返回错误号
示例	/*获取重力分量*/
输出	

86.**获取当前碰撞等级

87.设置机械臂碰撞等级 rs_set_collision_class

int rs_set_collision_class(UInt16 rshd, int grade);	
函数功能	设置碰撞等级
参数描述	碰撞等级：0~10
返回值	调用成功返回 RSERR_SUCC;错误返回错误号

示例	/*设置机械臂碰撞等级*/
	int grade = 2;
	rs_set_collision_class(rshd,grade);
	Console.Out.WriteLine("set robot collision ret is :" + grade);

输出	login succ set robot collision ret is :2 login out

88.获取设备信息 rs_get_device_info

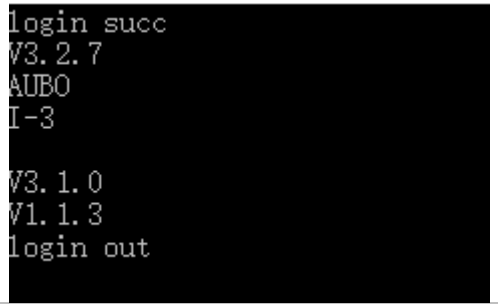
```
int rs_get_device_info(UInt16 rshd, ref RobotDevInfo dev);
```

函数功能	获取机器人设备信息（需连接真实机器人）

参数描述	public struct RobotDevInfo
	{
	//设备型号、芯片型号：上位机主站：0x01 接口板0x02
	byte type;
	//设备版本号，V1.0
	[MarshalAs(UnmanagedType.ByValArray, SizeConst = 16)]
	public char[] revision;
	//厂家ID，"OUR "的ASCII码0x4F 55 52 00
	[MarshalAs(UnmanagedType.ByValArray, SizeConst = 16)]
	public char[] manu_id;
	//机械臂类型
	[MarshalAs(UnmanagedType.ByValArray, SizeConst = 16)]
	public char[] joint_type;
	//机械臂关节及工具端信息
	[MarshalAs(UnmanagedType.ByValArray, SizeConst = 8)]
	public JointVersion[] joint_ver;
	//设备描述字符串以0x00结束
	[MarshalAs(UnmanagedType.ByValArray, SizeConst = 64)]
	public char[] desc;
	//关节ID信息
	[MarshalAs(UnmanagedType.ByValArray, SizeConst = 8)]
	public JointProductID[] jointProductID;
	
	//从设备版本号 - 字符串表示，如“V1.0.0
	[MarshalAs(UnmanagedType.ByValArray, SizeConst = 16)]
	public char[] slave_version;
	//IO扩展板版本号 -字符串标志，如“V1.0.0
	[MarshalAs(UnmanagedType.ByValArray, SizeConst = 16)]
	public char[] extio_version;
	
	}

返回值	调用成功返回 RSERR_SUCC;错误返回错误号

示例	RobotDevInfo dev = new RobotDevInfo();
	rs_get_device_info(rshd, ref dev);
	printchar(dev.revision);
	printchar(dev.manu_id);
	printchar(dev.joint_type);
	printchar(dev.desc);
	printchar(dev.slave_version);
	printchar(dev.extio_version);

输出	

89.**设置最大加速度

90.碰撞恢复 rs_collision_recover

int rs_collision_recover(UInt16 rshd);	
函数功能	获取机器人设备信息（需连接真实机器人）
参数描述	
返回值	调用成功返回 RSERR_SUCC;错误返回错误号
示例	rs_collision_recover(rshd);
输出	

91.获取机械臂当前运行状态 rs_get_robot_state

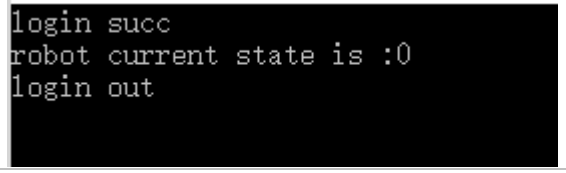
```
int rs_get_robot_state(UInt16 rshd, ref int state);
```

函数功能	获取机械臂当前运行状态
参数描述	state：运行状态，输出参数

	/** 机械臂状态枚举 **/
	enum RobotState
	{
	RobotStopped = 0,
	RobotRunning,
	RobotPaused,
	RobotResumed
	};

返回值	调用成功返回 RSERR_SUCC;错误返回错误号
示例	/*获取机械臂当前运行状态*/

	int state = 0;
	rs_get_robot_state(rshd, ref state);
	Console.WriteLine("robot current state is :" + state);

输出	

92.**获取 Mac 通信状态

93.判断真实机械臂是否存在 rs_is_have_real_robot

```
int rs_get_robot_state(UInt16 rshd, ref int state);
```

函数功能	获取是否存在真实机械臂
参数描述	为传出参数，true：存在真实机械臂　false：不存在真实机械臂
返回值	调用成功返回 RSERR_SUCC;错误返回错误号

示例	//判断真实臂是否存在
	bool state = false;
	rs_is_have_real_robot(rshd, ref state);
	Console.WriteLine("is real robot : " + state);

输出	login succ is real robot : True login out

94.**获取 Joint6 旋转 360° 使能标志

95.**获取机械臂关节状态 rs_get_joint_status

int rs_get_joint_status(UInt16 rshd, ref JointStatus1[] jointstatus);
函数功能　获取机械臂关节状态
参数描述
返回值　调用成功返回 RSERR_SUCC;错误返回错误号
示例
输出

96.获取机械臂诊断信息 rs_get_diagnosis_info

int rs_get_diagnosis_info(UInt16 rshd, ref RobotDiagnosis info);
函数功能　获取机械臂诊断信息
参数描述　传出参数，机械臂诊断信息

	/****机械臂诊断信息****/

	public struct RobotDiagnosis
	{
	//CAN通信状态:0x01~0x80：关节CAN通信错误（每个关节占用1bit）
	//0x00：无错误 0xff：CAN总线存在错误
	public byte armCanbusStatus;
	public float armPowerCurrent; //机械臂48V电源当前电流
	public float armPowerVoltage; //机械臂48V电源当前电压
	public bool armPowerStatus; //机械臂48V电源状态（开、关）
	public char contorllerTemp; //控制箱温度
	public byte contorllerHumidity; //控制箱湿度
	public bool remoteHalt; //远程关机信号
	public bool softEmergency; //机械臂软急停
	public bool remoteEmergency; //远程急停信号
	public bool robotCollision; //碰撞检测位
	public bool forceControlMode; //机械臂进入力控模式标志位
	public bool brakeStuats; //刹车状态
	public float robotEndSpeed; //末端速度
	public int robotMaxAcc; //最大加速度
	public bool orpeStatus; //上位机软件状态位
	public bool enableReadPose; //位姿读取使能位
	public bool robotMountingPoseChanged; //安装位置状态
	public bool encoderErrorStatus; //磁编码器错误状态
	public bool staticCollisionDetect; //静止碰撞检测开关
	public byte jointCollisionDetect; //关节碰撞检测 每个关节占用1bit 0-无碰撞 1-存在碰撞
	public bool encoderLinesError; //光电编码器不一致错误 0-无错误 1-有错误
	public bool jointErrorStatus; //joint error status
	public bool singularityOverSpeedAlarm; //机械臂奇异点过速警告
	public bool robotCurrentAlarm; //机械臂电流错误警告
	public byte toolIoError; //tool error
	public bool robotMountingPoseWarning; //机械臂安装位置错位（只在力控模式下起作用）
	public ushort macTargetPosBufferSize; //mac缓冲器长度
	public ushort macTargetPosDataSize; //mac缓冲器有效数据长度
	public byte macDataInterruptWarning; //mac数据中断

	}
返回值	调用成功返回 RSERR_SUCC;错误返回错误号
示例	/*获取机械臂诊断信息*/

```
            RobotDiagnosis robotdiagnosis = new RobotDiagnosis();
            rs_get_diagnosis_info(rshd, ref robotdiagnosis);
            Console.WriteLine("arm_Canbus_Status: " +robotdiagnosis.armCanbusStatus);
            Console.WriteLine("arm_Power_Current: " + robotdiagnosis.armPowerCurrent);
            Console.WriteLine("arm_Power_Status: " + robotdiagnosis.armPowerStatus);
            Console.WriteLine("arm_Power_Voltage: " + robotdiagnosis.armPowerVoltage);
            Console.WriteLine("brake_Status: " + robotdiagnosis.brakeStuats);
            Console.WriteLine("controller_Humidity: " + robotdiagnosis.contorllerHumidity);
            Console.WriteLine("controller_Temprature: " + robotdiagnosis.contorllerTemp );
            Console.WriteLine("encoderErrorStatus: " + robotdiagnosis.encoderErrorStatus );
            Console.WriteLine("encoderLinesError: " + robotdiagnosis.encoderLinesError );
            Console.WriteLine("forceControlMode: " + robotdiagnosis.forceControlMode );
            Console.WriteLine("jointCollisionDetect: " + robotdiagnosis.jointCollisionDetect );
            Console.WriteLine("jointErrorStatus: " + robotdiagnosis.jointErrorStatus );
            Console.WriteLine("macDataInterruptWarning: " + robotdiagnosis.macDataInterruptWarning );
            Console.WriteLine("macTargetPosBufferSize: " + robotdiagnosis.macTargetPosBufferSize );
            Console.WriteLine("macTargetPosDataSize: " + robotdiagnosis.macTargetPosDataSize );
            Console.WriteLine("orpeStatus: " + robotdiagnosis.orpeStatus );
            Console.WriteLine("remoteEmergency: " + robotdiagnosis.remoteEmergency );
            Console.WriteLine("remoteHalt: " + robotdiagnosis.remoteHalt );
            Console.WriteLine("robotCollision: " + robotdiagnosis.robotCollision );
            Console.WriteLine("robotCurrentAlarm: " + robotdiagnosis.robotCurrentAlarm );
            Console.WriteLine("robotEndSpeed: " + robotdiagnosis.robotEndSpeed );
            Console.WriteLine("robotMaxAcc: " + robotdiagnosis.robotMaxAcc );
            Console.WriteLine("robotMountingPoseChanged: " + robotdiagnosis.robotMountingPoseChanged );
            Console.WriteLine("robotMountingPoseWarning: " + robotdiagnosis.robotMountingPoseWarning );
            Console.WriteLine("singularityOverSpeedAlarm: " + robotdiagnosis.singularityOverSpeedAlarm );
            Console.WriteLine("softEmergency: " + robotdiagnosis.softEmergency );
            Console.WriteLine("staticCollisionDetect: " + robotdiagnosis.staticCollisionDetect );
            Console.WriteLine("toolIoError: " + robotdiagnosis.toolIoError );
```

输出	login succ arm_Canbus_Status: 0 arm_Power_Current: 0.6 arm_Power_Status: True arm_Power_Voltage: 0 brake_Status: False controller_Humidity: 0 controller_Temprature: encoderErrorStatus: False encoderLinesError: False forceControlMode: False jointCollisionDetect: 0 jointErrorStatus: False macDataInterruptWarning: 0 macTargetPosBufferSize: 0 macTargetPosDataSize: 0 orpeStatus: False remoteEmergency: True remoteHalt: False robotCollision: False robotCurrentAlarm: False robotEndSpeed: 1.681558E-42 robotMaxAcc: 0 robotMountingPoseChanged: False robotMountingPoseWarning: False singularityOverSpeedAlarm: False softEmergency: False staticCollisionDetect: False toolIoError: 0 login out

97.**获取机械臂当前关节角信息

98.获取实时路点信息 rs_get_current_waypoint

int rs_get_current_waypoint(UInt16 rshd, ref wayPoint_S waypoint);	
函数功能	获取机械臂当前路点信息

参数描述	Waypoint：为传出参数，路点信息
返回值	调用成功返回 RSERR_SUCC;错误返回错误号

示例	`/*获取实时路点信息*/`
	`wayPoint_S waypoint = new wayPoint_S();`
	`rs_get_current_waypoint(rshd,ref waypoint);`
	`PrintWaypoint(waypoint);`

输出	login succ ---------- pos.x=0.302088821377849 y=-0.213000000205949 z=0.616967379286421 ori.w=0.620541507529403 x=0.620536533593534 y=0.339010674353332 z=-0.339019779044463 joint1=0 joint2=0 joint3=57.2959775885683 joint4=-0.000840650107273366 joint5=-0.000840650107273366 joint6=-0.000840650107273366 ---------- login out

安全 IO 相关

99.使机械臂回初始位

**100. **通知接口板上位机暂停状态

**101. **通知接口板上位机停止状态

**102. **通知接口板上位机错误

**103. **解除系统紧急停止输出信号

**104. **解除缩减模式错误

**105. **防护重置成功

接口板 IO

**106. **获取接口板 IO 配置信息

函数功能	获取接口板指定 IO 集合的配置信息
参数描述	ioType：IO 类型的集合，输入参数 configVector：IO 配置信息的集合，输出参数
返回值	调用成功返回 RSERR_SUCC;错误返回错误号
示例 1	打印 U_DI 的配置信息 /*获取接口板 IO 配置信息*/ std::vector<aubo_robot_namespace::RobotIoType> ioTypeVector;

```
    std::vector<aubo_robot_namespace::RobotIoDesc> ioDescVector;
    //打印User_DI 配置信息
    ioTypeVector.push_back(aubo_robot_namespace::RobotBoardUserDI);
    robotService.robotServiceGetBoardIOConfig(ioTypeVector,ioDescVector);
    std::cout << "ioTypeVector lenth = " << ioTypeVector.size() << std::endl;
    std::cout << "ioDescVector lenth = " << ioDescVector.size() << std::endl;
    for(int i = 0; i < ioDescVector.size(); i++)
    {
        std::cout << "U_DO_" << i << std::endl;
        std::cout << "ioID = " << ioDescVector[i].ioId << " | ";
        std::cout << "ioType = " << ioDescVector[i].ioType << " | ";
        std::cout << "ioName = " << ioDescVector[i].ioName << " | ";
        std::cout << "ioAddr = " << ioDescVector[i].ioAddr << " | ";
        std::cout << "ioValue = " << ioDescVector[i].ioValue << std::endl;
    }
```

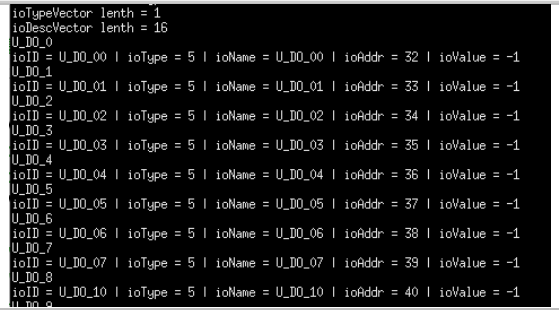
输出

示例 2

打印 U_DI 和 U_DO 的配置信息

```
    /*获取接口板 IO 配置信息*/
    std::vector<aubo_robot_namespace::RobotIoType> ioTypeVector;
    std::vector<aubo_robot_namespace::RobotIoDesc> ioDescVector;
    //打印User_DI 配置信息
    ioTypeVector.push_back(aubo_robot_namespace::RobotBoardUserDI);
    robotService.robotServiceGetBoardIOConfig(ioTypeVector,ioDescVector);
    std::cout << "ioTypeVector lenth = " << ioTypeVector.size() << std::endl;
    std::cout << "ioDescVector lenth = " << ioDescVector.size() << std::endl;
    for(int i = 0; i < ioDescVector.size(); i++)
    {
        std::cout << "U_DO_" << i << std::endl;
        std::cout << "ioID = " << ioDescVector[i].ioId << " | ";
        std::cout << "ioType = " << ioDescVector[i].ioType << " | ";
        std::cout << "ioName = " << ioDescVector[i].ioName << " | ";
        std::cout << "ioAddr = " << ioDescVector[i].ioAddr << " | ";
        std::cout << "ioValue = " << ioDescVector[i].ioValue << std::endl;
    }
    //打印User_DO 配置信息
    ioTypeVector.pop_back();
    ioTypeVector.push_back(aubo_robot_namespace::RobotBoardUserDO);
    robotService.robotServiceGetBoardIOConfig(ioTypeVector,ioDescVector);
    std::cout << "ioTypeVector lenth = " << ioTypeVector.size() << std::endl;
    std::cout << "ioDescVector lenth = " << ioDescVector.size() << std::endl;
    for(int i = 0; i < ioDescVector.size(); i++)
```

	{
	std::cout << "U_DO_" << i << std::endl;
	std::cout << "ioID = " << ioDescVector[i].ioId << "
		";
		std::cout << "ioType = " << ioDescVector[i].ioType << "
			";
			std::cout << "ioName = " << ioDescVector[i].ioName << "
				";
				std::cout << "ioAddr = " << ioDescVector[i].ioAddr << "
					";
					std::cout << "ioValue = " << ioDescVector[i].ioValue << std::endl;
					}

输出	

107. **获取接口板 IO 状态信息

108. **获取接口板 IO 状态信息 rs_get_board_io_status_by_name

int rs_get_board_io_status_by_name(UInt16 rshd, int io_type, string name, ref double val);	
函数功能	根据接口板 IO 类型和名称获取 IO 状态
参数描述	io_type IO 类型 name IO 名称 value IO 状态，输出参数
返回值	调用成功返回 RSERR_SUCC;错误返回错误号
示例	/*获取指定接口板 IO 的状态*/
输出	

109. 获取接口板 IO 状态信息 rs_get_board_io_status_by_addr

```
int rs_get_board_io_status_by_addr(UInt16 rshd, int io_type, int addr, ref double val);
```

函数功能	根据接口板 IO 类型和名称获取 IO 状态
参数描述	io_type　　IO 类型 addr　　IO 地址 　　F1:30 ~ F6:35 　　U_DI_00:36 ~ U_DI_17:51 　　U_DO_00:32 ~ U_DO_17:47 value　　IO 状态，输出参数
返回值	调用成功返回 RSERR_SUCC;错误返回错误号

示例	/*获取指定接口板 IO 的状态*/
	int io_type1 = Robot_User_DO;
	int addr1 = ROBOT_IO_U_DO_00;
	int io_type2 = Robot_User_DI;
	int addr2 = ROBOT_IO_U_DI_00;
	double val = 0;
	rs_get_board_io_status_by_addr(rshd, io_type1, addr1, ref val);
	Console.WriteLine("ROBOT_IO_U_DO_00的状态为："+val);
	rs_get_board_io_status_by_addr(rshd, io_type2, addr2, ref val);
	Console.WriteLine("ROBOT_IO_U_DI_00的状态为：" + val);

输出	login succ ROBOT_IO_U_DO_00的状态为： 1 ROBOT_IO_U_DI_00的状态为： 0 login out

110. **设置接口板 IO 状态 rs_set_board_io_status_by_name

```
int rs_set_board_io_status_by_name(UInt16 rshd, int io_type, string name, double val);
```

函数功能	根据接口板 IO 类型和名称设置 IO 状态
参数描述	io_type　　IO 类型 name　　IO 名称

	value　　IO 状态，输出参数
返回值	调用成功返回 RSERR_SUCC;错误返回错误号
示例	/*设置接口板 IO 状态*/
输出	

111.　设置接口板 IO 状态 rs_set_board_io_status_by_addr

int rs_set_board_io_status_by_addr(UInt16 rshd, int io_type, int addr, double val);	
函数功能	根据接口板 IO 类型和地址设置 IO 状态
参数描述	io_type　　IO 类型 addr　　IO 地址 　　F1:30 ~ F6:35 　　U_DI_00:36 ~ U_DI_17:51 　　U_DO_00:32 ~ U_DO_17:47 value　　IO 状态，输出参数
返回值	调用成功返回 RSERR_SUCC;错误返回错误号

示例	/*设置接口板 IO 状态*/
	int io_type = Robot_User_DO;
	int addr = 32;
	rs_set_board_io_status_by_addr(rshd, io_type, addr, 1);

输出	

查看联机模式

112.　是否在联机模式 rs_is_online_mode

int rs_is_online_mode(UInt16 rshd, ref bool isonline);	
函数功能	返回当前机械臂是否运行在联机模式
参数描述	isonline 输出参数　true：联机 false：非联机
返回值	调用成功返回 RSERR_SUCC;错误返回错误号

示例	/*是否在联机模式*/
	bool isonline=false;
	rs_is_online_mode(rshd, ref isonline);
	Console.WriteLine("robot is online mode : "+isonline);

输出	login succ robot is online mode : True login out

113. 是否在联机主模式 rs_is_online_master_mode

int rs_is_online_master_mode(UInt16 rshd, ref bool ismaster);	
函数功能	返回当前机械臂是否运行在联机主模式
参数描述	ismaster 输出参数 true：联机主模式/手动模式 false：联机从模式
返回值	调用成功返回 RSERR_SUCC;错误返回错误号

示例	/*是否在联机主模式*/
	bool ismaster = false;
	rs_is_online_master_mode(rshd, ref ismaster);
	Console.WriteLine("robot is online master mode : " + ismaster);

输出	login succ robot is online master mode : True login out

安全配置

114. **获取机械臂安全配置

115. **设置机械臂安全配置

116. **获取机械臂安全状态

工具 IO 接口

117. 设置工具端电源电压类型 rs_set_tool_power_type

`int rs_set_tool_power_type(UInt16 rshd, int type);`	
函数功能	设置工具端电源电压类型
参数描述	type：工具电源类型。

	`typedef enum`
	`{`
	`  OUT_0V  = 0,`
	`  OUT_12V = 1,`
	`  OUT_24V = 2`
	`}ToolPowerType;`

返回值	调用成功返回 RSERR_SUCC;错误返回错误号

示例	`/*设置工具电源电压类型*/`
	`int type = OUT_24V;`
	`rs_set_tool_power_type(rshd,type);`

输出	

118. 获取工具端电源电压类型 rs_get_tool_power_type

```
int rs_get_tool_power_type(UInt16 rshd, ref int type);
```

函数功能	获取工具端电源电压类型
参数描述	type：工具电源类型，输出参数

	enum ToolPowerType
	{
	OUT_0V = 0,
	OUT_12V = 1,
	OUT_24V = 2
	}

返回值	调用成功返回 RSERR_SUCC;错误返回错误号
示例	/*获取工具电源电压类型*/

	int type = 0;
	rs_get_tool_power_type(rshd, ref type);
	Console.WriteLine("tool power voltage type is : " + type);

输出	login succ tool power voltage type is : 2 login out

119. **获取工具端的电源电压 rs_get_tool_power_voltage

```
int rs_get_tool_power_voltage(UInt16 rshd, ref double voltage);
```

函数功能	获取工具端电源电压
参数描述	
返回值	调用成功返回 RSERR_SUCC;错误返回错误号
示例	
输出	

120. **设置工具端电源电压类型 and 所有数字量 IO 的类型

121. 设置工具端数字量 IO 的类型 rs_set_tool_io_type

`int rs_set_tool_io_type(UInt16 rshd, int addr, int type);`	
函数功能	设置工具端数字量 IO 的类型

参数描述	addr：IO 地址。
	`Enum addr`
	`{`
	`TOOL_DIGITAL_IO_0 = 0,`
	`TOOL_DIGITAL_IO_1 = 1,`
	`TOOL_DIGITAL_IO_2 = 2,`
	`TOOL_DIGITAL_IO_3 = 3`
	`}`
	type：类型。
	`enum Type //IO类型`
	`{ IO_IN = 0, //输入`
	`IO_OUT //输出`
	`}`

返回值	调用成功返回 RSERR_SUCC;错误返回错误号

示例	`/*设置工具数字 IO 类型*/`
	`int iotype = TOOL_IO_IN;`
	`rs_set_tool_io_type(rshd, 1, iotype);`

输出	

122. **获取工具端所有数字量 IO 的状态

123. **根据地址设置工具端数字量 IO 的状态

124. 根据名称设置工具端 IO 的状态 rs_set_tool_do_status

int rs_set_tool_do_status(UInt16 rshd, string name, int val);	
函数功能	根据名称设置工具端数字量 IO 的状态
参数描述	name:IO 名称。 T_DI/O_00 T_DI/O_01 T_DI/O_02 T_DI/O_03 T_AI_00 T_AI_01 val：IO 状态值。
返回值	调用成功返回 RSERR_SUCC;错误返回错误号

示例	/*根据名称设置工具 IO 状态*/
	
	int val = 1;
	string name = "T_DI/O_00";
	ret =rs_set_tool_do_status(rshd, name, val);

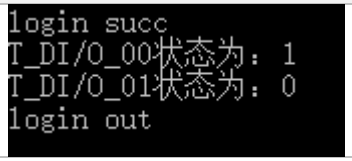
输出	

125. 根据名称获取工具端 IO 的状态 rs_get_tool_io_status

int rs_get_tool_io_status(UInt16 rshd, string name, ref double val);	
函数功能	根据名称设置工具端数字量 IO 的状态
参数描述	name:IO 名称。 T_DI/O_00

	T_DI/O_01 T_DI/O_02 T_DI/O_03 T_AI_00 T_AI_01 val：IO 状态值。
返回值	调用成功返回 RSERR_SUCC;错误返回错误号

示例	/*根据名称设置工具 IO 状态*/
	double val = 0;
	string name1 = "T_DI/O_00";
	string name2 = "T_DI/O_01";
	ret =rs_get_tool_io_status(rshd, name1, ref val);
	Console.Out.WriteLine("T_DI/O_00状态为："+val);
	ret = rs_get_tool_io_status(rshd, name2, ref val);
	Console.Out.WriteLine("T_DI/O_01状态为：" + val);

输出	

126. **获取工具端所有 AI 的状态

固件升级

127. **

设置关节补偿

128. **设置关节碰撞补偿

传送带跟踪

129. **设置编码器复位

130. **启动传送带

131. **停止传送带

132. **设置传送带方向

133. **设置手眼标定结果关系

134. **设置传送带线速度

135. **设置编码器距离关系

136. **设置传送带起始窗口上限

137. **设置传送带起始窗口下限

138. **设置传送带跟踪轨迹下限

139. **设置传送带跟踪的最大速度

140. **设置传送带跟踪的最大加速度

141. **设置传送带跟踪的系统延时时间

142. **设置机械臂工具

八、 错误代码

错误号	错误代码	错误信息
0	InterfaceCallSuccCode	成功
10001	ErrCode_Failed	通用失败
10002	ErrCode_ParamError	参数错误
10003	ErrCode_ConnectSocketFailed	Socket 连接失败
10004	ErrCode_SocketDisconnect	Socket 断开连接
10005	ErrCode_CreateRequestFailed	创建请求失败
10006	ErrCode_RequestRelatedVariableError	请求相关的内部变量出错
10007	ErrCode_RequestTimeout	请求超时
10008	ErrCode_SendRequestFailed	发送请求信息失败
10009	ErrCode_ResponseInfoIsNULL	响应信息为空
10010	ErrCode_ResolveResponseFailed	解析响应失败
10011	ErrCode_FkFailed	正解出错
10012	ErrCode_IkFailed	逆解出错
10013	ErrCode_ToolCalibrateError	工具标定参数有错

10014	ErrCode_ToolCalibrateParamError	工具标定参数有错
10015	ErrCode_CoordinateSystemCalibrateError	坐标系标定失败
10016	ErrCode_BaseToUserConvertFailed	基坐标系转用户座标失败
10017	ErrCode_UserToBaseConvertFailed	用户坐标系转基座标失败
10018	ErrCode_MotionRelatedVariableError	运动相关的内部变量出错
10019	ErrCode_MotionRequestFailed	运动请求失败
10020	ErrCode_CreateMotionRequestFailed	生成运动请求失败
10021	ErrCode_MotionInterruptedByEvent	运动被事件中断
10022	ErrCode_MotionWaypointVetorSizeError	运动相关的路点容器的长度不符合规定
10023	ErrCode_ResponseReturnError	服务器响应返回错误
10024	ErrCode_RealRobotNoExist	真实机械臂不存在，因为有些接口只有在真是机械臂存在的情况下才可以被调用
10025	ErrCode_moveControlSlowStopFailed	调用缓停接口失败
10026	ErrCode_moveControlFastStopFailed	调用急停接口失败
10027	ErrCode_moveControlPauseFailed	调用暂停接口失败
10028	ErrCode_moveControlContinueFailed	调用继续接口失败