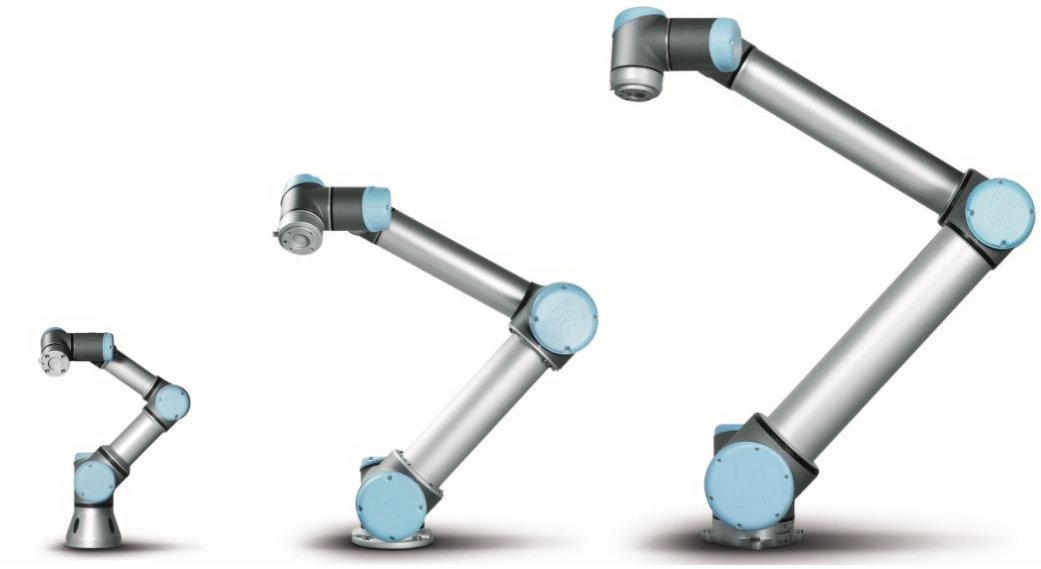
优傲机器人技术文档

Universal Robots Technical Document

文 档 名 称： URSDK (.Net 4.0) 快速指南

Document Name：URSDK (.Net 4.0) Quick Start



UR3, UR5 & UR10/CB3.0 及之上

优傲机器人贸易（上海）有限公司

Universal Robots (Shanghai) Co. Ltd.

www.universal-robots.com

2017 年 8 月 30 日

SUMMARY OF DOCUMENT REVISIONS			
Rev. No.	**Date Revised**	**Section Revised**	**Revision Description**
1	2017.8.30	Draft	Issued by CSL
2			
3			

目 录

1.总述

URSDK 是.Net 版的 UR 机器人驱动，实现了 UR 机器人的 29999-30004 端口，提供 UR 机器人的远程调用，控制和监控功能。若需自行解析这些端口，可参考：29999（Dashboard server）, 30001-30003（Primary, secondary&Realtime）, 30004（RTDE）。

URSDK (.Net 4.0) 快速指南主要介绍 URSDK 的大概功能，并提供一些调用示例，方便用户的快速学习和使用。如有任何疑问或者反馈，请发送至 support.china@uinversal-robots.com 。

URSDK 包含 URSDK.dll, MathNet.Numerics.dll, RTDEConfig.xml。其中 URSDK.dll 为 SDK 的核心库，MathNet.Numerics.dll 为数学库，RTDEConfig.xml 为 RTDE(30004)配置文件，如果使用 RTDE，必须指定指定 RTDEConfig.xml 的路径。

URSDK 快速指南目标是让读者快速学会 URSDK 的使用，详细的说明和资料请参考 UR Support 以及 URSDK 的参考手册。

使用要求：

1、.Net framework 4.0 及之上；

2、PolyScope3.2 及之上。

针对的人群：

1、 开发 UR 机器人上位机软件的工程师；

2、 熟悉 UR 机器人脚本和操作的工程师；

2. 测试环境搭建

.Net 开发环境搭建并不是本文的范畴，可根据个人的需求安装。本文使用 C#开发语言，IDE 为 Visual Studio 2015。UR 上位机软件的开发测试可以针对真实的机器人也可以是 URSim 仿真软件。真实机器人的测试只需配置好网络环境，确保上位机能够连接上机器人即可。

URSim 仿真软件对于软件的开发，快速测试非常方便。URSim 是运行在 Linux 上的软件，因此如果开发机器人是 Windows，那么可以安装虚拟机软件（Vmware workstation 12 或者 Virtual box，本文采用 Vmware workstation）,然后下载 Ursim for non-linux。为了完成 Windows 软件与虚拟机软件中 URSim 的通讯，还需配置网络。

Vmware 虚拟机软件安装完成后会在网络连接中增加两块网卡即 VMnet1 和 VMnet8，将 VMnet1 网卡设置为固定 IP，虚拟机网卡设置为仅主机模式如图 1。

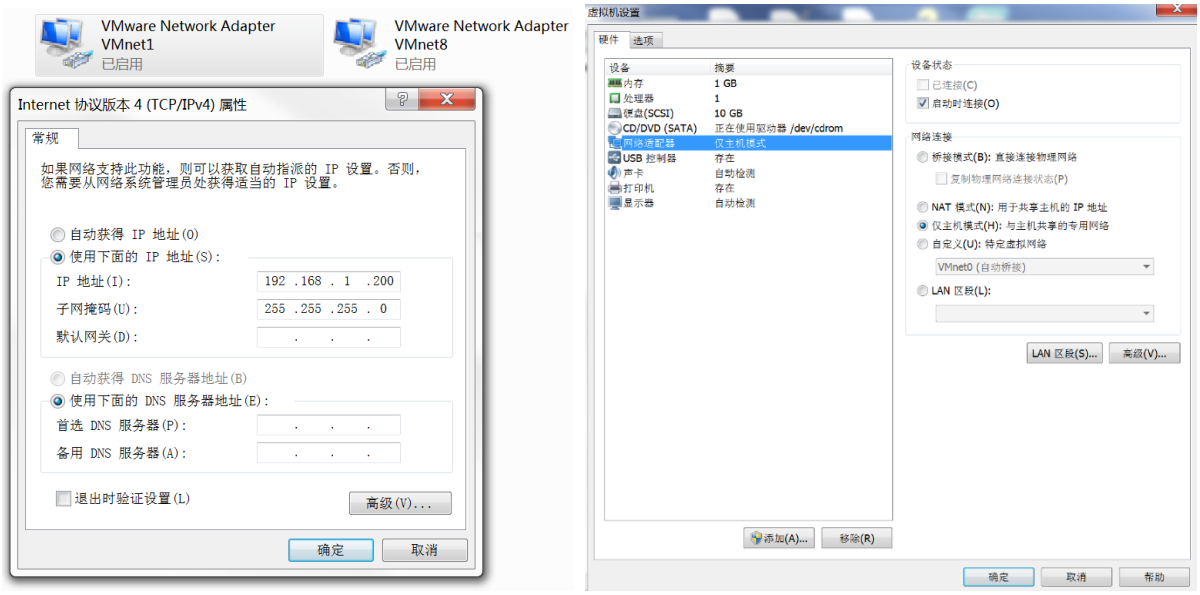


图 1 虚拟网卡(VMnet1)和虚拟机设置

与真机不同的是，URSim 的网络设置不能通过 PolyScope 设置，而要通过修改 linux 系统的 ip 地址实现，修改方法为将 /etc/network/interfaces 文件中的网卡配置修改为静态 IP，图 2 为一个典型的静态 IP 设置方式。注意 IP 地址与 Windows 虚拟网卡处于同一网段。

```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
#iface eth0 inet dhcp
iface eth0 inet static
address 192.168.1.40
netmask 255.255.255.0
```

图 2 Linux 静态 IP 设置

3. URSDK-Communication 命名空间使用

URSDK 包含三个命名空间，分别是 Communication，Datatype，MathLib。Communication 是 29999-30004 端口的驱动类，Datatype 则是一些数据类型，MathLib 包含数学运算类。本文将侧重如何利用 URSDK 实现与 UR 机器人或者 URSim 的通讯，因此主要涵盖 Communication 中的驱动类。

3.1 Dashboard server(29999)

Dashboard 是由机器人人机界面进程（示教器显示）负责维护和执行的一个端口。该端口主要负责接收上位机指令，执行机器人初始化、加载程序、开始和暂停程序运行以及设置用户角色等操作，上位机可以远程操作机器人就如同操作示教器一样。该接口主要应用在自动初始化机器人，无示教器应用（无示教器情况下如何设置 UR 机器人请参考无示教器设置）等场合。Dashboard 接口接收上位机发送的 Dashboard 命令字符串，机器人接收后，返回执行结果字符串。详细请参考 Dashboard 使用。

对应的，URSDK 中 DashBoard 类实现了 Dashboard 中所有指令，其调用方式为：

```
using URSDK.RobController.Communication;                //引用 communication 命名空间

DashBoard aDashBoard=new DashBoard("192.168.1.40");    //IP 为机器人的地址

String returnString;

//加载 test1.urp 程序，该程序需在机器人 programs 文件夹下
returnString = aDashBoard.loadProgram("test1.urp");
returnString = aDashBoard.play();                              //运行当前程序

//其他操作
returnString = aDashBoard.stop();                            //停止程序运行
returnString = aDashBoard.quit();                             //退出 DashBoard 连接
```

3.2 Primary interface(30001) & Secondary interface(30002)

Primary interface 以 10Hz 的频率向外发送机器人的状态，并以消息的形式对外发送机器人的一些运行错误，全局变量信息，TP 按钮消息等等；同时能通过 Primary interface 向机器人发送脚本指令（脚本指令下载）或者脚本程序段以实现对机器人的控制。详细请参考 Remote Control via TCP/IP 中的 client_interface 文档。由于 Secondary interface 可以理解为 Primary interface 的子集（除了不会以消息形式对外发送机器人的各种消息外，其余功能跟 Primary interface 一致），因此本小节中除了消息事件外，都适用于 Secondary interface。

PrimaryInterface 类是 Primary interface 端口的驱动， 其包含的数据成员有:

configurationData	配置信息，包含每个关节极限位置设置，DH 参数等数据
kinematicsInfo	机器人运动学信息，内部使用
safetyData	安全数据，内部使用
status	Sync 同步状态，PrimaryInterface.Status.Stopped/Started/Syncing
version	控制器版本信息

PrimaryInterface 类成员方法：

public PrimaryInterface(string IPAddress, int port = 30001)

PriamryInterface 类构造方法，IPAddress 字符串是被连接机器人的 IP 地址，例如“127.0.0.1”，port 默认是 30001，即 Primary interface，如果选择 port 不等于 30001，那么 port 将被设置为 30002，Secondary interface。

public void startPrimary()

该方法用于启动 PrimaryInterface 的后台同步，只有启动了后台同步，才可以读取机器人的状态(getRobotState)，订阅的事件才会生效，才能够发送脚本给机器人控制器。

public robotState getRobotState()

此方法用于用于获取机器人的状态（数据每 100ms 更新一次），返回 robotState 类型数据。robotState 包含数据包括：robotModeData，jointData，cartesianInfo，masterboardData，toolData，kinematicsInfo，configurationData，forceModeData，additionalInfo 和 calibrationData，详情请见 client interface(底部) 。机器人每次发回的数据并不一定包含上面所有的数据，因此在读取之前一定要检查是否为空(null)。

public bool sendScript(string scripts)

脚本发送方法，脚本发送格式请参考《UR 机器人与 PC 通讯》。发送成功返回 True，否则返回 False。

public void stopPrimary()

停止 PrimaryInterface 的后台同步，订阅的事件不会再被触发（仍需取消订阅），脚本不能被发送。

PrimaryInterface 类使用方法：

1、 添加对 communication 和 Datatype(含 Vector6<T>)命名空间；

```
using URSDK.RobController.Communication;
using URSDK.RobController.Datatype;
```

2、 新建 PrimaryInterface 对象，如果端口号是 30002，那么连接的是 Secondary interface；

```
PrimaryInterface primaryInt = new PrimaryInterface(“192.168.1.30”,30001);
```

3、 启动后台同步；

```
primaryInt.startPrimary();
```

4、 读取 robotState 数据；

```
robotState state=primaryInt.getRobotState();
```

5、 使用 robotState 中的数据；

```
if(state.cartesianInfo!=null){
  Vector6<double> tcp=state.cartesianInfo.tcp;
}
```

6、 给机器人发送脚本，返回发送成功与否结果；

```
bool sendSucc=primaryInt.sendScript("set_standard_digital_out(0,True)");
```

7、 使用完成之后，停止后台同步，释放资源；

```
primaryInt.stopPrimary();
```

事件订阅：

Primary interface(30001)还提供一些机器人事件消息（Secondary interface 无），可以根据需求来订阅，事件需要在后台同步开始后才能触发。

Primary interface 事件：

1. globalVariablesSetupMessageEvent， 全局变量设置事件（程序开始运行时触发）
2. globalVariablesUpdateMessageEvent， 全局变量更新事件（程序运行中触发）
3. keyMessageEvent， 按钮事件（如程序启动或停止）
4. labelMessageEvent， 标签事件（脚本中$标志行）
5. requestValueMessageEvent， 请求数据事件（例如 TP 中的 popup message）
6. robotCommMessageEvent， 机器人通讯事件（如 log 中的部分信息）
7. runtimeExceptionMessageEvent， 程序运行时错误事件（如程序中的报错）
8. safetyModeMessageEvent， 安全模式事件（如 log 中的部分消息）
9. textMessageEvent， 文本数据事件（如脚本中 textmsg 发送数据到 log）
10. varMessageEvent， 已废弃。
11. versionMessageEvent， 控制器版本消息（一般会在首次连接时发送）

事件的订阅和使用方法（这里以 keyMessage 为例，其余类似）：

```
//Primary Interface 对象实例化
public PrimaryInterface demoInstance=new PrimaryInterface("192.168.1.30");
public keyMessage msg;

public void yourEventHandler(object sender, keyMessageEventArgs e){
 msg=e.keyMessage;                                          //事件处理函数实现
}
demoInstance.keyMessageEvent+=yourEventHandler;             //事件注册

demoInstance.startPrimary();                                //启动同步
...    ...                                                  //其他处理
demoInstance.stopPrimary();                                 //停止同步
demoInstance.keyMessageEvent-=yourEventHandler;             //事件取消注册
```

实现事件处理函数需要注意的是，由于其运行在背景线程中，因此如果需要改变 UI 的内容，例如修改文本框中的数据，不能采用直接修改(例如 textbox.text=”hello”)的方式，而应该使用 invoke 方式：

```
private delegate void InvokeCallback(string msg); //委托定义

//事件处理函数
public void yourEventHandler(object sender, keyMessageEventArgs e){
    if(textBox2.InvokeRequired){                //判断是否需要invoke
            InvokeCallback realCall = new InvokeCallback(changeGUI);
            this.Invoke(realCall, new object[] { e.keyMessage.textMessage });
    }
    Else{
            changeGUI(e.keyMessage.textMessage);
        }
}

//界面更新函数
private void changeGUI(string msg){
            textBox2.Text = msg;
}
```

事件参数（例如 keyMessageEventArgs）包含触发该事件的消息本身（例如 keyMessage），对于多数事件处理来说，只需获取消息本身之后即可使用，这里面有两个消息事件比较特殊，一个是 globalVariablesSetupMessageEvent，另外一个是 globalVariablesUpdateMessageEvent；前者是全局变量设置消息（后称设置消息），后者是全局变量更新消息（后称更新消息）。设置消息是当启动程序运行之后，控制器脚本执行器建立全局变量表之后就会发送这个消息；更新消息则是在程序执行过程之中，一旦有全局变量值发生变化，就会发送更新消息。二者传递给事件处理函数的事件参数(globalVariablesSetupMessageEventArgs ，globalVariablesUpdateMessageEventArgs)均包含了相应的消息成员（globalVariablesSetupMessage ，globalVariablesUpdateMessage）。

globalVariablesSetupMessage 主要数据成员：

timeStamp: 表示收到消息的控制器时间。

startIndex: 表示设置消息中全局变量名偏置，例如全局变量非常多时，第一个设置消息 startIndex=0，含 N 个变量名，第二个设置消息的 startIndex=N，含 M 个变量名，第三个设置消息的 startIndex=M+N……

variableNames: ArrayList 类型，可以利用循环读出所有的变量名。

globalVariablesUpdateMessage 主要数据成员：

timeStamp: 表示收到消息的控制器时间。

startIndex: 表示更新消息中全局变量数据和类型的偏置，例如全局变量非常多时，第一个更新消息 startIndex=0，含 N 个变量数据，第二个更新消息的 startIndex=N，含 M 个变量数据，第三个更新消息的 startIndex=M+N……

varType: ArrayList 类型，包含对应 globalVariablesSetupMessage 中变量的类型。

varValue: ArrayList 类型，包含对应 globalVariablesSetupMessage 中变量的数据。

以 globalVariablesUpdateMessage 的解析为例：

```
//本例演示了如何解析PoLyScope3.3 及其以上版本的更新消息，3.2 版本的解析需将 valueTypes.valueType3_3 替换成 valuetypes.valueType
//本例演示了从更新消息中解析出每个变量的值，并将他们连接成字符串显示在文本框中。

private delegate void InvokeCallback(string msg); //委托定义

public void variableMsgHandler(object sender, globalVariablesUpdateMessageEventArgs e)
        {
            String variableString = "";
            valueTypes.valueType3_3 Type3_3;
            if (a.version.minorVersion >2) {                                //3.3版本valueType与之前不一样,之前的版本应该用valueTypes.valueType
                for (int ii = 0; ii < e.globalVariablesUpdateMessage.varType.Count; ii++) {
                    //判断数据是否为ArrayList还是valueTypes.valueType3_3，如果为arraylist，那么其数据就是list类型，元素为List中每个元素的类型
                    if (e.globalVariablesUpdateMessage.varType[ii].GetType() == typeof(valueTypes.valueType3_3))
                        Type3_3 = (valueTypes.valueType3_3)e.globalVariablesUpdateMessage.varType[ii];
                    else
                        Type3_3 = valueTypes.valueType3_3.ListVal;
                    switch (Type3_3)
                    {
                        case valueTypes.valueType3_3.BoolVal:  //布尔类型
                            variableString=variableString+ ((bool)e.globalVariablesUpdateMessage.varValue[ii]).ToString()+",";
                            break;
                        case valueTypes.valueType3_3.FloatVal: //单精度浮点数类型
                            variableString = variableString + ((float)e.globalVariablesUpdateMessage.varValue[ii]).ToString()+",";
                            break;
                        case valueTypes.valueType3_3.IntVal:   //整数类型
                            variableString = variableString + ((int)e.globalVariablesUpdateMessage.varValue[ii]).ToString()+",";
                            break;
                        case valueTypes.valueType3_3.NoneVal:  //空类型
                            break;
                        case valueTypes.valueType3_3.NumVal:   //双精度浮点数类型
                            variableString = variableString + ((double)e.globalVariablesUpdateMessage.varValue[ii]).ToString()+",";
                            break;
                        case valueTypes.valueType3_3.PoseVal:  //位姿变量类型
                            variableString = variableString + ((Vector6<double>)e.globalVariablesUpdateMessage.varValue[ii]).ToString()+",";
                            break;
                        case valueTypes.valueType3_3.ConstStringVal:   //常字符串类型
                            variableString = variableString + (string)e.globalVariablesUpdateMessage.varValue[ii]+",";
                            break;
                        case valueTypes.valueType3_3.VarStringVal:     //变字符串类型
                            variableString = variableString + (string)e.globalVariablesUpdateMessage.varValue[ii] + ",";
                            break;
                        case valueTypes.valueType3_3.ListVal:       //List类型
                            ArrayList tempType = (ArrayList)e.globalVariablesUpdateMessage.varType[ii];
                            ArrayList tempData = (ArrayList)e.globalVariablesUpdateMessage.varValue[ii];
                            //解析出List类型中的数据
                            for (int jj = 0; jj < tempType.Count; jj++)
                            {
                                switch ((valueTypes.valueType3_3)(tempType[jj])){
                                    case valueTypes.valueType3_3.BoolVal:
                                        variableString = variableString + ((bool)tempData[jj]).ToString() + ",";
                                        break;
                                    case valueTypes.valueType3_3.FloatVal:
                                        variableString = variableString + ((float)tempData[jj]).ToString() + ",";
                                        break;
                                    case valueTypes.valueType3_3.IntVal:
                                        variableString = variableString + ((int)tempData[jj]).ToString() + ",";
                                        break;
                                    case valueTypes.valueType3_3.NoneVal:
                                        break;
                                    case valueTypes.valueType3_3.NumVal:
                                        variableString = variableString + ((double)tempData[jj]).ToString() + ",";
                                        break;
                                    case valueTypes.valueType3_3.PoseVal:
                                        variableString = variableString + ((Vector6<double>)tempData[jj]).ToString() + ",";
                                        break;
                                    case valueTypes.valueType3_3.ConstStringVal:
                                        variableString = variableString + (string)tempData[jj] + ",";
                                        break;
                                    case valueTypes.valueType3_3.VarStringVal:
                                        variableString = variableString + (string)tempData[jj] + ",";
                                        break;
                                    default:
                                        break;
                                }
                            }
                            break;
                        default:
                            break;
                    }
                }
            }

            //更新界面信息
            if(textBox2.InvokeRequired){
                InvokeCallback realCall = new InvokeCallback(changeGUI);
                this.Invoke(realCall, new object[] {variableString });
            }
            else{
               changeGUI(variableString);
            }

        }

        private void changeGUI(string msg)
        {
            textBox2.Text = msg;
        }
```

3.3 Realtime interface(30003)

Realtime interface 以 125Hz 的频率往外发送机器人的详细状态信息（位置，速度，关节电流，力矩等），同时也可以接收上位机发送的脚本指令并立即执行。Realtime 通讯的详细解析请参考 Remote Control via TCP/IP。

RTClient 类时 Realtime interface 的驱动类，其主要的数据成员为 Status: Status==RTClientStatus.Started 表示已启动后台同步，但是还未同步数据; Status=RTClientStatus.Syncing 表示已同步数据；Status= RTClientStatus.Stopped 表示后台同步已关闭。

RTClient 类成员方法：

public RTClient(string IP = "127.0.0.1",int Port = 30003)

RTClient 构造函数，IP 为连接机器人控制器地址，端口号只能是 30003。

public void startRTClient()

该方法用于启动 RTClient 的后台同步，只有启动了后台同步，才可以读取机器人的状态(getRTClientObj)和发送脚本给机器人控制器。

public RTClientObj getRTClientObj()

此方法用于获取机器人的状态（数据每 8ms 更新一次），返回 RTClientObj 类型数据。RTClientObj 包含数据包括机器人关节角度，速度,TCP 位置，IO 状态等等信息，详细请参考 URSDK 参考手册和 client_interface(底部) 。机器人每次发回的数据并不一定包含上面所有的数据，因此在读取之前一定要检查是否为空(null)。

public bool sendScript(string script)

脚本发送方法，脚本发送格式请参考《UR 机器人与 PC 通讯》。发送成功返回 True，否则返回 False。

public void stopRTClient()

停止 RTClient 的后台同步，无法获取当前机器人状态(RTClientObj)，不能发送脚本。

RTClient 类使用方法：

```
using URSDK.RobController.Communication;      //添加命名空间引用

RTClient aRTClient=new RTClient("192.168.1.100");          //IP 为机器人的地址
aRTClient.startRTClient();                                 //启动 RTClient server

if(aRTClient.Status==RTClientStatus.Syncint){
     RTClientObj aRTClientObj=getRTClientObj();            //读取 aRTClientObj
     double currenttime= aRTClientObj.timestamp;
 //使用 aRTClientObj 中的数据
aRTClientObj.sendScript("movel(p[0.1,0.2,0.3,0,0,0])");    //发送脚本给机器人;
}

aRTClientObj.stopRTClient();                          //停止 RTClient server 释放资源
```

3.4 Realtime Data Exchange(RTDE, 30004)

RTDE 可以根据配置以 125Hz 的频率往外发送机器人的详细状态信息（位置，速度，关节电流，力矩等），机器人也只会接收配置的输入信息（IO，寄存器等）。与 Realtime 接口相比，RTDE 接口不能接收脚本指令，只能接收配置好的输入信息；但是 RTDE 接口不会中断当前程序的执行，而且由于输入输出信息都是经过配置，传输的信息没有不需要的数据，可以按需启动和停止，因此能够大大降低对网络带宽的占用。RTDE 接口的详细资料请参考 RTDE 通讯。

RTDE 类是 RTDE 的驱动类，包含的主要数据成为是：
isSynchronizing，bool 类型，表明后台同步是否正在进行；
URControlVersion, URControlVersion 类型，代表连接的机器人控制器版本号。

RTDEConfig.xml 文件是 RTDE 类的配置文件，其示例如下，<Group key="In">中定义机器人接收的信息和类型，<Group key="Out">中则定义需要机器人发送的状态信息，详见 RTDE 通讯。

```
<?xml version="1.0"?>
<rtde_config>
  <Group key="In">
      <field name="standard_digital_output_mask" type="UINT8"/>
      <field name="standard_digital_output" type="UINT8"/>
      <field name="input_int_register_22" type="INT32"/>
      <field name="input_int_register_23" type="INT32"/>
      <field name="input_double_register_0" type="DOUBLE"/>
      <field name="input_double_register_1" type="DOUBLE"/>
      <field name="input_double_register_2" type="DOUBLE"/>
      <field name="input_double_register_3" type="DOUBLE"/>
      <field name="input_double_register_4" type="DOUBLE"/>
      <field name="input_double_register_5" type="DOUBLE"/>
  </Group>
  <Group key="Out">
        <field name="joint_control_output" type="VECTOR6D"/>
        <field name="actual_q" type="VECTOR6D"/>
        <field name="robot_mode" type="INT32"/>
        <field name="actual_TCP_pose" type="VECTOR6D"/>
        <field name="output_int_register_22" type="INT32"/>
        <field name="output_int_register_23" type="INT32"/>

  </Group>
</rtde_config>
```

RTDE 成员方法：

```
public RTDE(string filepath,string IPAddr = "127.0.0.1",int portnumber = 30004)
```

构造函数,filepath 是配置文件 RTDEConfig.xml 的路径，IPAddr 是连接机器人的 IP 地址，端口只能是 30004。

```
public void startSync()
```

该方法用于启动 RTDE 的后台同步，只有启动了后台同步，才可以读取机器人的状态(*getOutputObj*)和发送消息（*sendMessage*）和输入（*setInputObj*）给机器人控制器。

```
public RTDEOutputObj getOutputObj()
```

此方法用于获取机器人的输出（数据每 8ms 更新一次），返回 RTDEOutputObj 类型数据。RTDEOutputObj 包含数据需是在 RTDEConfig.xml 中定义的输出。

public bool setInputObj(RTDEInputObj InputObj)

此方法用于设置机器人的输入，传入参数为 RTDEInputObj 类型，RTDEInputObj 包含的数据需是 RTDEConfig.xml 中定义的输入，否则不会发送给机器人控制器。

public bool sendMessage(RTDEMessage msg)

此函数用于发送消息到机器人 log 窗口，类似脚本指令 testmsg(s1,s2=’’)。传入参数为 RTDEMessage

public void stopSync()

此方法用于停止 RTDE 的后台同步，读取机器人状态，发送消息以及设置输入都将不可用。

RTDE 类使用方法：

```
using URSDK.RobController.Communication;   //添加命名空间

//IP 为机器人 IP 地址，RTDEConfig.xml 为 RTDE output,input 配置文件
aRTDE=new RTDE("C:\\RTDEConfig.xml","192.168.1.100");

try{
   aRTDE.startSync();              //启动 RTDE server 同步
}
catch{
  throw;
}
If(aRTDE.isSynchronizing){

  RTDEOutputObj outobj=aRTDE.getOutputObj();     //读取 RTDE output 数据
  string tcppose= outobj.actual_TCP_pose.tostring();

   RTDEMessage aMsg=new RTDEMessage("Content of message","Message source",
   RTDEMessage.INFO_MESSAGE);  //创建 RTDEMessage 对象，Warning Level 为 Info

   aRTDE.sendMessage(aMsg);                          //发送 RTDEMessage

   RTDEInputObj inobj=new RTDEInputObj();          //创建 RTDE input 对象
// 将数字输出置为 0b00000011,RTDEInputObj 同步数据一定要是 RTDEConfig.xml 文件中已经配置好的 input
   inobj.standard_digital_output=3;

   aRTDE.setInputObj(inobj);                       //发送 input
}

aRTDE.stopSync();                            //关闭 RTDE server 同步，释放资源
```

4. URSDK-DataType 命名空间使用

DataType 命名空间包含一些与机器人相关的数据类型如下表，其使用方式可参考《URSDK 参考手册》。

数据类型	说明
JointPose	关节位姿，包含六个关节的角度值
Pose	TCP 位姿，由 trans 和 rot 组成

quaternion	四元数
rot	旋转矢量(Rx,Ry,Rz)
trans	平移位置(X,Y,Z)
URControlVersion	RTDE 返回的控制器版本类型
Vector3<T>	3 向量泛型
Vector6<T>	6 向量泛型，主要用于 tcp 数据和关节数据的表示

5. URSDK-MathLib 命名空间使用

MathLib 命名空间主要包含一些位姿变换函数。

public static quaternion Matrix2Quaternion(Matrix *Min*)	三维旋转矩阵转四元数
public static rot Matrix2RotVet(Matrix *Min*)	三维旋转矩阵转旋转矢量
public static rot Matrix2RPY(Matrix *Min*)	三维旋转矩阵转 RPY 角
public static Pose pose_inv(Pose *Pose*)	Pose 求逆
public static Pose pose_trans(Pose *a*, Pose *b*)	Pose 相乘
public static rot Quaternion2RotVec(quaternion *Qin*)	四元数转旋转矢量
public static rot Quaternion2RPY(quaternion *Qin*)	四元数转 RPY 角
public static Matrix RotVec2Matrix(rot *Rin*)	旋转矢量转三维旋转矩阵
public static quaternion RotVec2Quaternion(rot *Rin*)	旋转矢量转四元数

public static rot RotVec2RPY(rot *Rin*)	旋转矢量转 RPY 角
public static Matrix RPY2Matrix(rot *Rin*)	RPY 角转三维旋转矩阵
public static quaternion RPY2Quaternion(rot *Rin*)	RPY 角转四元数
public static rot RPY2RotVec(rot *Rin*)	RPY 角转旋转矢量
LsParaObj 类	最小二乘估计