Dobot Magician Demo Description

Issue: V1.1

Date: 2019-03-12

Shenzhen Yuejiang Technology Co., Ltd

Disclaimer

To the maximum extent permitted by applicable law, the products described (including its hardware, software and firmware, etc.) in this document are provided **AS IS**, which may have flaws, errors or faults. Yuejiang makes no warranties of any kind, express or implied, including but not limited to, merchantability, satisfaction of quality, fitness for a particular purpose and non-infringement of third party rights. In no event will Yuejiang be liable for any special, incidental, consequential or indirect damages resulting from the use of our products and documents.

Before using our product, please thoroughly read and understand the contents of this document and related technical documents that are published online, to ensure that the robotic arm is used on the premise of fully understanding the robotic arm and related knowledge. Please use this document with technical guidance from professionals. Even if follow this document or any other related instructions, Damages or losses will be happen in the using process, Dobot shall not be considered as a guarantee regarding to all security information contained in this document.

The user has the responsibility to make sure following the relevant practical laws and regulations of the country, in order that there is no significant danger in the use of the robotic arm.

Shenzhen Yuejiang Technology Co., Ltd

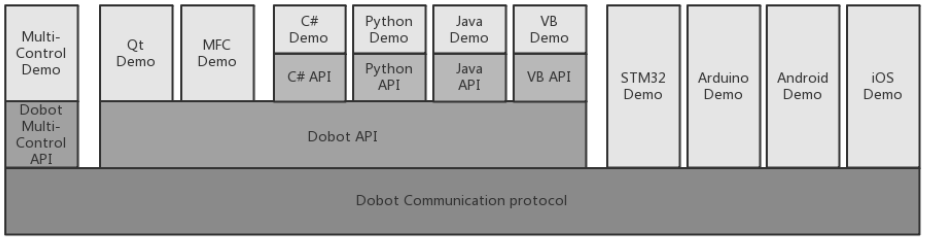
Address: 3F, Building NO.3, Tongfuyu Industrial Town, Nanshan District, Shenzhen, China

Website: www.dobot.cc

Preface

Purpose

This document describes the secondary development environment building and demo codes in multiple languages, frameworks, and systems, aiming to help secondary developer to understand common API of Dobot Magician and build development environment quickly.



Intended Audience

This document is intended for:

- Customer Engineer
- Installation and Commissioning Engineer
- Technical Support Engineer

Change History

Date	Change Description
2018/03/01	The first release
2019/03/12	Update the parameters of the ConnectDobot method in the Java Demo; Update C# Demo interface display diagram and get pose code.

Symbol Conventions

The symbols that may be founded in this document are defined as follows.

Symbol	Description
⚠DANGER	Indicates a hazard with a high level of risk which, if not avoided, could result in death or serious injury
⚠WARNING	Indicates a hazard with a medium level or low level of risk which, if not avoided, could result in minor or moderate injury, robotic arm damage
⚠NOTICE	Indicates a potentially hazardous situation which, if not avoided, can result in robotic arm damage, data loss, or unanticipated result
🕮NOTE	Provides additional information to emphasize or supplement important points in the main text

Contents

1. Common System

For common system, we have supported DLLs for secondary developer. You can call DLL directly to control Dobot Magician without development related to communication protocol.

1.1 Dobot DLL

The source codes and precompiled files can be found in **DobotDLL** directory. Please use Qt 5.6 software to check source codes. In addition, the corresponding DLLs for Windows 32-bit, Windows 64-bit, Linux and Mac can also be found in this directory.

1.1.1 Compiling

Please download the Qt version for your system and install it.

The download path is https://download.qt.io/archive/qt/5.6/5.6.0/

⚠NOTICE

If the Qt library is used when compiling DLLs, please use the Qt software with MSVC compiler and compile Dobot DLLs with MSVC.

1.1.2 Usage

- For Windows OS, please add the DLLs directory to environment variable **Path**.
- For Linux OS, please add the following statement at the end of **~/.bash_profile** file and restart computer.

Program 1.1 Add statement in Linux OS

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:DOBOT_LIB_PATH
```

- For Mac OS, please add the following statement at the end of **~/.bash_profile** file and restart computer.

Program 1.2 Add statement in Max OS

```
export DYLD_LIBRARY_PATH=$DYLD_LIBRARY_PATH: DOBOT_LIB_PATH
```

1.2 Java Demo

1.2.1 Project Description

Configure environment: Import **jna**, so that Java can access the local DLL directly.

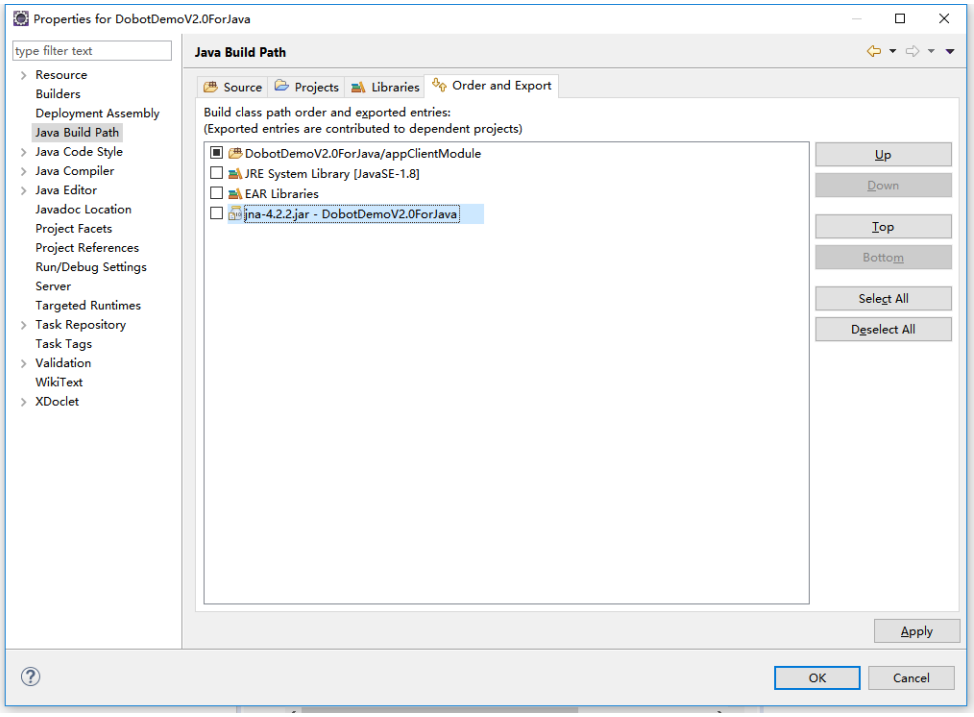


Figure 1.1 Environment configuration

1.2.2 Java API

DobotDll.java encapsulates the C type interface of Dobot DLL secondary, which is Java API of Dobot. The example for loading DLL is shown as follows.

Program 1.3 Load DLL

```
DobotDll instance = (DobotDll) Native.loadLibrary("DobotDll",    DobotDll.class);
```

DobotDll in the example is the DLL name in Windows OS. Please modify the DLL name according to the different OS.

1.2.3 Code Description

(1) Connect to Dobot Magician and check whether the connection is successful.

Program 1.4 Connect to Dobot Magician and check whether the connection is successful

```
IntByReference ib = new IntByReference();
DobotResult ret = DobotResult.values()[ DobotDll.instance.ConnectDobot((char)0, 115200 ,(char)0,(char)0)];
// Start to connect
if ( ret == DobotResult.DobotConnect_NotFound ||
        ret == DobotResult.DobotConnect_Occupied )
{
      Msg("Connect error, code:" + ret.name());
      return;
}
```

```
Msg("connect success code:" + ret.name());
```

(2) Set the offset of the end effector.

Program 1.5 Set the offset of end effector

```
EndEffectorParams endEffectorParams = new EndEffectorParams();
endEffectorParams.xBias = 71.6f;
endEffectorParams.yBias = 0;
endEffectorParams.zBias = 0;
DobotDll.instance.SetEndEffectorParams(endEffectorParams, false, ib);
```

(3) Set the speed and acceleration of joint coordinate axis when jogging.

Program 1.6 Set the speed and acceleration of joint coordinate axis

```
JOGJointParams jogJointParams = new JOGJointParams();
for(int i = 0; i < 4; i++) {
    jogJointParams.velocity[i] = 200;
    jogJointParams.acceleration[i] = 200;
}
DobotDll.instance.SetJOGJointParams(jogJointParams, false, ib);
```

(4) Set the speed and acceleration of Cartesian coordinate axis when jogging.

Program 1.7 Set the speed and acceleration of Cartesian coordinate axis

```
JOGCoordinateParams jogCoordinateParams = new JOGCoordinateParams();
for(int i = 0; i < 4; i++) {
    jogCoordinateParams.velocity[i] = 200;
    jogCoordinateParams.acceleration[i] = 200;
}
DobotDll.instance.SetJOGCoordinateParams(jogCoordinateParams, false, ib);
```

(5) Set the speed ratio and acceleration ratio when playback. The default value is 50%. If not set, the default value will be used.

Program 1.8 Set the speed ratio and acceleration ratio when playback

```
JOGCommonParams jogCommonParams = new JOGCommonParams();
jogCommonParams.velocityRatio = 50;
jogCommonParams.accelerationRatio = 50;
DobotDll.instance.SetJOGCommonParams(jogCommonParams, false, ib);
```


(6) Set the speed and acceleration of joint coordinate axis when playback.

Program 1.9 Set the speed and acceleration of joint coordinate axis when playback

```
PTPJointParams ptpJointParams = new PTPJointParams();
for(int i = 0; i < 4; i++) {
    ptpJointParams.velocity[i] = 200;
    ptpJointParams.acceleration[i] = 200;
}
DobotDll.instance.SetPTPJointParams(ptpJointParams, false, ib);
```

(7) Set the speed and acceleration of Cartesian coordinate axis when playback.

Program 1.10 Set the speed and acceleration of Cartesian coordinate axis when playback

```
PTPCoordinateParams ptpCoordinateParams = new PTPCoordinateParams();
ptpCoordinateParams.xyzVelocity = 200;
ptpCoordinateParams.xyzAcceleration = 200;
ptpCoordinateParams.rVelocity = 200;
ptpCoordinateParams.rAcceleration = 200;
DobotDll.instance.SetPTPCoordinateParams(ptpCoordinateParams, false, ib);
```

(8) Set the lifting height and the maximum lifting height in JUMP mode.

Program 1.11 Set the lifting height and the maximum lifting height in JUMP mode

```
PTPJumpParams ptpJumpParams = new PTPJumpParams();
ptpJumpParams.jumpHeight = 20;
ptpJumpParams.zLimit = 180;
DobotDll.instance.SetPTPJumpParams(ptpJumpParams, false, ib);
```

(9) Get the attitude information of Dobot Magician

Program 1.12 Get the attitude information of Dobot Magician

```
Pose pose = new Pose();
DobotDll.instance.GetPose(pose);
Msg( "joint1Angle="+pose.jointAngle[0]+"    "
    + "joint2Angle="+pose.jointAngle[1]+"    "
    + "joint3Angle="+pose.jointAngle[2]+"    "
    + "joint4Angle="+pose.jointAngle[3]+"    "
    + "x="+pose.x+"    "
```



```
    + "y="+pose.y+"    "
    + "z="+pose.z+"    "
    + "r="+pose.r+"    ");
```

(10) Set the starting point and the end point to make Dobot Magician move back and forth between the two points in PTP mode.

Program 1.13 Move back and forth between two points

```
while(true)
{
    try{
        PTPCmd ptpCmd = new PTPCmd();
        ptpCmd.ptpMode = 0;
        ptpCmd.x = 260;
        ptpCmd.y = 0;
        ptpCmd.z = 50;
        ptpCmd.r = 0;
        DobotDll.instance.SetPTPCmd(ptpCmd, true, ib);
        //Thread.sleep(200);
        ptpCmd.ptpMode = 0;
        ptpCmd.x = 220;
        ptpCmd.y = 0;
        ptpCmd.z = 80;
        ptpCmd.r = 0;
        DobotDll.instance.SetPTPCmd(ptpCmd, true, ib);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

1.3 MFC Demo

1.3.1 Project Description

The three function modules in Figure 1.2 indicate jogging, getting attitude information and implementing playback in PTP mode respectively.

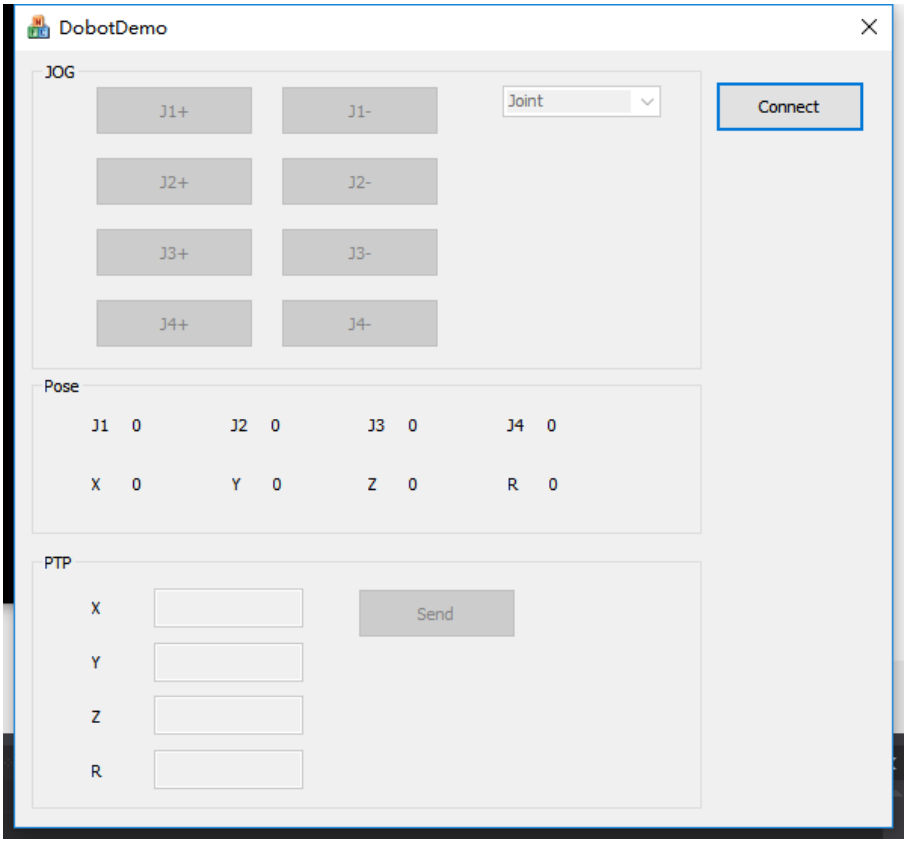


Figure 1.2 MFC demo GUI

1.3.2 Code Description

(1) Connect to Dobot Magician and check whether the connection is successful.

Program 1.14 Connect to Dobot Magician

```
if (!m_bConnectStatus) {
    if (ConnectDobot(0, 115200) != DobotConnect_NoError) {
        ::AfxMessageBox(L"Cannot connect Dobot!");
        return;
    }
```

(2) Get the serial number of Dobot Magician.

Program 1.15 Get serial number of Dobot Magician

```
char deviceSN[64];
GetDeviceSN(deviceSN, sizeof(deviceSN));
```

(3) Get the Dobot Magician name.

Program 1.16 Get the Dobot Magician name

```
char deviceName[64];
GetDeviceName(deviceName, sizeof(deviceName));
```

(4) Get the version information of Dobot Magician

Program 1.17 Get the version information of Dobot Magician

```
uint8_t majorVersion, minorVersion, revision;
GetDeviceVersion(&majorVersion, &minorVersion, &revision);
```

(5) Set the offset of the end effector.

Program 1.18 Set the offset of the end effector

```
EndEffectorParams endEffectorParams;
memset(&endEffectorParams, 0, sizeof(EndEffectorParams));
endEffectorParams.xBias = 71.6f;
SetEndEffectorParams(&endEffectorParams, false, NULL);
```

(6) Set the speed and acceleration of joint coordinate axis when jogging.

Program 1.19 Set the speed and acceleration of joint coordinate axis when jogging

```
JOGJointParams jogJointParams;
for (uint32_t i = 0; i < 4; i++) {
    jogJointParams.velocity[i] = 200;
    jogJointParams.acceleration[i] = 200;
}
SetJOGJointParams(&jogJointParams, false, NULL);
```

(7) Set the speed and acceleration of Cartesian coordinate axis when jogging.

Program 1.20 Set the speed and acceleration of Cartesian coordinate axis when jogging

```
JOGCoordinateParams jogCoordinateParams;
for (uint32_t i = 0; i < 4; i++) {
    jogCoordinateParams.velocity[i] = 200;
    jogCoordinateParams.acceleration[i] = 200;
}
SetJOGCoordinateParams(&jogCoordinateParams, false, NULL);
```

(8) Set the speed ratio and acceleration ratio when playback. The default value is 50%. If

not set, the default value will be used.

Program 1.21 Set the speed ratio and acceleration ratio when playback

```
JOGCommonParams jogCommonParams;
jogCommonParams.velocityRatio = 50;
jogCommonParams.accelerationRatio = 50;
SetJOGCommonParams(&jogCommonParams, false, NULL);
```

(9) Set the speed and acceleration of joint coordinate axis when playback.

Program 1.22 Set the speed and acceleration of joint coordinate axis when playback

```
PTPJointParams ptpJointParams;
for (uint32_t i = 0; i < 4; i++) {
    ptpJointParams.velocity[i] = 200;
    ptpJointParams.acceleration[i] = 200;
}
SetPTPJointParams(&ptpJointParams, false, NULL);
```

(10) Set the speed and acceleration of Cartesian coordinate axis when playback.

Program 1.23 Set the speed and acceleration of Cartesian coordinate axis when playback

```
PTPCoordinateParams ptpCoordinateParams;
ptpCoordinateParams.xyzVelocity = 200;
ptpCoordinateParams.xyzAcceleration = 200;
ptpCoordinateParams.rVelocity = 200;
ptpCoordinateParams.rAcceleration = 200;
SetPTPCoordinateParams(&ptpCoordinateParams, false, NULL);
```

(11) Set the lifting height and the maximum lifting height in JUMP mode.

Program 1.24 Set the lifting height and the maximum lifting height in JUMP mode

```
PTPJumpParams ptpJumpParams;
ptpJumpParams.jumpHeight = 10;
ptpJumpParams.zLimit = 150;
SetPTPJumpParams(&ptpJumpParams, false, NULL);
```

(12) Jog Dobot Magician.

Program 1.25 Jog Dobot Magician

```
JOGCmd jogCmd;
jogCmd.isJoint = m_JOGMode.GetCurSel() == 0;
jogCmd.cmd = i + 1;
SetJOGCmd(&jogCmd, false, NULL);
```

(13) Get the attitude information of Dobot Magician.

Program 1.26 Get the attitude information of Dobot Magician

```
Pose pose;
if (GetPose(&pose) != DobotCommunicate_NoError) {
    break;
}
CString str;
str.Format(L"%1.3f", pose.jointAngle[0]);
m_StaticJ1.SetWindowText(str);
str.Format(L"%1.3f", pose.jointAngle[1]);
m_StaticJ2.SetWindowText(str);
str.Format(L"%1.3f", pose.jointAngle[2]);
m_StaticJ3.SetWindowText(str);
str.Format(L"%1.3f", pose.jointAngle[3]);
m_StaticJ4.SetWindowText(str);
str.Format(L"%1.3f", pose.x);
m_StaticX.SetWindowText(str);
str.Format(L"%1.3f", pose.y);
m_StaticY.SetWindowText(str);
str.Format(L"%1.3f", pose.z);
m_StaticZ.SetWindowText(str);
str.Format(L"%1.3f", pose.r);
m_StaticR.SetWindowText(str);
```

(14) Set the starting point and the end point to make Dobot Magician move in PTP mode.

Program 1.27 Set the starting point and the end point to make Dobot Magician move

```
PTPCmd ptpCmd;
ptpCmd.ptpMode = mode;
ptpCmd.x = x;
ptpCmd.y = y;
```



```
ptpCmd.z = z;
ptpCmd.r = r;
uint64_t queuedCmdIndex;
do {
    int result = SetPTPCmd(&ptpCmd, true, &queuedCmdIndex);
    if (result == DobotCommunicate_NoError) {
        break;
    }
} while (1);
```

1.4 C# Demo

1.4.1 Project Description

The three function modules in Figure 1.3 indicate jogging, getting attitude information and implementing playback in PTP mode respectively.

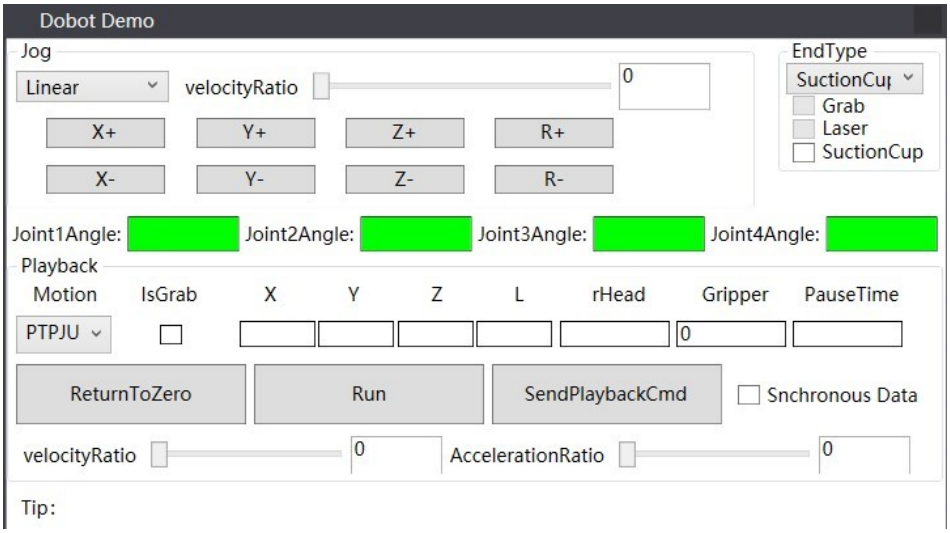


Figure 1.3 C# demo GUI

1.4.1 C# API

DobotDll.cs and DobotDllType.cs encapsulate the C type of Dobot DLL, which are C # API of Dobot Magician. The example of the connection function is shown as follows.

Program 1.28 Connection function

```
DllImport("DobotDll.dll",
                EntryPoint = "ConnectDobot",
                CallingConvention = CallingConvention.Cdecl
)]
```

```
public static extern int ConnectDobot(string portName,
                                      int baudrate,
                                      StringBuilder fwType,
                                      StringBuilder version);
```

DobotDll in the example is the DLL name in Windows OS. Please modify the DLL name according to the different OS.

1.4.2 Code Description

(1) Connect to Dobot Magician and check whether the connection is successful.

Program 1.29 Connect to Dobot Magician

```
int ret = DobotDll.ConnectDobot("", 115200, fwType, version);
if (ret != (int)DobotConnect.DobotConnect_NoError)
{
    Msg("Connect error", MsgInfoType.Error);
    return;
}
```

(2) Set the speed and acceleration of joint coordinate axis when jogging.

Program 1.30 Set the speed and acceleration of joint coordinate axis

```
JOGJointParams jsParam;
jsParam.velocity = new float[] { 200, 200, 200, 200 };
jsParam.acceleration = new float[] { 200, 200, 200, 200 };
DobotDll.SetJOGJointParams(ref jsParam, false, ref cmdIndex);
```

(3) Set the speed radio and acceleration radio when jogging.

Program 1.31 Set the speed radio and acceleration radio when jogging

```
JOGCommonParams jdParam;
jdParam.velocityRatio = 100;
jdParam.accelerationRatio = 100;
DobotDll.SetJOGCommonParams(ref jdParam, false, ref cmdIndex);
```

(4) Set the speed and acceleration of joint coordinate axis when playback.

Program 1.32 Set the speed and acceleration of joint coordinate axis when playback

```
PTPJointParams pbsParam;
pbsParam.velocity = new float[] { 200, 200, 200, 200 };
```



```
pbsParam.acceleration = new float[] { 200, 200, 200, 200 };
DobotDll.SetPTPJointParams(ref pbsParam, false, ref cmdIndex);
```

(5) Set the speed and acceleration of Cartesian coordinate axis when playback.

Program 1.33 Set the speed and acceleration of Cartesian coordinate axis when playback

```
PTPCoordinateParams cpbsParam;
cpbsParam.xyzVelocity = 100;
cpbsParam.xyzAcceleration = 100;
cpbsParam.rVelocity = 100;
cpbsParam.rAcceleration = 100;
DobotDll.SetPTPCoordinateParams(ref cpbsParam, false, ref cmdIndex);
```

(6) Set the lifting height and the maximum lifting height in JUMP mode.

Program 1.34 Set the lifting height and the maximum lifting height in JUMP mode

```
PTPJumpParams pjp;
pjp.jumpHeight = 20;
pjp.zLimit = 100;
DobotDll.SetPTPJumpParams(ref pjp, false, ref cmdIndex);
```

(7) Set the speed ratio and acceleration ratio when playback. The default value is 50%. If not set, the default value will be used.

Program 1.35 Set the speed ratio and acceleration ratio when playback

```
PTPCommonParams pbdParam;
pbdParam.velocityRatio = 30;
pbdParam.accelerationRatio = 30;
DobotDll.SetPTPCommonParams(ref pbdParam, false, ref cmdIndex);
```

(8) Jog Dobot Magician.

Program 1.36 Jog Dobot Magician

```
currentCmd.isJoint = isJoint;
currentCmd.cmd = e.ButtonState == MouseButtonState.Pressed ?
                        (byte)JogCmdType.JogAPPressed :
                        (byte)JogCmdType.JogIdle;
DobotDll.SetJOGCmd(ref currentCmd, false, ref cmdIndex);
```


(9) Get the attitude information of Dobot Magician.

Program 1.37 Get the attitude information of Dobot Magician

```
DobotDll.GetPose(ref pose);
DobotDll.GetPoseL(ref PoseL);
this.Dispatcher.BeginInvoke((Action)delegate()
{
        tbJoint1Angle.Text = pose.jointAngle[0].ToString();
        tbJoint2Angle.Text = pose.jointAngle[1].ToString();
        tbJoint3Angle.Text = pose.jointAngle[2].ToString();
        tbJoint4Angle.Text = pose.jointAngle[3].ToString();
        if (sync.IsChecked == true)
        {
            X.Text = pose.x.ToString();
            Y.Text = pose.y.ToString();
            Z.Text = pose.z.ToString();
            L.Text = PoseL.ToString();
            rHead.Text = pose.rHead.ToString();
            pauseTime.Text = "0";
          }
});
```

(10) Set the starting point and the end point to make Dobot Magician move in PTP mode.

Program 1.38 Set the starting point and the end point to make Dobot Magician move

```
pdbCmd.ptpMode = style;
pdbCmd.x = x;
pdbCmd.y = y;
pdbCmd.z = z;
pdbCmd.rHead = r;
while(true)
{
    int ret = DobotDll.SetPTPCmd(ref pdbCmd, true, ref cmdIndex);
    if (ret == 0)
        break;
}
```

(11) Get the alarm information of Dobot Magician.

Program 1.39 Get alarm information

```
int ret;
byte[] alarmsState = new byte[32];
UInt32 len = 32;
ret = DobotDll.GetAlarmsState(alarmsState,ref len,alarmsState.Length);
```

1.5 VB Demo

1.5.1 Project Description

This topic describes Dobot Magician moves from PTP1 to PTP2 in PTP mode after connecting to Dobot Magician.

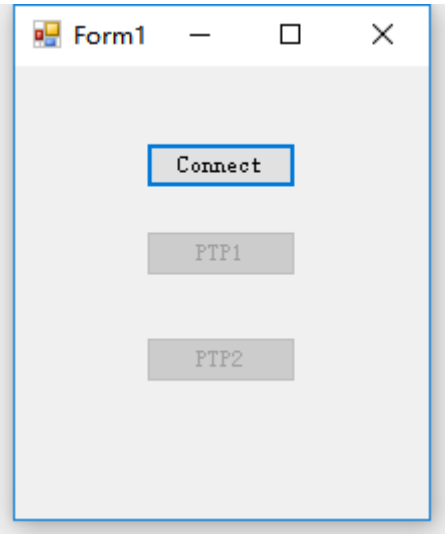


Figure 1.4 VB Demo GUI

1.5.1 VB API

DobotDll.vb and DobotDllType.vb encapsulate the C type interface of Dobot DLL, which are VB API of Dobot. The example of the connection function is shown as follows.

Program 1.40 Connection Function

```
Class DobotDll
<DllImport("DobotDll.dll", CallingConvention:=CallingConvention.Cdecl)> Public Shared Function
ConnectDobot(ByVal portName As String, ByVal baudrate As Int32) As Int32
End Function
End Class
```

DobotDll in the example is the DLL name in Windows OS. Please modify the DLL name according to the different OS.

1.5.2 Code Description

(1) Connect to Dobot Magician.

Program 1.41 Connect to Dobot Magician

```
result = DobotDll.ConnectDobot("", 115200)
If result <> 0 Then
    MsgBox("Could not find Dobot or Dobot is occupied!")
    Return
End If
```

(2) Get Dobot Magician name.

Program 1.42 Get Dobot Magician name

```
DobotDll.GetDeviceName(deviceName, 64)
```

(3) Set the starting point and the end point to make Dobot Magician move in PTP mode.

Program 1.43 Set the starting point and the end point to make Dobot Magician move

```
Dim ptpCmd As PTPCmd
ptpCmd.ptpMode = ptpMode
ptpCmd.x = x
ptpCmd.y = y
ptpCmd.z = z
ptpCmd.r = r
Dim result As Int32
Dim queuedCmdIndex As UInt64
Dim currentQueuedCmdIndex As UInt64
While True
result = DobotDll.SetPTPCmd(ptpCmd, True, queuedCmdIndex)
If result = DobotCommunicate.DobotCommunicate_NoError Then
    Exit While
End If
End While
```

(4) Get the attitude information of Dobot Magician.

Program 1.44 Get the attitude information of Dobot Magician

```
result = DobotDll.GetPose(pose)
If result <> DobotCommunicate.DobotCommunicate_NoError Then
    Return
```



```
End If
Debug.Print(pose.x)
Debug.Print(pose.y)
Debug.Print(pose.z)
Debug.Print(pose.r)
Debug.Print(pose.joint1Angle)
Debug.Print(pose.joint2Angle)
Debug.Print(pose.joint3Angle)
Debug.Print(pose.joint4Angle)
```

1.6 Qt Demo

1.6.1 Project Description

Please download **Qt5.6**. If you use **MSVC** compiler, the lib file should be loaded (Add DobotDll.lib to the directory that DobotDll.dll is stored). While if you use **MingGW** complier, this is not required.

The three function modules in Figure 1.5 indicate jogging, getting attitude information and implementing playback in PTP mode respectively.

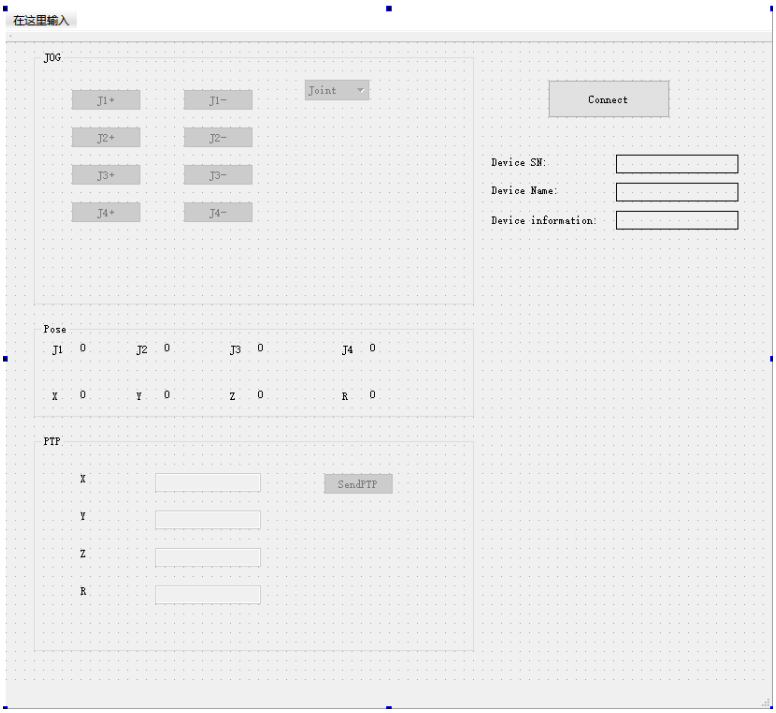


Figure 1.5 QT demo GUI

1.6.2 Code Description

(1) Connect to Dobot Magician and check whether the connection is successful.

Program 1.45 Connect to Dobot Magician

```
if (!connectStatus) {
    if (ConnectDobot(0, 115200) != DobotConnect_NoError) {
        QMessageBox::information(this, tr("error"),
                                 tr("Connect dobot error!!!"),
                                 QMessageBox::Ok);
        return;
    }
}
```

(2) Get the serial number of Dobot Magician.

Program 1.46 Get the serial number of Dobot Magician

```
char deviceSN[64];
GetDeviceSN(deviceSN, sizeof(deviceSN));
ui->deviceSNLabel->setText(deviceSN);
```

(3) Get the Dobot Magician name.

Program 1.47 Get Dobot Magician name

```
char deviceName[64];
GetDeviceName(deviceName, sizeof(deviceName));
ui->DeviceNameLabel->setText(deviceName);
```

(4) Get the version information of Dobot Magician.

Program 1.48 Get the version information of Dobot Magician

```
uint8_t majorVersion, minorVersion, revision;
GetDeviceVersion(&majorVersion, &minorVersion, &revision);
ui->DeviceInfoLabel->setText(QString::number(majorVersion) +
                             "." + QString::number(minorVersion) +
                             "." + QString::number(revision));
```

(5) Set the offset of the end effector.

Program 1.49 Set the offset of the end effector

```
EndEffectorParams endEffectorParams;
memset(&endEffectorParams, 0, sizeof(endEffectorParams));
```



```
endEffectorParams.xBias = 71.6f;
SetEndEffectorParams(&endEffectorParams, false, NULL);
```

(6) Set the speed and acceleration of joint coordinate axis when jogging.

Program 1.50 Set the speed and acceleration of joint coordinate axis when jogging

```
JOGJointParams jogJointParams;
for (int i = 0; i < 4; i++) {
    jogJointParams.velocity[i] = 100;
    jogJointParams.acceleration[i] = 100;
}
SetJOGJointParams(&jogJointParams, false, NULL);
```

(7) Set the speed and acceleration of Cartesian coordinate axis when jogging.

Program 1.51 Set the speed and acceleration of Cartesian coordinate axis when jogging

```
JOGCoordinateParams jogCoordinateParams;
for (int i = 0; i < 4; i++) {
    jogCoordinateParams.velocity[i] = 100;
    jogCoordinateParams.acceleration[i] = 100;
}
SetJOGCoordinateParams(&jogCoordinateParams, false, NULL);
```

(8) Set the speed ratio and acceleration ratio when playback. The default value is 50%. If not set, the default value will be used.

Program 1.52 Set the speed ratio and acceleration ratio when playback

```
JOGCommonParams jogCommonParams;
jogCommonParams.velocityRatio = 50;
jogCommonParams.accelerationRatio = 50;
SetJOGCommonParams(&jogCommonParams, false, NULL);
```

(9) Set the speed and acceleration of joint coordinate axis when playback.

Program 1.53 Set the speed and acceleration of joint coordinate axis when playback

```
for (int i = 0; i < 4; i++) {
    ptpJointParams.velocity[i] = 100;
    ptpJointParams.acceleration[i] = 100;
}
```



```
SetPTPJointParams(&ptpJointParams, false, NULL);
PTPJointParams ptpJointParams;
```

(10) Set the speed and acceleration of Cartesian coordinate axis when playback.

Program 1.54 Set the speed and acceleration of Cartesian coordinate axis when playback

```
PTPCoordinateParams ptpCoordinateParams;
ptpCoordinateParams.xyzVelocity = 100;
ptpCoordinateParams.xyzAcceleration = 100;
ptpCoordinateParams.rVelocity = 100;
ptpCoordinateParams.rAcceleration = 100;
SetPTPCoordinateParams(&ptpCoordinateParams, false, NULL);
```

(11) Set the lifting height and the maximum lifting height in JUMP mode.

Program 1.55 Set the lifting height and the maximum lifting height in JUMP mode

```
PTPJumpParams ptpJumpParams;
ptpJumpParams.jumpHeight = 20;
ptpJumpParams.zLimit = 150;
SetPTPJumpParams(&ptpJumpParams, false, NULL);
```

(12) Jog Dobot Magician

Program 1.56 Jog Dobot Magician

```
JOGCmd jogCmd;
jogCmd.isJoint = ui->teachMode->currentIndex() == 0;
jogCmd.cmd = index + 1;
while (SetJOGCmd(&jogCmd, false, NULL) != DobotCommunicate_NoError)
{…}
```

(13) Get the attitude information of Dobot Magician.

Program 1.57 Get the attitude information of Dobot Magician

```
Pose pose;
while (GetPose(&pose) != DobotCommunicate_NoError) {…}
ui->joint1Label->setText(QString::number(pose.jointAngle[0]));
ui->joint2Label->setText(QString::number(pose.jointAngle[1]));
ui->joint3Label->setText(QString::number(pose.jointAngle[2]));
ui->joint4Label->setText(QString::number(pose.jointAngle[3]));
```



```
ui->xLabel->setText(QString::number(pose.x));
ui->yLabel->setText(QString::number(pose.y));
ui->zLabel->setText(QString::number(pose.z));
ui->rLabel->setText(QString::number(pose.r));
```

(14) Set the starting point and the end point to make Dobot Magician move in PTP mode.

Program 1.58 Set the starting point and the end point to make Dobot Magician move

```
PTPCmd ptpCmd;
ptpCmd.ptpMode = PTPMOVJXYZMode;
ptpCmd.x = ui->xPTPEdit->text().toFloat();
ptpCmd.y = ui->yPTPEdit->text().toFloat();
ptpCmd.z = ui->zPTPEdit->text().toFloat();
ptpCmd.r = ui->rPTPEdit->text().toFloat();
while (SetPTPCmd(&ptpCmd, true, NULL) != DobotCommunicate_NoError)
{…}
```

1.7 Multi-Control Demo

1.7.1 Project Description

The DobotDll library in this demo is exclusively used for multi-control and cannot be used in other demos.

1.7.2 Code Description

The codes of this demo are much same as that of QtDemo, but each API has one more parameter (dobotId) to comfirm the ID number of Dobot Magician that has been connected, for multi-control.

(1) Connect to Dobot Magician and DLL will return the ID number of Dobot Magician that has been connected. For subsequent operations, you need to carry the ID number to specify Dobot Magician.

Program 1.59 Connect to Dobot Magician

```
if (!connectStatus) {
    if (ConnectDobot(ui->lineEdit->text().toLatin1().data(),
                     115200, fwType, version, &dobotId) !=
                     DobotConnect_NoError)
{
        QMessageBox::information(this, tr("error"),
                                 tr("Connect dobot error!!!"),
                                 QMessageBox::Ok);
        return;
```

```
        }
}
qDebug() << "dobotId" << dobotId;
```

(2) Get the serial number of Dobot Magician.

Program 1.60 Get the serial number of Dobot Magician

```
char deviceSN[64];
GetDeviceSN(dobotId, deviceSN, sizeof(deviceSN));
ui->deviceSNLabel->setText(deviceSN);
```

(3) Get the Dobot Magician name.

Program 1.61 Get the Dobot Magician name

```
char deviceName[64];
GetDeviceName(dobotId, deviceName, sizeof(deviceName));
ui->DeviceNameLabel->setText(deviceName);
```

(4) Get the version information of Dobot Magician.

Program 1.62 Get the version information of Dobot Magician

```
uint8_t majorVersion, minorVersion, revision;
GetDeviceVersion(dobotId, &majorVersion, &minorVersion, &revision);
ui->DeviceInfoLabel->setText(QString::number(majorVersion) +
                             "." + QString::number(minorVersion) +
                             "." + QString::number(revision));
```

(5) Set the offset of the end effector.

Program 1.63 Set the offset of the end effector

```
EndEffectorParams endEffectorParams;
memset(&endEffectorParams, 0, sizeof(endEffectorParams));
endEffectorParams.xBias = 71.6f;
SetEndEffectorParams(dobotId, &endEffectorParams, false, NULL);
```

(6) Set the speed and acceleration of joint coordinate axis when jogging.

Program 1.64 Set the speed and acceleration of joint coordinate axis when jogging

```
JOGJointParams jogJointParams;
```



```
for (int i = 0; i < 4; i++) {
    jogJointParams.velocity[i] = 100;
    jogJointParams.acceleration[i] = 100;
}
SetJOGJointParams(dobotId, &jogJointParams, false, NULL);
```

(7) Set the speed and acceleration of Cartesian coordinate axis when jogging.

Program 1.65 Set the speed and acceleration of Cartesian coordinate axis when jogging

```
JOGCoordinateParams jogCoordinateParams;
for (int i = 0; i < 4; i++) {
    jogCoordinateParams.velocity[i] = 100;
    jogCoordinateParams.acceleration[i] = 100;
}
SetJOGCoordinateParams(dobotId, &jogCoordinateParams, false, NULL);
```

(8) Set the speed ratio and acceleration ratio when playback. The default value is 50%. If not set, the default value will be used.

Program 1.66 Set the speed ratio and acceleration ratio when playback

```
JOGCommonParams jogCommonParams;
jogCommonParams.velocityRatio = 50;
jogCommonParams.accelerationRatio = 50;
SetJOGCommonParams(dobotId, &jogCommonParams, false, NULL);
```

(9) Set the speed and acceleration of joint coordinate axis when playback.

Program 1.67 Set the speed and acceleration of joint coordinate axis when playback.

```
PTPJointParams ptpJointParams;
for (int i = 0; i < 4; i++) {
    ptpJointParams.velocity[i] = 100;
    ptpJointParams.acceleration[i] = 100;
}
SetPTPJointParams(dobotId, &ptpJointParams, false, NULL);
```

(10) Set the speed and acceleration of Cartesian coordinate axis when playback.

Program 1.68 Set the speed and acceleration of Cartesian coordinate axis when playback

```
PTPCoordinateParams ptpCoordinateParams;
```



```
ptpCoordinateParams.xyzVelocity = 100;
ptpCoordinateParams.xyzAcceleration = 100;
ptpCoordinateParams.rVelocity = 100;
ptpCoordinateParams.rAcceleration = 100;
SetPTPCoordinateParams(dobotId, &ptpCoordinateParams, false, NULL);
```

(11) Set the lifting height and the maximum lifting height in JUMP mode.

Program 1.69 Set the lifting height and the maximum lifting height in JUMP mode

```
PTPJumpParams ptpJumpParams;
ptpJumpParams.jumpHeight = 20;
ptpJumpParams.zLimit = 150;
SetPTPJumpParams(dobotId, &ptpJumpParams, false, NULL);
```

(12) Jog Dobot Magician.

Program 1.70 Jog Dobot Magician

```
JOGCmd jogCmd;
jogCmd.isJoint = ui->teachMode->currentIndex() == 0;
jogCmd.cmd = index + 1;
while (SetJOGCmd(dobotId, &jogCmd, false, NULL) !=
        DobotCommunicate_NoError)
{…}
```

(13) Get the attitude information of Dobot Magician.

Program 1.71 Get the attitude information of Dobot Magician

```
Pose pose;
while (GetPose(dobotId, &pose) != DobotCommunicate_NoError) {
}
ui->joint1Label->setText(QString::number(pose.jointAngle[0]));
ui->joint2Label->setText(QString::number(pose.jointAngle[1]));
ui->joint3Label->setText(QString::number(pose.jointAngle[2]));
ui->joint4Label->setText(QString::number(pose.jointAngle[3]));
ui->xLabel->setText(QString::number(pose.x));
ui->yLabel->setText(QString::number(pose.y));
ui->zLabel->setText(QString::number(pose.z));
ui->rLabel->setText(QString::number(pose.r));
```


(14) Set the starting point and the end point to make Dobot Magician move in PTP mode.

Program 1.72 Set the starting point and the end point to make Dobot Magician move

```
PTPCmd ptpCmd;
ptpCmd.ptpMode = PTPMOVJXYZMode;
ptpCmd.x = ui->xPTPEdit->text().toFloat();
ptpCmd.y = ui->yPTPEdit->text().toFloat();
ptpCmd.z = ui->zPTPEdit->text().toFloat();
ptpCmd.r = ui->rPTPEdit->text().toFloat();
SetPTPCmd(dobotId, &ptpCmd, true, NULL);
```

1.8 Python Demo

1.8.1 Project Description

There are two files in Python demo.

- **DobotControl.py**: Secondary encapsulation of Dobot API
- **DobotDllType.py**: Specific implementing file

Before running **DobotControl.py**, please add Dobot DLLs directory to the running directory of python, or add them to system environment variable.

1.8.2 Python API

DobotDllType.py encapsulates the C type interface of Dobot DLL, which is Python API of Dobot. The example for loading DLL is shown as follows.

Program 1.73 Load DLL

```
def load():
    if platform.system() == "Windows":
        return CDLL("DobotDll.dll",    RTLD_GLOBAL)
    elif platform.system() == "Darwin" :
        return CDLL("libDobotDll.dylib",    RTLD_GLOBAL)
    elif platform.system() == "Linux":
        return cdll.loadLibrary("libDobotDll.so")
```

⚠NOTICE

Please be sure to add Dobot DLLs directory to system environment variable, to ensure that DLLs are loaded correctly. For details, please see *1.1.2 Usage.*

1.8.3 Code Description

When calling APIs related to motion (PTP, Jog, etc.), queue mode is used in this demo.

(1) Load DLLs and obtain Store object (api). When Python API is called, this object will be used.

Program 1.74 Load DLL

```
api = dType.load()
```

(2) Connect to Dobot Magician and print the connecting information. After the connection is successful, the related codes will be handled.

Program 1.75 Connect to Dobot

```
state = dType.ConnectDobot(api, "", 115200)[0]
print("Connect status:",CON_STR[state])
if (state == dType.DobotConnect.DobotConnect_NoError):
    #Dobot interactive codes
dType.DisconnectDobot(api)
```

(3) Control the queue:

- Clear the queue.
- Start the queue.
- Stop the queue.

Program 1.76 Queue control

```
dType.SetQueuedCmdClear(api)
dType.SetQueuedCmdStartExec(api)
dType.SetQueuedCmdStopExec(api)
```

(4) Set the motion parameters.

Program 1.77 Set the motion parameters

```
dType.SetHOMEParams(api, 200, 200, 200, 200, isQueued = 1)
dType.SetPTPJointParams(api, 200, 200, 200, 200, 200, 200, 200, 200, isQueued = 1)
dType.SetPTPCommonParams(api, 100, 100, isQueued = 1)
```

(5) Download the PTP commands to the queue and obtain the index of the last command.

Program 1.78 PTP movement

```
for i in range(0, 5):
        if i % 2 == 0:
            offset = 50
```

```
        else:
            offset = -50
        lastIndex = dType.SetPTPCmd(api,
                                    dType.PTPMode.PTPMOVLXYZMode,
                                    200 + offset,
                                    offset,
                                    offset,
                                    offset,
                                    isQueued = 1)[0]
```

(6) Wait for the last motion command to be completed.

Program 1.79 Wait for the last command

```
while lastIndex > dType.GetQueuedCmdCurrentIndex(api)[0]:
    dType.dSleep(100)
```

2. Embedded System

For embedded system, the development is performed according to the Dobot communication protocols.

2.1 Precautions

The level signal of the external interface is 3.3V, and the maximum withstand voltage is 5V. For A/D function, the input voltage of Dobot Magician cannot be greater than 3.3V. For other functions, the input voltage of Dobot Magician cannot be greater than 5V. When using chips other than STM32 and Arduino for secondary development, please notice the level capability.

2.2 STM32 Demo

2.2.1 Hardware Description

This demo is developed based on **STM32F103VET6** chip. Please prepare a **STM32F103VET6** development board when using this demo. If you use other kinds of STM32 chips, you need to migrate this demo.

The communication port of Dobot Magician is an extension 10P interface, of which the type is **FC-10P**. Figure 2.1 shows the definition of the interface. The **RX**, **TX**, **GND** pins in this interface need to be used. Figure 2.2 shows the connection between Dobot Magician and the development board: **RX->TX1**, **TX->RX1**, **GND->GND**.

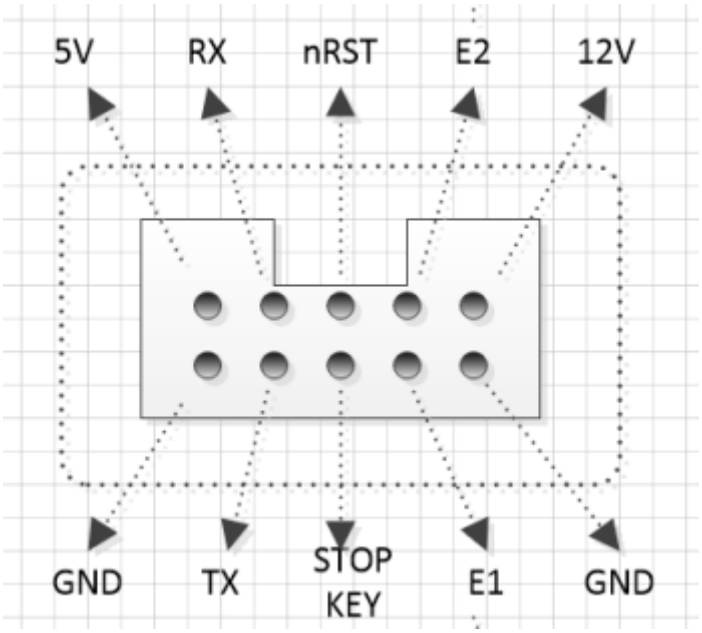


Figure 2.1 The definition of the external interface

Figure 2.2 The connection between Dobot Magician and the development board

2.2.2 Project Description

The compiler used in this demo is KEIL (4 or 5) and the version of DFP is 2.0.

（1）Communication protocol

This topic is just a brief description. The details of the communication protocols are shown in *Dobot Magician Communication Protocol*.

Data packet sent and received includes the following contents, as listed in Table 2.1.

- Header: Two packet headers
- Parameter length: The length is 2+N
- Command number ID
- Ctrl bits: include **RW** and **isQueued**
- Params: Command parameters
- Checksum

Table 2.1 Format of Communication protocol

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0XAA 0XAA	2+N	XX	1/0	1/0	N(Byte)	Payload Checksum

- Queue command: Dobot controller receives the queue instruction, the command is pressed into the controller internal instruction queue. Dobot controller will execute instructions in the order in which the instruction was pushed into the queue.
- Immediate Command: Dobot controller will process the command once received regardless of whether there is the rest commands processing or not in the current controller.

(2) File structure

The project includes **APP**, **driver**, **CORE**, **STLIB**, **STM32F10X**, and **ComPlatform** files.

- **APP**: The commands and main function are stored in **APP** directory, which are the main files used.
- **driver**: The hardware-driver files are stored in **driver** directory, which are used for port and clock configuration of chip.
- **CORE**: The core files of **M3** are stored in **CORE** directory without modification.
- **STLIB** and **STM32F10X**: The lib files are stored in **STLIB** and **STM32F10X** directories without modification.
- **ComPlatform**: The files related to protocols are stored in **ComPlatform** directory without modification.

2.2.3 Code Description

(1) ProtocolProcess function description

The sending commands and receiving commands are stored in **Ringbuffer** and processed by the ProtocolProcess function.

(2) Commands parsing

main.cpp is main-function file, **command.app** is command-handling file, which are the main files used. Let's take the PTP commands for example, the three parameters PTPCmd structure, queue tag, and index (reserved, which is used for recording the number of the current command) should be passed in the **SetPTPCmd** function.

Program 2.1 SetPTPCmd interface

```
int SetPTPCmd(PTPCmd *ptpCmd, bool isQueued,
                        uint64_t *queuedCmdIndex)
{
    Message tempMessage;
    memset(&tempMessage, 0, sizeof(Message));
    tempMessage.id = ProtocolPTPCmd;
    tempMessage.rw = true;
    tempMessage.isQueued = isQueued;
    tempMessage.paramsLen = sizeof(PTPCmd);
    memcpy(tempMessage.params, (uint8_t *)ptpCmd,
                tempMessage.paramsLen);
    MessageWrite(&gUART4ProtocolHandler, &tempMessage);
    (*queuedCmdIndex)++;
    return true;
}
```

According to Table 2.1, the input data in Program 2.1 should be the **id**, **rw**, **isQueued**, **params**

and **length** parameters of **Payload**.

Now we have provided 13 commands for completing basic motion control. If you need to implement more advanced functionality, please see *Dobot Magician Communication Protocol*.

(3) Commands sending and receiving

In protocol file, the program will check whether the sending buffer is empty. If not, the program will enable the sending interrupt of UART 4 and then send commands by the interrupt routine of UART 4. The receiving mode of UART 4 is receiving interrupt and the data received will be stored in the receiving buffer. The data in the buffer will be read by **MessageRead(ProtocolHandler *protocolHandler, Message *message)**, which will be stored in the variable of the Message structure.

Program 2.2 Message structure

```
typedef struct tagMessage {
    uint8_t id;
    uint8_t rw;
    uint8_t isQueued;
    uint8_t paramsLen;
    uint8_t params[MAX_PAYLOAD_SIZE - 2];
}Meassage;
```

(4) main function

main.cpp in this demo realizes the function that Dobot Magician move back and forth between two points. If you need to modify the two points, please modify the coordinate parameter in the structure **gPTPCmd**. If you need to implement more advanced functionality, please see *Dobot Magician Communication Protocol*.

Program 2.3 The main functions

```
int main(void)
{
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);
    SystickInit();              //Initialize clock
    Uart1Init(115200);      // Initialize UART1, and the baud rate is 115200
    Uart4Init(115200);      // Initialize UART4, and the baud rate is 115200
    InitRAM();                      // Initialize motion parameters
    ProtocolInit();             // Initialize protocol
    // Configure the motion parameters in Cartesian coordinate system
    SetPTPCoordinateParams(&gPTPCoordinateParams,true,&gQueuedCmdIndex);
    // Configure the speed radio
    SetPTPCommonParams(&gPTPCommonParams,
```

```
                                        true,
                                        &gQueuedCmdIndex);
    printf("\r\n======Enetr demo application======\r\n");
    for(; ;)
    {
        static uint32_t timer = gSystick;
        static uint32_t count = 0;
        if(gSystick - timer > 3000)              //Delay 3s
        {
            timer = gSystick;
            count++;
            if(count & 0x01)
            {
                // Set the X coordinate
                gPTPCmd.x += 100;
                // Set PTP motion, and the coordinate is the coordinate in gPTPCmd structure
                SetPTPCmd(&gPTPCmd,
                                    true,
                                    &gQueuedCmdIndex);
            }
            else
            {
                / Set the X coordinate
                gPTPCmd.x -= 100;
                // Set PTP motion, and the coordinate is the coordinate in gPTPCmd structure
                SetPTPCmd(&gPTPCmd,true,&gQueuedCmdIndex);
            }
        }
        ProtocolProcess();The
    }
}
```

2.3 Arduino Demo

2.3.1 Hardware Description

This demo is developed based on **ArduinoMega2560** chip. Please prepare an **ArduinoMega2560** development board when using this demo. If you use other kinds of Arduino chips, you need to migrate this demo.

The communication port of Dobot Magician is an extension 10P interface, of which the type is **FC-10P**. Figure 2.3 shows the definition of the interface. The **RX**, **TX**, **GND** pins in this interface need to be used. Figure 2.4 shows the connection between Dobot Magician and the development board: **RX->TX1**, **TX->RX1**, **GND->GND**.

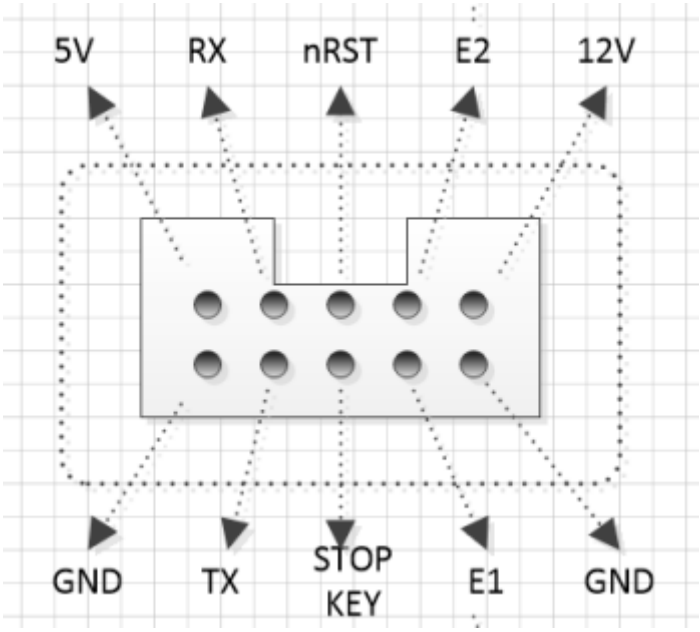


Figure 2.3 The definition of the external interface

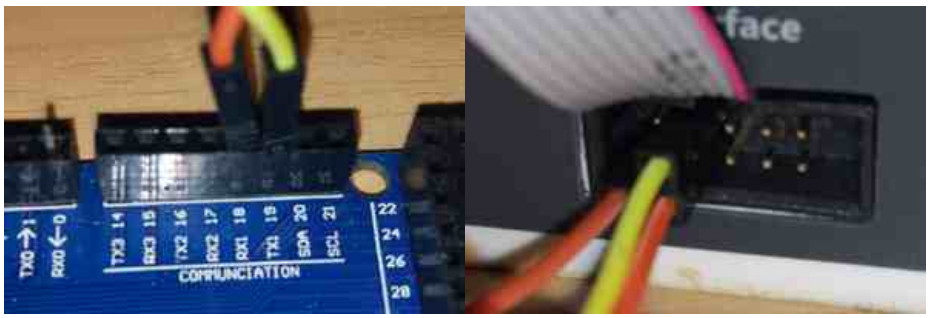


Figure 2.4 The connection between Dobot Magician and the development board

2.3.2 Project Description

The compiler of this project is **Arduino 1.8.1**.

(1) Communication protocol

This topic is just a brief description. The details of the communication protocols are shown in *Dobot Magician Communication Protocol*.

Data packet per frame includes the following contents, as listed in Table 2.2.

- Header: Two packet headers
- Parameter length: The length is 2+N

- Command number ID
- Ctrl bits: include **RW** and **isQueued**
- Params: Command parameters
- Checksum

Table 2.2 Format of Communication protocol

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0XAA 0XAA	2+N	XX	1/0	1/0	N(Byte)	Payload Checksum

- Queue command: Dobot controller receives the queue instruction, the command is pressed into the controller internal instruction queue. Dobot controller will execute instructions in the order in which the instruction was pushed into the queue.
- Immediate Command: Dobot controller will process the command once received regardless of whether there is the rest commands processing or not in the current controller.

(2) File Structure

The project files contains the following contents.

- Protocol layer processing files: **Protocol**, **Message** and **Packet** files.
- Application files: **Command** and **DobotDemo** files
- **FexTimer2** files are the driver library of Arduino for implementing the timer function.

2.3.3 Code Description

(1) ProtocolProcess function description

The sending commands and receiving commands are stored in **Ringbuffer** and processed by the ProtocolProcess function.

(2) Commands parsing

DobotDemo.ino is main-function file, **command.app** is command-handling file, which are the main files used. Let's take the PTP commands for example, the three parameters PTPCmd structure, queue tag, and index (reserved, which is used for recording the number of the current command) should be passed in the **SetPTPCmd** function.

Program 2.4 SetPTPCmd interface

```
int SetPTPCmd(PTPCmd *ptpCmd, bool isQueued, uint64_t *queuedCmdIndex)
{
    Message tempMessage;
```

```
    memset(&tempMessage, 0, sizeof(Message));
    tempMessage.id = ProtocolPTPCmd;
    tempMessage.rw = true;
    tempMessage.isQueued = isQueued;
    tempMessage.paramsLen = sizeof(PTPCmd);
    memcpy(tempMessage.params, (uint8_t *)ptpCmd, tempMessage.paramsLen);
    MessageWrite(&gUART4ProtocolHandler, &tempMessage);
    (*queuedCmdIndex)++;
    return true;
}
```

According to Table 2.2 the input data in Program 2.5 should be should be the **id**, **rw**, **isQueued**, **params** and **length** parameters of **Payload**.

Now we have provided 13 commands for completing basic motion control. If you need to implement more advanced functionality, please see *Dobot Magician Communication Protocol*.

（3）Commands sending and receiving

In protocol file, the program will check whether the sending buffer is empty. If not, the program will enable the sending interrupt of UART 1 and then send commands by the interrupt routine of UART 1. The receiving mode of UART 1 is receiving interrupt and the data received will be stored in the receiving buffer. The data in the buffer will be read by **MessageRead(ProtocolHandler *protocolHandler, Message *message)**, which will be stored in the variable of the Message structure.

Program 2.5 Message Structure

```
typedef struct tagMessage {
    uint8_t id;
    uint8_t rw;
    uint8_t isQueued;
    uint8_t paramsLen;
    uint8_t params[MAX_PAYLOAD_SIZE - 2];
}Meassage;
```

(4) Configuring function description

1. Initial Setup function.

Program 2.6 setup function

```
void setup() {
    Serial.begin(115200);                    // Start UART 0, the baud rate is 115200
    Serial1.begin(115200);                   // Start UART 1, the baud rate is 115200
```

```
    printf_begin();                          // Configure Printf, and output to UART 0 directionally
    //Set Timer Interrupt
    FlexiTimer2::set(100,Serialread);        // Configure timer interrupt and perform Serialread function every
100ms
    FlexiTimer2::start();                    // Start timer
}
```

2. Read the data in UART 1 and store in the receiving buffer.

sProgram 2.7 Serialread function

```
void Serialread()
{
  while(Serial1.available()) {                    // Check whether there is any data in UART1
        uint8_t data = Serial1.read();            // Read data
        if (RingBufferIsFull(
                                    &gSerialProtocolHandler.rxRawByteQueue)
                                    == false) {
              // If there is free space in RingBuffer, the data will be saved
              RingBufferEnqueue( &gSerialProtocolHandler.rxRawByteQueue, &data);
        }
  }
}
```

3. The Serial_putc(char c, struct __file *) and printf_begin(void) functions implement printing function.
4. The InitRAM(void) function is used for configuring motion parameters.

(5) 主循环函数 Loop function

The loop function in this demo realizes the function that Dobot Magician move back and forth between two points. If you need to modify the two points, please modify the coordinate parameter in the structure **gPTPCmd**. If you need to implement more advanced functionality, please see *Dobot Magician Communication Protocol*.

Program 2.8 Loop function

```
i void loop()
{
    InitRAM();

    ProtocolInit();
```

```
SetJOGJointParams(&gJOGJointParams, true, &gQueuedCmdIndex);

SetJOGCoordinateParams(&gJOGCoordinateParams, true, &gQueuedCmdIndex);

SetJOGCommonParams(&gJOGCommonParams, true, &gQueuedCmdIndex);

printf("\r\n=======Enter demo application=======\r\n");

SetPTPCmd(&gPTPCmd, true, &gQueuedCmdIndex);
for(; ;)
{
    static uint32_t timer = millis();
    static uint32_t count = 0;
    #ifdef JOG_STICK
    if(millis() - timer > 1000)
    {
        timer = millis();
        count++;
        switch(count){
            case 1:
                gJOGCmd.cmd = AP_DOWN;
                gJOGCmd.isJoint = JOINT_MODEL;
                SetJOGCmd(&gJOGCmd, true, &gQueuedCmdIndex);
                break;
            case 2:
                gJOGCmd.cmd = IDEL;
                gJOGCmd.isJoint = JOINT_MODEL;
                SetJOGCmd(&gJOGCmd, true, &gQueuedCmdIndex);
                break;
            case 3:
                gJOGCmd.cmd = AN_DOWN;
                gJOGCmd.isJoint = JOINT_MODEL;
                SetJOGCmd(&gJOGCmd, true, &gQueuedCmdIndex);
                break;
            case 4:
                gJOGCmd.cmd = IDEL;
                gJOGCmd.isJoint = JOINT_MODEL;
```



```
                    SetJOGCmd(&gJOGCmd, true, &gQueuedCmdIndex);
                    break;
                default:
                    count = 0;
                    break;
            }
        }
        #else
        if(millis() - timer > 3000)
        {
            timer = millis();
            count++;
            if(count & 0x01)
            {
                gPTPCmd.x += 100;
                SetPTPCmd(&gPTPCmd, true, &gQueuedCmdIndex);
            }
            else
            {
                gPTPCmd.x -= 100;
                SetPTPCmd(&gPTPCmd, true, &gQueuedCmdIndex);
            }
        }
        #endif
        ProtocolProcess();
    }
}
```

2.4 IOS Demo

2.4.1 Project Demo

DOBOTKit.framework is a static library, you can add it to the project to use. DOBOTkit is a project example based on DOBOTKit.framework.

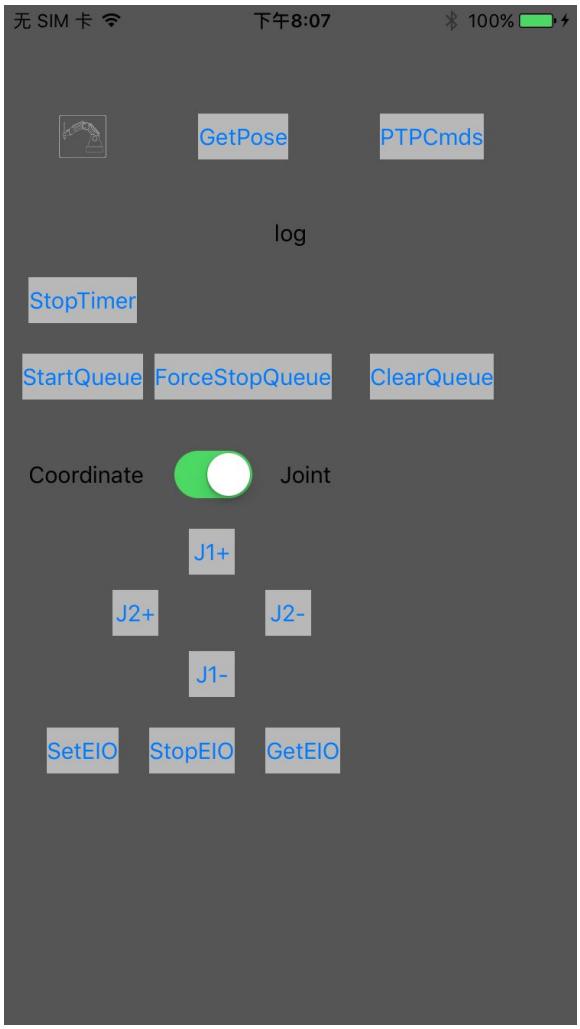


Figure 2.5 Demo GUI

2.4.2 Code Demo

This demo describes how to get the real-time pose. You can refer to this demo and *Dobot Magician Communication Protocol* for implementing other functions. The corresponding APIs have been encapsulated in the IOS static library.

(1) Initialization.

Please initialize the **BLEMsgMgr** object and consider ViewController as an agent and a message handler. **BLEMsgMgr** will handle the Bluetooth connection and the message sending and receiving.

Program 2.9 Initial BLEMsgMgr

```
[BLEMsgMgr sharedMgr].delegate = self;
[[BLEMsgMgr sharedMgr] addMsgHandler:self;
```

Add the current **ViewController** to the message handler to receive the message call-back

notification.

(2) Bluetooth connection and disconnection.

Program 2.10 Connection control

```
if ([[BLEMsgMgr sharedMgr] isConnected]) {
        // Disconnection
        [[BLEMsgMgr sharedMgr] disconnect];
    }
    else
    {
        [[BLEMsgMgr sharedMgr]
                    scanDevice:30.0f
                    mode:BLESearchMode_FindAndConnectTheFirst];
    }
```

When connecting to the Bluetooth, the app will connect to the first searched robotic arm.

(3) Real-time pose getting.

As shown in Program 2.11, please build a **Payload** object and set the corresponding parameters. Call the **sendMsg** method of **BLEMsgMgr** to download commands to Dobot Magician via Bluetooth.

Program 2.11 Download commands

```
Payload *payload = [[Payload alloc] init];
    [payload cmdGetPose];
    payload.complete = ^(MsgResult result, id msg){
        if (result == MsgResult_Ok) {
            // Parse the location information
            Payload *msgPayload = ((DobotMagicianMsg *)msg).payload;
            Pose p;
            [msgPayload.params getBytes:&p length:sizeof(p)];
            NSString *text = [NSString stringWithFormat:
                                @"Pose:x:%.0f,y:%.0f,z:%.0f,r:%.0f",
                                p.x,p.y,p.z,p.r];
            dispatch_async(dispatch_get_main_queue(), ^{
                _lblLog.text = text;
            });
        }
    };
```

```
    [[BLEMsgMgr sharedMgr] sendMsg:payload];
```

As shown in Program 2.12, when implementing **MsgHandler** protocol, ViewController will receive the response from Dobot Magician in the **handleMsg** method. You can also implement **payload.complete** to handle the returned message in the closure.

Program 2.12 Receive the returned data

```
-(void)handleMsg:(DobotMagicianMsg *)msg
{
    Payload *payload = msg.payload;
    switch ((int)payload.ID) {
        case ProtocolGetPose:{
            Pose p;
            [payload.params getBytes:&p length:sizeof(p)];
            NSString *text = [NSString stringWithFormat:
                              @"Pose:x:%.0f,y:%.0f,z:%.0f,r:%.0f",
                              p.x,p.y,p.z,p.r];
            // Record the current pose
            _lblLog.text = text;
        }
            break;
        default:
            break;
    }
}
```

2.5 Android Demo

2.5.1 Project Description

The Dobot.jar library is the encapsulating library of Dobot Magician, which encapsulates the BLE common operations in Android4.3+ platform and some Dobot Magician communication protocols. You only need to import Dobot.jar to the libs directory in the AndroidStudio (or Eclipse) project for calling encapsulated APIs, to operate DobotMagician. DobotDemo is an example that how to call APIs of the Dobot.jar library.

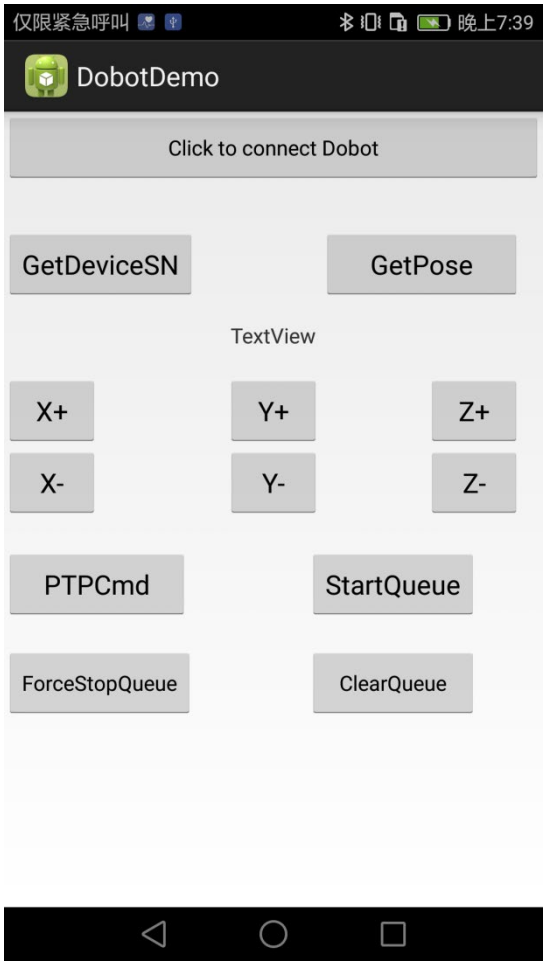


Figure 2.6 Android demo GUI

2.5.2 Code Description

This demo describes how to get the real-time pose. You can refer to this demo and *Dobot Magician Communication Protocol* for implementing other functions. The corresponding APIs have been encapsulated in the **Dobot.jar** library.

(1) Add Bluetooth permission in the **AndroidManifest.xml** file of Android project.

Program 2.13 Add Bluetooth permission

```
<uses-permission
    android:name="android.permission.BLUETOOTH"/>
<uses-permission
    android:name="android.permission.BLUETOOTH_ADMIN"/>
<uses-feature
    android:name="android.hardware.bluetooth_le"
    android:required="true" />
```

(2) Create Dobot object.

Program 2.14 Creat Dobot object

```
// Pass the Context parameters when creating Dobot object to implement DobotCalbacks() interface
Dobot    myDobot = new    Dobot(this,new DobotCallbacks() {
      @Override
      public void DobotDisconnected(BluetoothGatt arg0, BluetoothDevice arg1) {
            // TODO Auto-generated method stub
            Log.d("dobot","dobot Disconnected");
            }

      @Override
      public void DobotConnected(BluetoothGatt arg0, BluetoothDevice arg1) {
            // TODO Auto-generated method stub
            Log.d("dobot","dobot connected");
            }

      @Override
      public void DobotConnectTimeOut() {
            // TODO Auto-generated method stub
            Log.d("dobot","dobot connect timeout");
      }
});
```

(3) Initial Dobot object

Program 2.15 Initial Dobot object

```
myDobot.initialize();
```

(4) Connect mobile phone to Dobot Magician.

Program 2.16 Connect to Dobot Magician

```
myDobot.Connect(); // If disconnect to Dobot Magician, please call myDobot.close().
```

(5) Call API to get real-time pose.

Program 2.17 Get real-time pose

```
myDobot.GetPose(new DataReceiveListener() {
      @Override
```

```
    public void OnReceive() {
        // TODO Auto-generated method stub
        TagPose    pose = myDobot.ReadPose();
        float x= pose.getX();
        float y= pose.getY();
        float z= pose.getZ();
        float r= pose.getR();
        Log.d("dobot","X :"+x+"---Y :"+y+"---Z :"+z+"---R :"+r);
      }
};
```