# Dobot Magician デモ

# 説明書

Issue: V1.0.4
Date: 2018-03-08

Translated by TechShare Inc.

TechShare

**免責事項**

本マニュアルに記載されている製品（ハードウェア、ソフトウェア、ファームウェア等を含む）は、適用される法律の許容範囲内において、現状のまま提供されるため、瑕疵、過誤、或いは欠陥がある可能性があります。TechShare は、明示的黙示的を問わず、販売適合性、品質満足度、特定の目的への適合性、第三者の権利の非侵害性などの保証をいたしかねます。本マニュアル及び当社製品の使用による特別、偶発的、派生的または間接的な損害に対して一切の責任を負いかねます。

本製品を使用する前に、本マニュアル及びオンラインで公開されている関連技術資料を熟読し、ロボットアーム及びその関連知識を十分に理解していることを確認してください。技術者の指導の元で本マニュアルを使用してください。本マニュアル及び関連する指示書に従い、使用過程で損害が発生した場合でも、TechShare はセキュリティ情報の保証者とはみなされません。

使用者はロボットアームの使用に際し、いかなる重大な危険要素も発生しないように、国の規制及び関連する法律に従う責任があります。

TechShare Inc.

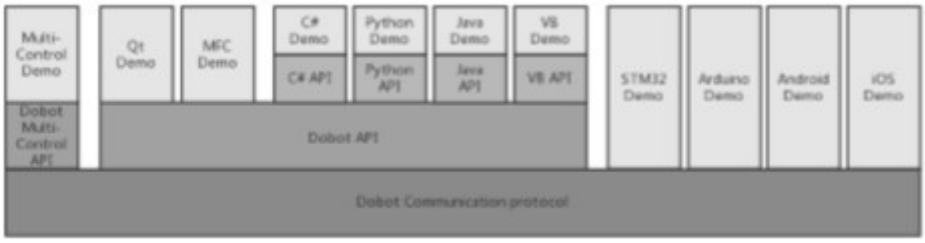
Address: TS Bldg. 5-28-6 Toyo, Koto-ku Tokyo 135-0016, Japan

Website: https://techshare.co.jp/

# 序文

## 目的

このドキュメントでは、Dobot Magician を利用したアプリケーション開発者が共通 API を理解し、開発環境をすばやく構築できるようにするため、複数の言語、フレームワーク、およびシステムの応用開発環境構築とデモコードについて説明します。




## 対象

このドキュメントは以下の読者を対象としています。

- カスタマーエンジニア
- インストレーションおよびコミッショニングエンジニア
- テクニカルサポートエンジニア

## 変更履歴

| 日付 | 変更の説明 |
|---|---|
| 2018/03/01 | 初版 |

## 記号の定義

この文書では、以下のように記号を定義します。

| 記号 | 定義 |
|---|---|
| DANGER | 高い危険性があることを示します。死亡または重傷を負う可能性があります |
| WARNING | 中・低レベルの危険性を示します。軽・中程度のけが、ロボットアームの損傷などを負う可能性があります |
| NOTICE | 潜在的な危険性を示します。ロボットアームの損傷、データの損失、または予期しない結果を招く可能性があります |
| NOTE | メインテキストの重要なポイントを強調または補足するための追加情報を提供します |

# 目次



# 1.共通システム

共通システムでは、開発者が利用できる DLL をサポートしています。Dobot Magician を制御する DLL を直接呼び出すことができるので、通信プロトコル関連の開発は不要です。

## 1.1 Dobot DLL

ソースコードとコンパイル済みファイルは、DobotDLL ディレクトリにあります。Qt 5.6 ソフトウェアを使用してソースコードを確認してください。さらに Windows 32 ビット、Windows 64 ビット、Linux、及び Mac 対応の DLL もこのディレクトリにあります。

### 1.1.1 コンパイル

お使いのシステム用の Qt バージョンをダウンロードしてインストールしてください。ダウンロードパスは https://download.qt.io/archive/qt/5.6/5.6.0/です。


DLL をコンパイルする際に Qt ライブラリを使用する場合は、MSVC コンパイラで Qt ソフトウェアを使用し、MSVC で Dobot DLL をコンパイルしてください。

### 1.1.2 使用法

・Windows OS の場合、環境変数 **Path** に DLL ディレクトリを追加してください。

・Linux OS の場合、～/ **.bash_profile** ファイルの末尾に次の文を追加し、コンピュータを再起動してください。

プログラム 1.1 Linux OS での文の追加

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

・Mac OS の場合、～/ **.bash_profile** ファイルの末尾に次の文を追加し、コンピュータを再起動してください。

プログラム 1.2 Max OS での文の追加

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

## 1.2 Java デモ

### 1.2.1 プロジェクトの説明

環境設定：Java がローカル DLL に直接アクセスできるように **jna** をインポートします。

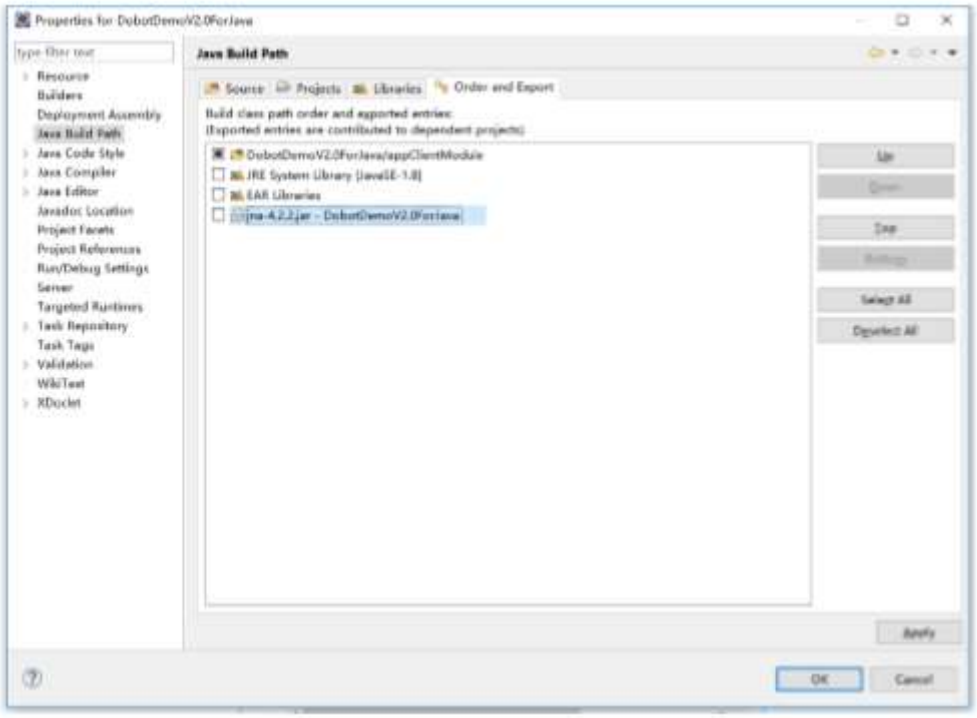


図 1.1　環境設定

### 1.2.2 Java API

**DobotDll.java** は、Dobot の Java API である Dobot DLL secondary の C 言語型のインタフェースをカプセル化します。 DLL をロードする例を以下に示します。

プログラム 1.3 DLL をロード

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

この例の **DobotDll** は、Windows OS の DLL 名です。 異なる OS に合わせて DLL 名を変更してください。

### 1.2.3 コードの説明

(1) Dobot Magician に接続し、接続が成功したかどうかを確認します。

プログラム 1.4 Dobot Magician に接続し、接続が成功したかどうかを確認

```
IntByReference ib = new IntByReference();
DobotResult ret = DobotResult.values()[
                                        DobotDll.instance.ConnectDobot(
                                        (char)0, 115200)
                                      ];
```

```
// 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）エンドエフェクタのオフセットを設定します。

プログラム 1.5 エンドエフェクタのオフセットを設定

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

（3）ジョグ動作時のジョイントの座標軸速度と加速度を設定します。

プログラム 1.6 ジョイントの座標軸速度と加速度の設定

```
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）ジョギング時のデカルト座標軸の速度と加速度を設定します。

プログラム 1.7 デカルト座標軸の速度と加速度の設定

```
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）リプレイ時の速度比と加速度比を設定します。設定しない場合は、デフォルト値の

50%が使用されます。

プログラム 1.8 リプレイ時の速度比と加速度比の設定

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

（6）リプレイ時のジョイント座標軸の速度と加速度を設定します。

プログラム 1.9 リプレイのジョイントの座標軸速度と加速度の設定

```
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）リプレイ時のデカルト座標軸の速度と加速度を設定します。

プログラム 1.10 リプレイ時のデカルト座標軸速度と加速度の設定

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

（8）JUMP モードで、持ち上げ高さと最大持ち上げ高さを設定します。

プログラム 1.11 JUMP モードでの持ち上げ高さと最大持ち上げ高さの設定

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

(9) Dobot Magician の姿勢情報を入手する

プログラム 1.12 Dobot Magician の姿勢情報の取得

```
Pose pose = new Pose();
DobotDll.instance.GetPose(pose);
```

```
Msg( "joint1Angle="+pose.jointAngle[0]+"    "
     + "joint2Angle="+pose.jointAngle[1]+"    "
                              Dobot Magician Demo Description 1 Common System
Issue V1.0 (2018-03-08) User Guide Copyright © Yuejiang Technology Co., Ltd    5
     + "joint3Angle="+pose.jointAngle[2]+"    "
     + "joint4Angle="+pose.jointAngle[3]+"    "
     + "x="+pose.x+"    "
     + "y="+pose.y+"    "
     + "z="+pose.z+"    "
+ "r="+pose.r+"    ");
```

（10）Dobot Magician が PTP モードで 2 つのポイント間を往復するように開始点と終了点を設定します。

プログラム 1.13 2 点間の往復

```
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 デモ

### 1.3.1 プロジェクトの説明

図 1.2 の 3 つの汎用モジュールは、ジョギング、姿勢情報の取得、PTP モードでの再生の実装を、それぞれ示しています。


図 1.2　MFC デモの GUI

### 1.3.2 コードの説明

(1) Dobot Magician に接続し、接続が成功したかどうかを確認します。

プログラム 1.14 Dobot Magician に接続

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

(2) Dobot Magician のシリアル番号を取得します。

プログラム 1.15 Dobot Magician のシリアル番号の取得

```
char deviceSN[64];
```

```
GetDeviceSN(deviceSN, sizeof(deviceSN));
```

（3）Dobot Magician の名前を取得します。

プログラム 1.16 Dobot Magician の名前の取得

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

（4）Dobot Magician のバージョン情報を取得します。

プログラム 1.17 Dobot Magician のバージョン情報の取得

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

（5）エンドエフェクタのオフセットを設定します。

プログラム 1.18 エンドエフェクタのオフセットの設定

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

（6）ジョグ動作時のジョイント座標軸の速度と加速度を設定します。

プログラム 1.19 ジョギング時のジョイント座標軸の速度と加速度の設定

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

（7）ジョギング時のデカルト座標軸の速度と加速度を設定します。

プログラム 1.20 ジョギング時のデカルト座標軸の速度と加速度の設定

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

（8）再生時の速度比と加速度比を設定します。設定しない場合は、デフォルト値の50%が使用されます。

プログラム 1.21 再生時の速度比と加速度比の設定

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

（9）再生時のジョイント座標軸の速度と加速度を設定します。

プログラム 1.22 再生時のジョイント座標軸の速度と加速度の設定

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

（10）再生時のデカルト座標軸の速度と加速度を設定します。

プログラム 1.23 再生時のデカルト座標軸の速度と加速度の設定

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

（11）JUMP モードで、持ち上げ高さと最大持ち上げ高さを設定します。

プログラム 1.24 JUMP モードでの持ち上げ高さと最大持ち上げ高さの設定

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

（12）Dobot Magician をジョグ動作させます。

Program 1.25　Dobot Magician のジョグ動作

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

（13）Dobot Magician の姿勢情報を取得します。

プログラム 1.26 Dobot Magician の姿勢情報の取得

```
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）Dobot Magician を PTP モードで動かすための開始点と終了点を設定します。

プログラム 1.27 Dobot Magician を動かすための開始点、終点の設定

```
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＃デモ

### 1.4.1 プロジェクトの説明

図 1.3 の 3 つの汎用モジュールは、ジョギング、姿勢情報の取得、および PTP モードでの再生の実装をそれぞれ示しています。

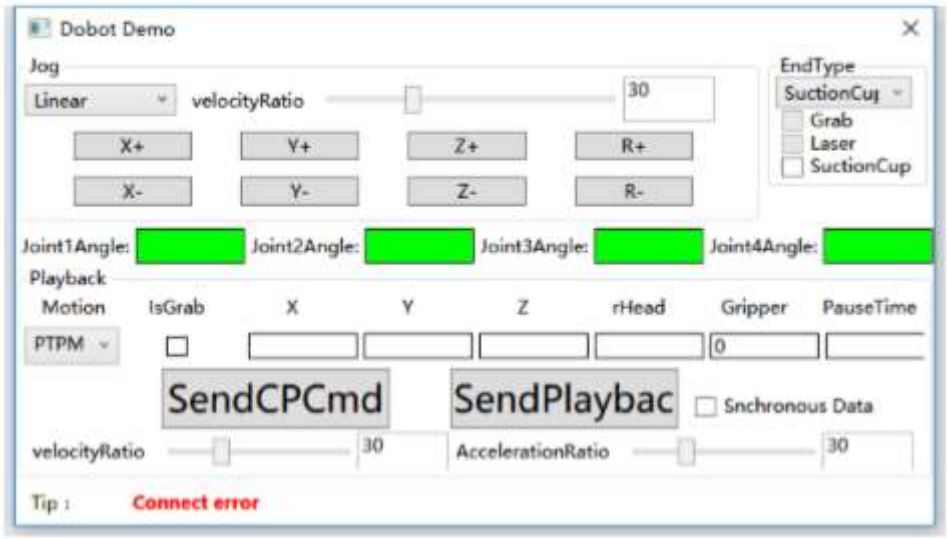



図 1.3 C＃デモ GUI

### 1.4.1 C＃API

DobotDll.cs と DobotDllType.cs は、Dobot Magician の C＃API である Dobot DLL の C 言語型の関数をカプセル化します。接続機能の例を次に示します。

プログラム 1.28 接続機能

```
DllImport（ "DobotDll.dll"、
        EntryPoint = "ConnectDobot"、
        CallingConvention = CallingConvention.Cdecl
）]
public static extern int ConnectDobot(string portName,
```

```
                                        int baudrate,
                                        StringBuilder fwType,
                                        StringBuilder version);
```

この例の **DobotDll** は、Windows OS の DLL 名です。OS に合わせて DLL 名を変更してください。

### 1.4.2 コードの説明

（1）Dobot Magician に接続し、接続が成功したかどうかを確認します。

プログラム 1.29 Dobot Magician に接続

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

（2）ジョグ動作時のジョイント座標軸の速度と加速度を設定します。

プログラム 1.30 ジョイント座標軸速度と加速度の設定

```
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）ジョグ動作時の速度比と加速度比を設定します。

プログラム 1.31 ジョグ動作時の速度比と加速度比の設定

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

（4）再生時のジョイント座標軸の速度と加速度を設定します。

プログラム 1.32 再生時のジョイント座標軸の速度と加速度の設定

```
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）再生時のデカルト座標軸の速度と加速度を設定します。

プログラム 1.33 再生時のデカルト座標軸の速度と加速のを設定

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

（6）JUMP モードでの、持ち上げ高さと最大持ち上げ高さを設定します。

プログラム 1.34 JUMP モードでの持ち上げ高さと最大持ち上げ高さの設定

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

（7）再生時の速度比と加速度比を設定します。設定しない場合は、デフォルト値の 50%が使用されます。

プログラム 1.35 再生時の速度比と加速度比を設定

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

（8）Dobot Magician をジョグ動作させます。

プログラム 1.36 Dobot Magician のジョグ動作

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

(9) Dobot Magician の姿勢情報を取得します。

プログラム 1.37 Dobot Magician の姿勢情報の取得

```
DobotDll.GetPose(ref pose);
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();
        rHead.Text = pose.rHead.ToString();
        pauseTime.Text = "0";
    }
});
```

（10）Dobot Magician を PTP モードで動かすための始点と終点を設定します。

プログラム 1.38 Dobot Magician を動かすための始点と終点の設定

```
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）Dobot Magician のアラーム情報を取得します。

プログラム 1.39 アラーム情報の取得

```
int ret;
byte[] alarmsState = new byte[32];
UInt32 len = 32;
```

```
ret = DobotDll.GetAlarmsState(alarmsState,ref len,alarmsState.Length);
```

## 1.5 VB デモ

### 1.5.1 プロジェクトの説明

このトピックでは、Dobot Magician に接続した後の PTP モードの PTP1 から PTP2 への Dobot Magician の移動について説明します。


図 1.4 VB デモ GUI

### 1.5.1 VB API

DobotDll.vb と DobotDllType.vb は、Dobot の VB API である Dobot DLL の C 言語型インターフェイスをカプセル化します。 接続機能の例を次に示します。

プログラム 1.40 接続機能 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
```

この例の **DobotDll** は、Windows OS の DLL 名です。OS に合わせて DLL 名を変更してください。

### 1.5.2 コードの説明

（1）Dobot Magician に接続します。

プログラム 1.41 Dobot Magician に接続

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

(2) Dobot Magician の名前を取得します

プログラム 1.42　Dobot Magician の名前の取得

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

(3) Dobot Magician を PTP モードで動かすために、始点と終点を設定します。

プログラム 1.43 Dobot Magician を動かすために始点と終点を設定

```
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) Dobot Magician の姿勢情報を取得する。

プログラム 1.44 Dobot Magician の姿勢情報の取得

```
result = DobotDll.GetPose(pose)
If result <> DobotCommunicate.DobotCommunicate_NoError Then
    Return
                    Dobot Magician Demo Description 1 Common System
Issue V1.0 (2018-03-08) User Guide Copyright © Yuejiang Technology Co., Ltd   16
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 デモ

### 1.6.1 プロジェクトの説明

Qt5.6 をダウンロードしてください。 MSVC コンパイラを使用する場合は、lib ファイルをロードする必要があります（DobotDll.dll が格納されているディレクトリに DobotDll.lib を追加してください）。 MingGW コンパイラを使用する場合、これは必須ではありません。 図 1.5 の 3 つの機能モジュールは、それぞれジョギング、姿勢情報の取得、PTP モードでの再生の実装を示しています。

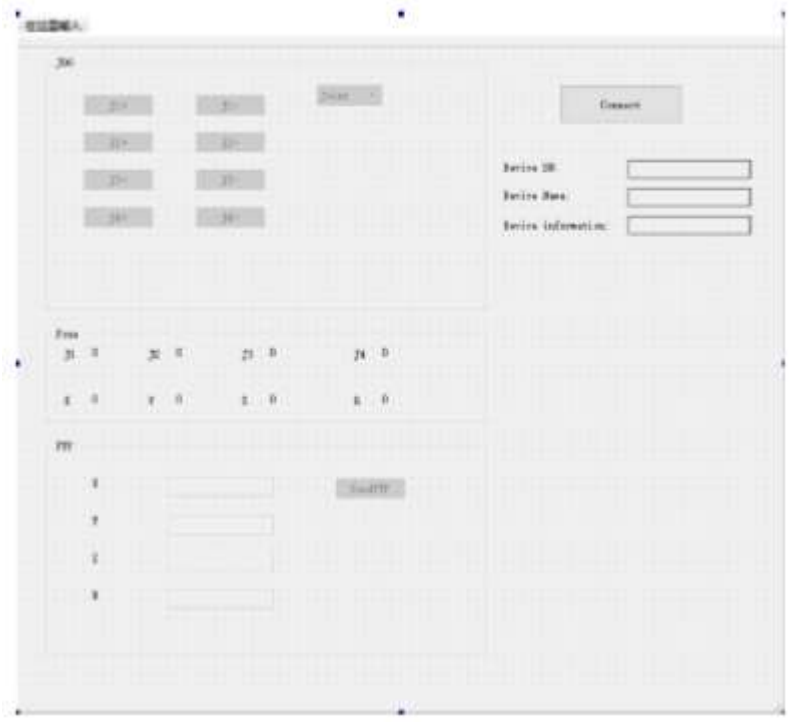


図 1.5　QT デモの GUI

### 1.6.2 コードの説明

(1) Dobot Magician に接続し、接続が成功したかどうかを確認します。

プログラム 1.45 Dobot Magician に接続

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

(2) Dobot Magician のシリアル番号を取得します。

プログラム 1.46 Dobot Magician のシリアル番号の取得

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

(3) Dobot Magician の名前を取得します。

Program 1.47　Dobot Magician の名前の取得

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

(4) Dobot Magician のバージョン情報を取得します。

Program 1.48 Dobot Magician のバージョン情報の取得

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

(5) エンドエフェクタのオフセットを設定します。

プログラム 1.49 エンドエフェクタのオフセットの設定

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

（6）ジョグ動作時のジョイント座標軸の速度と加速度を設定します。

プログラム 1.50 ジョギング時のジョイント座標軸の速度と加速度の設定

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

（7）ジョグ動作時のデカルト座標軸の速度と加速度を設定します。

プログラム 1.51 ジョグ動作時のデカルト座標軸の速度と加速度の設定

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

（8）再生時の速度比と加速度比を設定します。設定しない場合、デフォルト値の 50%が使用されます。

プログラム 1.52 再生時の速度比と加速度比の設定

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

（9）再生時のジョイント座標軸の速度と加速度を設定します。

プログラム 1.53 再生時のジョイント座標軸の速度と加速度の設定

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

（10）再生時のデカルト座標軸の速度と加速度を設定します。

プログラム 1.54 再生時デカルト座標軸の速度と加速度の設定

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

（11）JUMP モードでの、持ち上げ高さと最大持ち上げ高さを設定します。

プログラム 1.55 JUMP モードでの持ち上げ高さと最大持ち上げ高さの設定

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

（12）Dobot Magician をジョグ動作させます。

Program 1.56　Dobot Magician のジョグ動作

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

（13）Dobot Magician の姿勢情報を取得します。

プログラム 1.57 Dobot Magician の姿勢情報の取得

```
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）Dobot Magician が PTP モードで動くように始点と終点を設定します。

プログラム 1.58 Dobot Magician を動かすための始点と終点を設定

```
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 マルチコントロールデモ

### 1.7.1 プロジェクトの説明

このデモの DobotDll ライブラリは、マルチコントロール専用のため、他のデモでは使用できません。

### 1.7.2 コードの説明

このデモのコードは QtDemo のコードとほとんど同じですが、各 API には、マルチコントロールに接続されている Dobot Magician の ID 番号を確認するための、もう 1 つのパラメータ（dobotId）があります。

（1）Dobot Magician に接続すると、DLL は接続されている Dobot Magician の ID 番号を返します。 その後の操作では、その ID 番号を使って Dobot Magician を指定する必要があります。

プログラム 1.59 Dobot Magician に接続

```
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)Dobot Magician のシリアル番号を取得します。

Program 1.60　Dobot Magician のシリアル番号の取得

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

(3) Dobot Magician の名前を取得します。

Program 1.61　Dobot Magician の名前の取得

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

(4) Dobot Magician のバージョン情報を取得します。

Program 1.62　Dobot Magician のバージョン情報の取得

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

(5) エンドエフェクタのオフセットを設定します。

プログラム 1.63 エンドエフェクタのオフセットの設定

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

(6) ジョグ動作時のジョイント座標軸の速度と加速度を設定します。

プログラム 1.64 ジョグ動作時のジョイント座標軸の速度と加速度の設定

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

（7）ジョグ動作時のデカルト座標軸の速度と加速度を設定します。

プログラム 1.65 ジョギング時のデカルト座標軸の速度と加速度の設定

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

（8）再生時の速度比と加速度比を設定します。 設定しない場合、デフォルト値の 50%が使用されます。

プログラム 1.66 再生時の速度比と加速度比の設定

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

（9）再生時のジョイント座標軸の速度と加速度を設定します。

プログラム 1.67 再生時のジョイント座標軸の速度と加速度を設定

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

（10）再生時のデカルト座標軸の速度と加速度を設定します。

プログラム 1.68 再生時のデカルト座標軸の速度と加速度を設定

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

(11) JUMP モードでの、持ち上げ高さと最大持ち上げ高さを設定します。

プログラム 1.69 JUMP モードでの持ち上げ高さと最大持ち上げ高さの設定

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

(12) Dobot Magician をジョグ動作させる

Program 1.70　Dobot Magician のジョグ動作

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

(13) Dobot Magician の姿勢情報を取得する

Program 1.71　Dobot Magician の姿勢情報の取得

```
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));
                    Dobot Magician Demo Description 1 Common System
Issue V1.0 (2018-03-08) User Guide Copyright © Yuejiang Technology Co., Ltd   24
ui->zLabel->setText(QString::number(pose.z));
ui->rLabel->setText(QString::number(pose.r));
```

(14) Dobot Magician が PTP モードで動くように始点と終点を設定します。

プログラム 1.72 Dobot Magician が動作する始点と終点の設定

```
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 デモ

### 1.8.1 プロジェクトの説明

Python デモには 2 つのファイルがあります。

- **DobotControl.py**：Dobot API のカプセル化
- **DobotDllType.py**：具体的な実装ファイル

**DobotControl.py** を実行する前に、Dobot の DLL ディレクトリを python の実行ディレクトリに追加するか、システム環境変数に追加してください。

### 1.8.2 Python API

DobotDllType.py は、Dobot の Python API である Dobot DLL の C 言語型インタフェースをカプセル化します。 DLL をロードする例を以下に示します。

Program 1.73　DLL のロード

```
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")
```


Dobot DLLs ディレクトリをシステム環境変数に追加して、DLL が正しくロードされるようにしてください。 詳細は 1.1.2 使用法を参照してください。

### 1.8.3 コードの説明

モーションに関連する API（PTP、Jog など）を呼び出すときは、このデモでキューモードが使用されます。

(1) DLL をロードし、Store オブジェクト（API）を取得します。 Python API が呼び出されると、このオブジェクトが使用されます。

プログラム 1.74 DLL のロード

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

(2) Dobot Magician に接続し、接続情報を印刷します。 接続が成功すると、関連するコードが処理されます。

Program 1.75 Dobot に接続

```
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) キューを制御します：

・キューをクリアします。

・キューを開始します。

・キューを停止します。

Program 1.76 Queue control

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

(4) モーションパラメータを設定します。

プログラム 1.77 モーションパラメータを設定

```
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) PTP コマンドをキューにダウンロードし、最後のコマンドのインデックスを取得します。

プログラム 1.78 PTP 移動

```
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）最後のモーションコマンドが完了するのを待ちます。

プログラム 1.79 最後のコマンドを待つ

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

# 2.組み込みシステム

組み込みシステムの場合、開発は Dobot の通信プロトコルに従って行われます。

## 2.1 使用上の注意

外部インタフェースのレベル信号は 3.3V、最大耐圧は 5V です。 A/D 機能の場合、Dobot Magician の入力電圧は 3.3V 以下です。他の機能の場合、Dobot Magician の入力電圧は 5V 以下です。 STM32 と Arduino 以外のチップを応用開発に使用する場合は、レベル機能に注意してください。

## 2.2 STM32 デモ

### 2.2.1 ハードウェアの説明

このデモは **STM32F103VET6** チップに基づいて開発されています。このデモを使用するには、**STM32F103VET6** 開発ボードを用意してください。他の種類の STM32 チップを使用する場合は、このデモを移行する必要があります。 Dobot Magician の通信ポートは、拡張タイプの 10P インターフェイスで、タイプは FC-10P です。図 2.1 にインタフェースの定義を示します。このインタフェースの RX、TX、GND ピンを使用する必要があります。図 2.2 に、Dobot Magician と開発ボードの接続を示します。RX-> TX1、TX-> RX1、GND-> GND です。

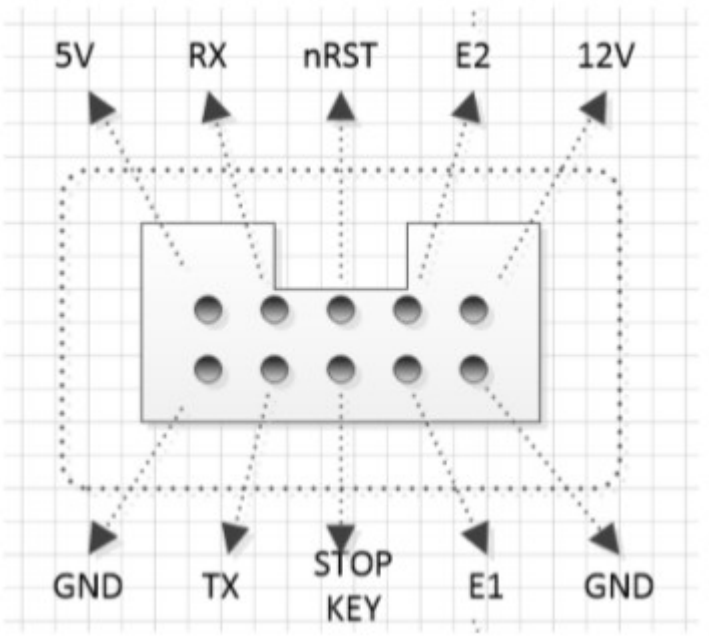



図 2.1 外部インタフェースの定義


図 2.2 Dobot Magician と開発ボードの接続

### 2.2.2 プロジェクトの説明

このデモで使用されるコンパイラは KEIL（4 又は 5）で、DFP のバージョンは 2.0 です。

(1) 通信プロトコル

このトピックでは簡単な説明を行います。通信プロトコルの詳細は、Dobot Magician Communication Protocol をご参照ください。

送受信されるデータパケットには、表 2.1 に示す以下の内容が含まれます。

•ヘッダ：2 パケットヘッダ

•パラメータ長：2 + N

•コマンド番号 ID

•制御ビット：**RW** と **isQueued** を含む

•パラメータ：コマンドパラメータ

•チェックサム

表 2.1 通信プロトコルのフォーマット

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

・キューコマンド：Dobot コントローラがキュー命令を受け取り、コマンドがコントローラの内部命令キューに入力されます。Dobot コントローラは、命令がキューにプッシュされた順序で命令を実行します。

・即時コマンド：現在のコントローラに残りのコマンド処理があるかどうかに関係なく、受信したコマンドを Dobot コントローラが処理します。

(2) ファイル構造

このプロジェクトには、**APP、ドライバ、CORE、STLIB、STM32F10X**、および **ComPlatform** ファイルが含まれています。

・**APP**：主に使用されるファイルである APP ディレクトリに、コマンドとメイン機能が格納されます。

•**ドライバ**：ハードウェアドライバファイルはドライバのディレクトリに保存され、チップのポートとクロック構成に使用されます。

・**CORE**：**M3** のコアファイルは、そのまま **CORE** ディレクトリに保存されます。

•**STLIB** と **STM32F10X**：lib ファイルは **STLIB** と **STM32F10X** ディレクトリに変更されずに格納されます。

•**ComPlatform**：プロトコル関連のファイルは、**ComPlatform** ディレクトリにそのまま保存されます。

### 2.2.3 コードの説明

(1)ProtocolProcess 関数の説明

送信コマンドと受信コマンドは **Ringbuffer** に格納され、ProtocolProcess 関数で処理されます。

(2)コマンド処理

**main.cpp** はメイン関数ファイル、**command.app** はコマンド処理ファイルであり、主に使用されるファイルです。PTP コマンドの例を見てみましょう。PTPCmd 構造体、キュータグ、およびインデックス（現在のコマンドの番号を記録するために予約済み）の 3 つのパラメータが、**SetPTPCmd** 関数に引き渡されます。

プログラム 2.1 SetPTPCmd インタフェース

```
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;
}
```

表 2.1 によると、プログラム 2.1 の入力データは、**id、rw、isQueued、params**、及び **Payload** の **length** パラメータでなければなりません。

ここまでで、基本的なモーションコントロールをマスターするための 13 のコマンドをご説明しました。より高度な機能を実装する必要がある場合は、Dobot Magician Communication Protocol をご参照ください。

(3) 送受信コマンド

プロトコルファイルでは、送信バッファが空であるかどうかをチェックします。 そうでなければ、プログラムは UART4 の送信割込みをイネーブルにし、UART4 の割込みルーチンによってコマンドを送信します。UART4 の受信モードは受信割込みであり、受信したデータは受信バッファに格納されます。バッファ内のデータは、**MessageRead（ProtocolHandler * protocolHandler、Message * message）**によって読み取られ、Message 構造体の変数に格納されます。

プログラム 2.2 メッセージ構造体

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

（4）主機能

このデモの **main.cpp** は、Dobot Magician が 2 つのポイント間を行き来する機能を実現します。この 2 つのポイントを修正する必要がある場合は、構造体 **gPTPCmd** の座標パラメータを変更してください。より高度な機能を実装する必要がある場合は、Dobot Magician Communication Protocol をご参照ください。

プログラム 2.3 主な機能

```
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 デモ

### 2.3.1 ハードウェアの説明

このデモは **ArduinoMega2560** チップをベースに開発されています。このデモを使用するには、**ArduinoMega2560** 開発ボードを用意してください。他の種類の Arduino チップを使用する場合には、このデモを移行する必要があります。

Dobot Magician の通信ポートは、拡張タイプの 10P インターフェイスで、タイプは **FC-10P** です。 図 2.3 にインタフェースの定義を示します。 このインタフェースの **RX、TX、GND** ピンを使用する必要があります。 図 2.4 に、Dobot Magician と開発ボードの接続を示します。**RX-> TX1、TX-> RX1、GND-> GND** です。

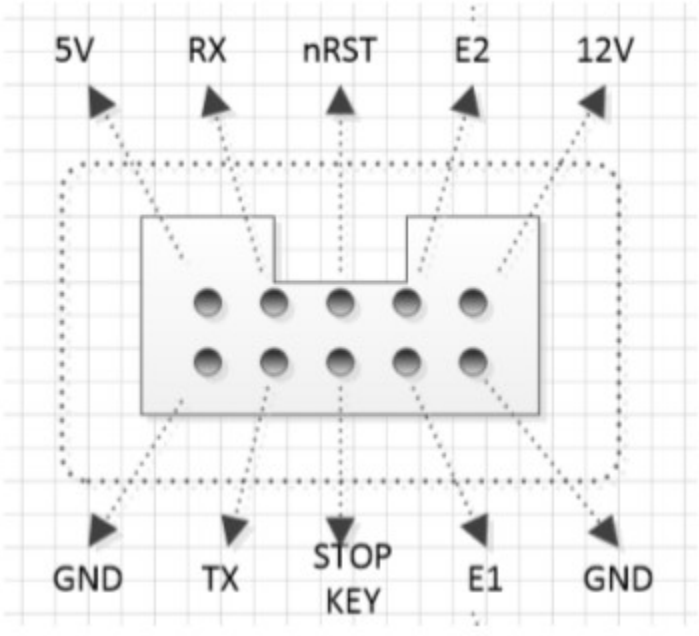



図 2.4　外部インターフェースの定義

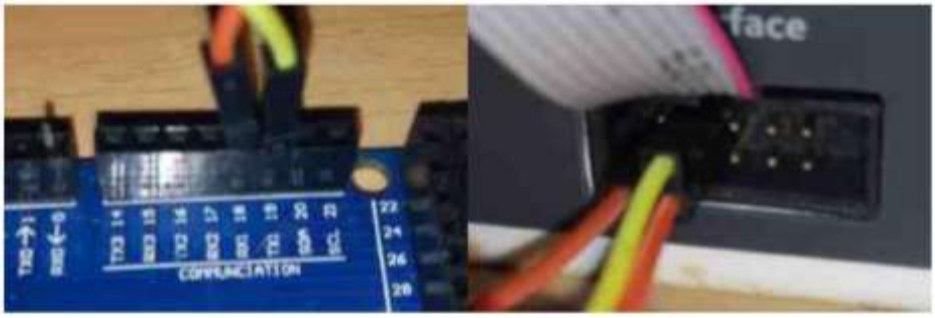



図 2.4 Dobot Magician と開発ボードの接続

### 2.3.2 プロジェクトの説明

このプロジェクトのコンパイラは **Arduino 1.8.1** です。

(1) 通信プロトコル

このトピックでは簡単な説明を行います。通信プロトコルの詳細は、Dobot Magician Communication Protocol をご参照ください。

表 2.2 に示すように、フレームあたりのデータパケットには次の内容が含まれています。

•ヘッダー：2 つのパケットヘッダー

•パラメーター長：2 + N

•コマンド番号 ID

•Ctrl ビット：**RW** と **isQueued** を含む

•パラメータ：コマンドパラメータ

•チェックサム

表 2.2 通信プロトコルのフォーマット

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

・キューコマンド：Dobot コントローラがキュー命令を受け取り、コマンドがコントローラの内部命令キューに入力されます。Dobot コントローラは、命令がキューにプッシュされた順序で命令を実行します。

・即時コマンド：現在のコントローラに残りのコマンド処理があるかどうかに関係なく、

受信したコマンドを Dobot コントローラが処理します。

（2）ファイル構造

プロジェクトファイルには、以下の内容が含まれています。

・プロトコルレイヤ処理ファイル：**プロトコル、メッセージ**、および**パケット**ファイル。

・アプリケーションファイル：**コマンド**ファイルと **DobotDemo** ファイル

・**FexTimer2** ファイルは、タイマ機能を実装する Arduino のドライバライブラリです。

### 2.3.3 コードの説明

（1）ProtocolProcess 関数の説明

送信コマンドと受信コマンドは **Ringbuffer** に格納され、ProtocolProcess 関数で処理されます。

（2）構文解析コマンド

**DobotDemo.ino** はメイン関数ファイル、**command.app** はコマンド処理ファイルで、主に使用されるファイルです。PTP コマンドの例を見てみましょう。PTPCmd 構造体、キュータグ、およびインデックス（現在のコマンドの番号を記録するために予約済み）の 3 つのパラメータが **SetPTPCmd** 関数に引き渡されます。

プログラム 2.4 SetPTPCmd インターフェース

```
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;
}
```

表 2.2 によると、プログラム 2.5 の入力データは、**id、rw、isQueued、params**、および **Payload** の **length** パラメータである必要があります。

ここまでで、基本的なモーションコントロールをマスターするための 13 のコマンドをご紹介しました。より高度な機能を実装する必要がある場合は、Dobot Magician

Communication Protocol をご参照ください。

(3)送受信コマンド

プロトコルファイルでは、送信バッファが空であるかどうかをチェックします。 そうでない場合、プログラムは UART1 の送信割り込みをイネーブルにし、UART1 の割り込みルーチンによってコマンドを送信します。UART1 の受信モードは受信割り込みであり、受信したデータは受信バッファに格納されます。バッファ内のデータは、**MessageRead（ProtocolHandler * protocolHandler、Message * message**）によって読み取られ、Message 構造体の変数に格納されます。

プログラム 2.5 メッセージ構造体

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

（4）機能説明の設定

1. 初期設定機能

プログラム 2.6 セットアップ機能

```
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. UART1 のデータを読み出し、受信バッファに格納します。

プログラム 2.7 シリアルリード機能

```
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. Serial_putc（char c, struct __file *）関数と printf_begin（void）関数は、印刷機能を実装しています。

4. InitRAM（void）関数は、モーションパラメータの設定に使用されます。

（5）主循環環境ループ関数

このデモのループ機能は、Dobot Magician が 2 つのポイント間を行き来する機能を実現します。この 2 つのポイントを修正する必要がある場合は、構造体 **gPTPCmd** の座標パラメータを変更してください。より高度な機能を実装する必要がある場合は、Dobot Magician Communication Protocol をご参照ください。

プログラム 2.8 ループ機能

```
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 デモ

### 2.4.1 プロジェクトのデモ

DOBOTKit.framework はスタティックライブラリです。プロジェクトに追加して使用することができます。DOBOTkit は DOBOTKit.framework に基づくプロジェクトの例です。


Figure 2.5 Demo GUI

2.4.2 コードデモ

このデモでは、リアルタイムポーズを取得する方法について説明します。他の機能を実装する時にも、このデモと Dobot Magician Communication Protocol をご参照いただけます。対応する API は、IOS スタティックライブラリにカプセル化されています。

（1）初期化。

**BLEMsgMgr** オブジェクトを初期化し、ViewController をエージェント兼メッセージハンドラとしてください。**BLEMsgMgr** は、Bluetooth 接続とメッセージの送受信を処理します。

プログラム 2.9 BLEMsgMgr の初期化

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

```
[[BLEMsgMgr sharedMgr] addMsgHandler:self;
```

現在の **ViewController** をメッセージハンドラに追加して、メッセージのコールバック通知を受信します。

（2）Bluetooth の接続と切断。

プログラム 2.10 接続制御

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

Bluetooth に接続すると、アプリは最初に検索されたロボットアームに接続します。

（3）リアルタイムポーズ取得

プログラム 2.11 に示すように、ペイロードオブジェクトを構築し、対応するパラメータを設定してください。Bluetooth 経由で Dobot Magician にコマンドをダウンロードするには、BLEMsgMgr の sendMsg メソッドを呼び出します。

プログラム 2.11 ダウンロードコマンド

```
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];
```

プログラム 2.12 に示すように、**MsgHandler** プロトコルを実装する場合、ViewController は **handleMsg** メソッドで Dobot Magician から応答を受信します。また、closure で返されたメッセージを処理するために **payload.complete** を実装することもできます。

プログラム 2.12 返されたデータを受信

```
-(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 デモ

### 2.5.1 プロジェクトの説明

Dobot.jar ライブラリは、Android4.3 +プラットフォームといくつかの Dobot Magician 通信プロトコルで、BLE の共通操作をカプセル化する Dobot Magician のカプセル化ライブラリです。 DobotMagician を操作するには、Dobot.jar を AndroidStudio（または Eclipse）プロジェクトの libs ディレクトリにインポートするだけで、カプセル化 API を呼

び出すことができます。DobotDemo は、Dobot.jar ライブラリの API を呼び出す方法の一例です。


図 2.6 Android デモの GUI

### 2.5.2 コードの説明

このデモでは、リアルタイムポーズを取得する方法について説明します。他の機能を実装する時にも、このデモと Dobot Magician Communication Protocol をご参照いただけます。 対応する API は **Dobot.jar** ライブラリにカプセル化されています。

(1) Android プロジェクトの **AndroidManifest.xml** ファイルに Bluetooth 権限を追加します。

プログラム 2.13 Bluetooth の権限追加

```
<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）Dobot オブジェクトを作成します。

プログラム 2.14 Dobot オブジェクトの作成

```
// 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）Dobot オブジェクトを初期化します

プログラム 2.15　Dobot オブジェクトの初期化

```
myDobot.initialize（）;
```

（4）Dobot Magician に携帯電話を接続します。

プログラム 2.16 Dobot Magician に接続

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

（5）リアルタイムのポーズを取得するために API を呼び出します。

プログラム 2.17 リアルタイムポーズの取得する

```
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);
      } };
```