Dobot Magician Demo Description (ROS)

Issue: V1.0

Date: 2018-08-30

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

Change History

Date	Change Description
2018/08/30	The first release.

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. ROS Demo

This topic is aimed at helping user to understand common API of Dobot Magician and build development environment quickly.

1.1 Environment Building

The **ROS** demo is developed based on Ubuntu Linux OS, for **ROS** can only be installed on the Linux OS. You need to install **Ubuntu** Linux OS and the matched **ROS**. The **Ubuntu** Linux OS can be installed on the local system or on a virtual machine. In this demo, the **Ubuntu 16.04 LTS** version is recommended, and the matched **ROS** version is **Kinetic**, and we install the **Ubuntu** Linux OS on the VMware virtual machine. The details how to install the Ubuntu Linux OS is not descripted in this topic.

If you have installed the **Ubuntu** Linux OS, please run the command **cat /etc/issue** in the terminal to check the version and then install the matched **ROS**. Table 1.1 shows the **Ubuntu** Linux OS version with the matched **ROS** version. For details about ROS, please see http://www.ros.org/.

Table 1.1 The **Ubuntu** Linux OS version with the matched **ROS** version

ROS Version	Ubuntu Version
Lunar	Ubuntu 17.04
Kinetic (Recommend)	**Ubuntu16.04**
Jade	Ubuntu 15.04
Indigo	Ubuntu 14.04
…	…

⚠NOTICE

If the versions of the **ROS** and the **Ubuntu** Linux OS are not matched, an error may be occurred. Please ensure that the **ROS** is matched with the **Ubuntu** Linux OS.

Step 1 Log in to **Ubuntu** Linux OS as the common user.

Step 2 Configure **Ubuntu** repository.

Before installing the **ROS**, please check the **Ubuntu** environment.

1. Select the options with the keywords **universe**, **restricted**, **multiverse** and set **Download from** to **Server for United States** on the **System Settings… > Software&Updates > Ubuntu Software** page, as shown in Figure 1.1.

Figure 1.1 Ubuntu setting

2. Modify the settings as shown in Figure 1.2 on the **System Settings…** > **Software&Updates** > **Others Software** page.

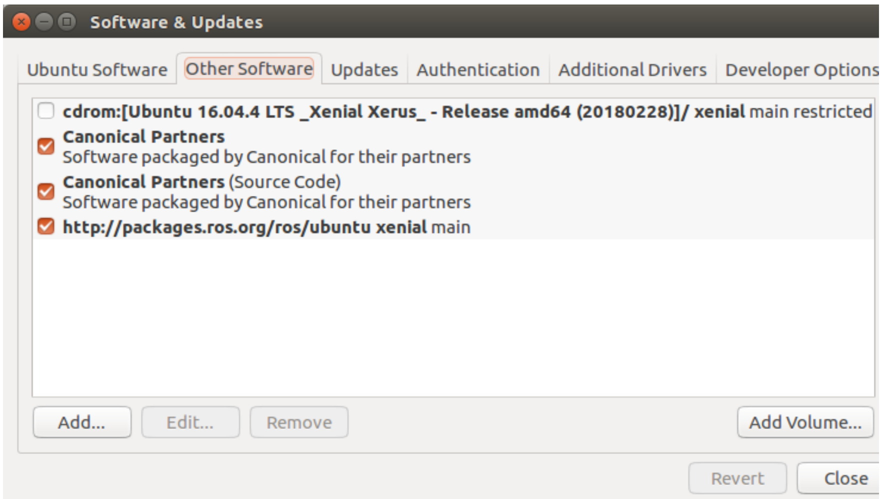

Figure 1.2 Other software setting

3. Click **Close**, and after about 30s later, the cache update will be completed.

Step 3 Configure the **apt** source of the **ROS**.

1. Add **sources.list**. Namely, add the mirror source to the system sources-list of the Ubuntu Linux OS.

The official source is recommended, for giving a fast loading time.

- Mode 1: Official source

```
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

- Mode 2: USTC source

```
$ sudo sh -c '. /etc/lsb-release && echo "deb http://mirrors.ustc.edu.cn/ros/ubuntu/ $DISTRIB_CODENAME main" > /etc/apt/sources.list.d/ros-latest.list'
```

- Mode 3: Singapore source

```
$ sudo sh -c '. /etc/lsb-release && echo "deb http://mirror-ap.packages.ros.org/ros/ubuntu/ $DISTRIB_CODENAME main" > /etc/apt/sources.list.d/ros-latest.list'
```

2. Add keys.

```
$ sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116
```

3. Update system, to make sure that the Debian package index is up-to-date.

```
$ sudo apt-get update && sudo apt-get upgrade
```

4. Install ROS package.

   ROS provides four default configurations to get your started, including Desktop-full installation, Desktop installation, ROS-base installation and individual package installation. There are many different libraries and tools in ROS. Different installation provides different libraries and tools.

   In our ROS demo, the Desktop-full installation (includes ROS, rqt, rviz, robot-generic libraries, 2D/3D simulators and 2D/3D perception) is recommended. Take **Kinetic** as an example, please run the following command in the terminal.

```
$ sudo apt-get install ros-kinetic-desktop-full
```

   If your ROS version is other version, please modify the command. For example, if your ROS version is Indigo, please run the following command in the terminal.

```
$ sudo apt-get install ros-indigo-desktop-full
```

   You can choose other installation method based on site requirements.

- esktop Installation: ROS, rqt, rviz and robot-generic libraires.

```
$ sudo apt-get install ros-kinetic-desktop
```

- ROS-Base Installation: ROS package, build tool, and communication libraries.

```
$ sudo apt-get install ros-kinetic-ros-base
```

- Individual Package Installation: Install a specific ROS package. Where, *PACKAGE* is the package name, please replace it based on site requirements.

```
$ sudo apt-get install ros-kinetic-PACKAGE
```

   For example, if you cannot find slam-gmapping when running ROS, please run the following command in the terminal.

```
$ sudo apt-get install ros-kinetic-slam-gmapping
```

   If you need to find available packages, please run the following command in the terminal.

```
$ apt-cache search ros-kinetic
```

   If the ROS version is not compatible or the mirror source is not updated, an error may be occurred when installing the ROS. Please visit http://wiki.ros.org/ROS/ to find the solution to the problem.

Step 4 Configure ROS.

1. Initialize rosdep.

```
$ sudo rosdep init && rosdep update
```

Before you can use ROS, you need to initialize rosdep. rosdep enables you to easily install system dependencies for source you want to compile and is required to run some core components in ROS.

2. Configure ROS environment.

```
#For Ubuntu 16.04
$ echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc
$ source ~/.bashrc
```

```
#For Ubuntu 14.04
$ echo "source /opt/ros/indigo/setup.bash" >> ~/.bashrc
$ source ~/.bashrc
```

3. Install rosinstall.

rosinstall is a frequently used command tool that enables you to easily download many source trees for ROS packages with one command.

```
$ sudo apt-get install python-rosinstall
```

Step 5 Check whether ROS can work normally.

Run the following command in the terminal to start ROS.

```
$ roscore
```

Figure 1.3 shows that ROS starts successfully.

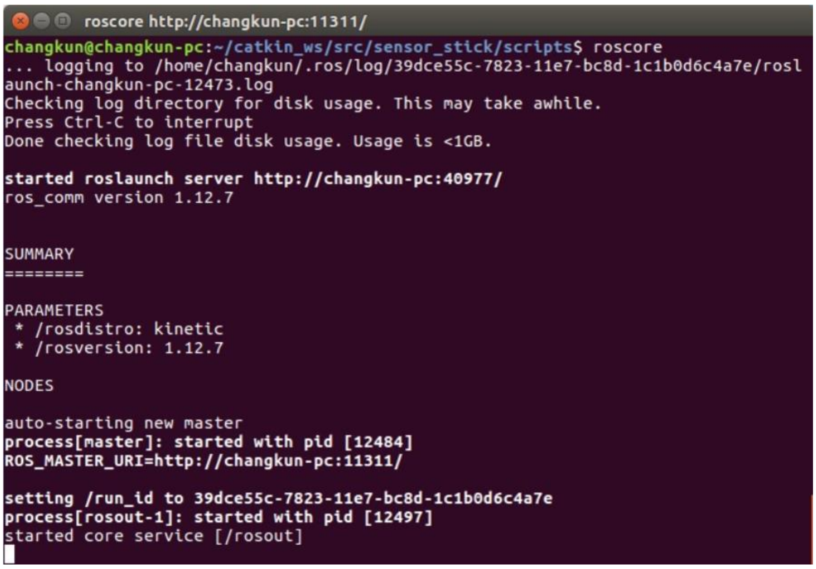

Figure 1.3 ROS starts successfully

1.2 ROS Demo Description

1.2.1 Project Description

Step 1 Decompress the obtained **dobot_ws** Demo package to the **/home** directory, as shown in Figure 1.4.

⚠NOTICE

Please decompress the Demo package on the Linux OS. Otherwise, a compile error will be occurred.

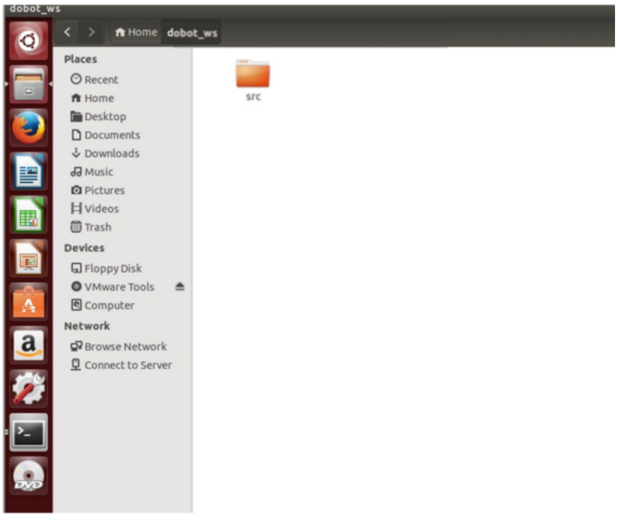

Figure 1.4 Compress dobot_ws Demo

Step 2 Open a terminal and run the following command to assign the permission to the common user for serial port operations.

```
$ sudo usermod -a -G dialout username
```

Where, *username* is the common user name, please replace it based on site requirements.

Step 3 Run the following commands to assign the permission to the **dobot_ws** directory.

```
$ cd /home/dobot-ws
$ sudo chmod 777 ./* -R
```

Step 4 Run the following commands to compile ROS demo.

```
$ cd /home/dobot-ws
$ catkin_make
```

Figure 1.5 shows that ROS demo is compiled successfully.

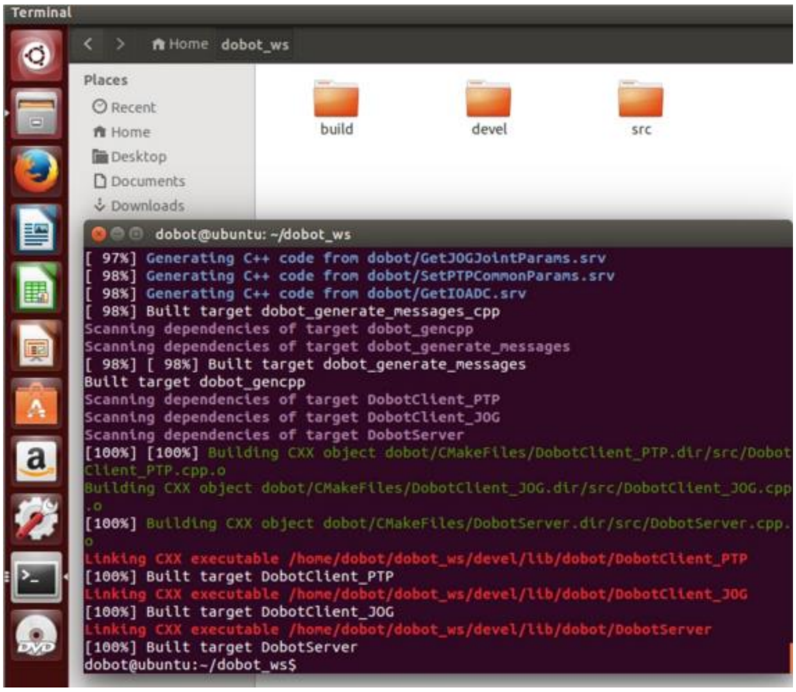

Figure 1.5 Compile completely

Step 5 Run the following commands to add **dobot_ws** environment variables.

```
$ echo "source /home/dobot_ws/devel/setup.bash " >> ~/.bashrc
$ source ~/.bashrc
```

Step 6 Start ROS.

```
$ roscore
```

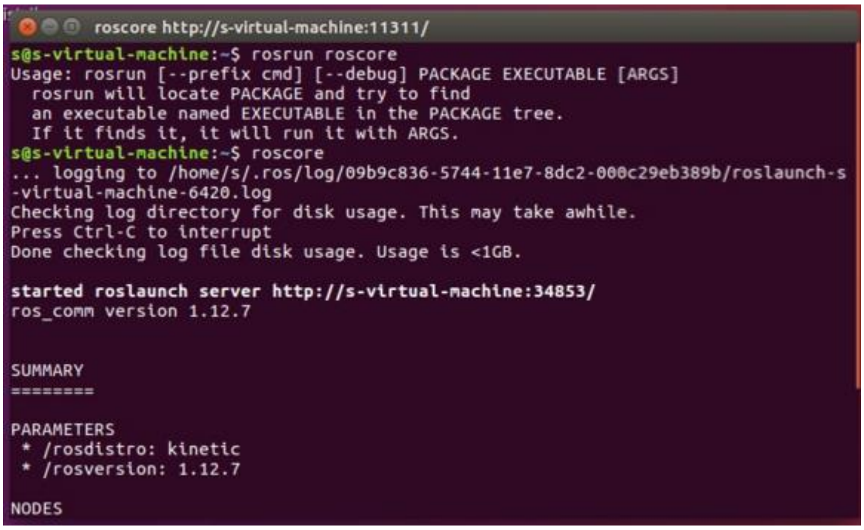

Figure 1.6 Start ROS

Step 7 Reopen a terminal as the service node and run the following command to connect to Dobot Magician.

Where, *Port* is the Dobot serial port, please run the command **ls /dev -l** to view the really serial port. The serial port is named **ttyUSB*X***. *X* indicates the serial number, please replace it based on site requirements.

```
$ rosrun dobot DobotServer Port
```

Figure 1.7 shows that Dobot Magician is connected successfully.

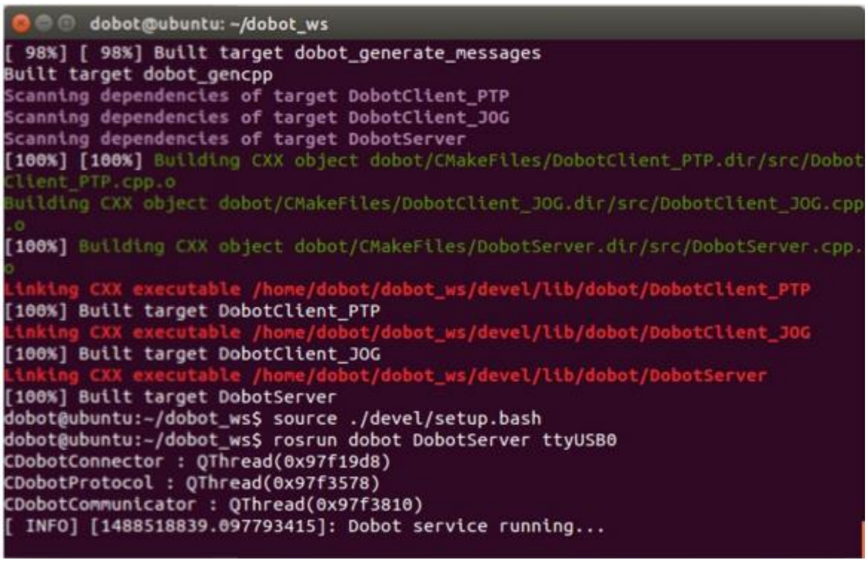

Figure 1.7 Connect Dobot Magician successfully

If the garbled code is shown when running this command, the serial port may be wrong. Please check the right serial port by running the command **ls /dev -l**.

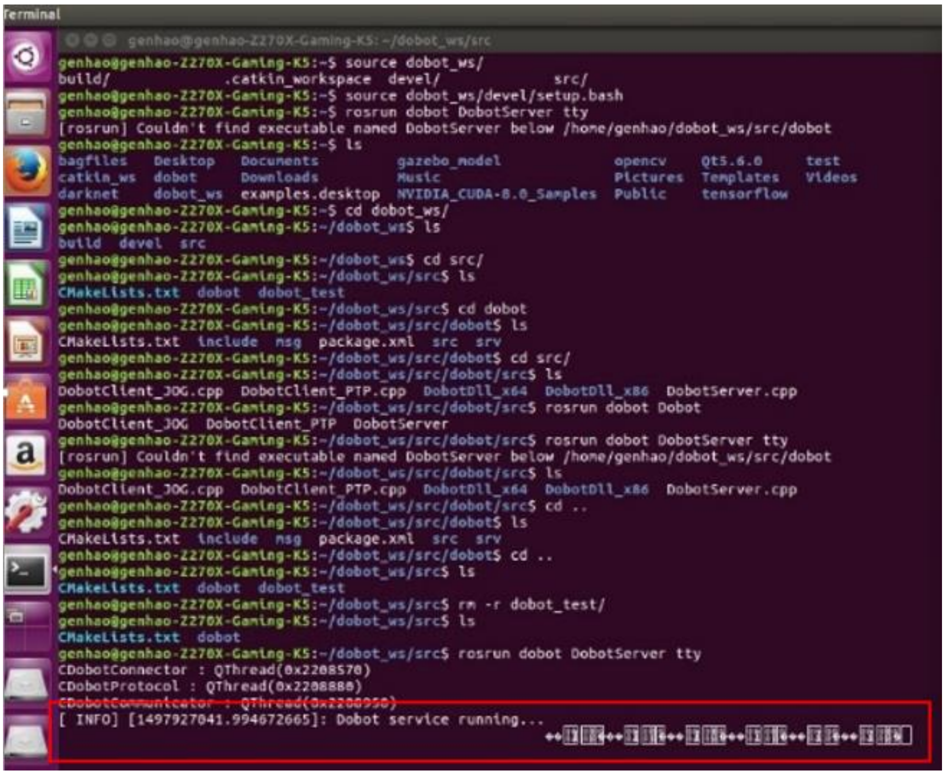

Figure 1.8 Serial port error

Step 8 After Dobot Magician is connected, please reopen another terminal as the client node and run the following command to start jogging Dobot Magician with keyboards.

```
$ rosrun dobot DobotClient_JOG
```

After running this command, you can use keyboards to control Dobot Magician, as shown in Table 1.2.

Table 1.2 Keyboard description

Keys	Direction
W	Forward
S	Backward
A	Leftward
D	Rightward
U	Upward
I	Downward
J	Rotate counterclockwise
K	Rotate clockwise
Other keys	Stop jogging

If you press **W**, **A**, **S**, **D** or other keys without reaction, the ASCII codes of the keys are inconsistent with what you set in the Demo. You can modify the ASCII codes of the keys in the **src** file, as shown in Figure 1.10.

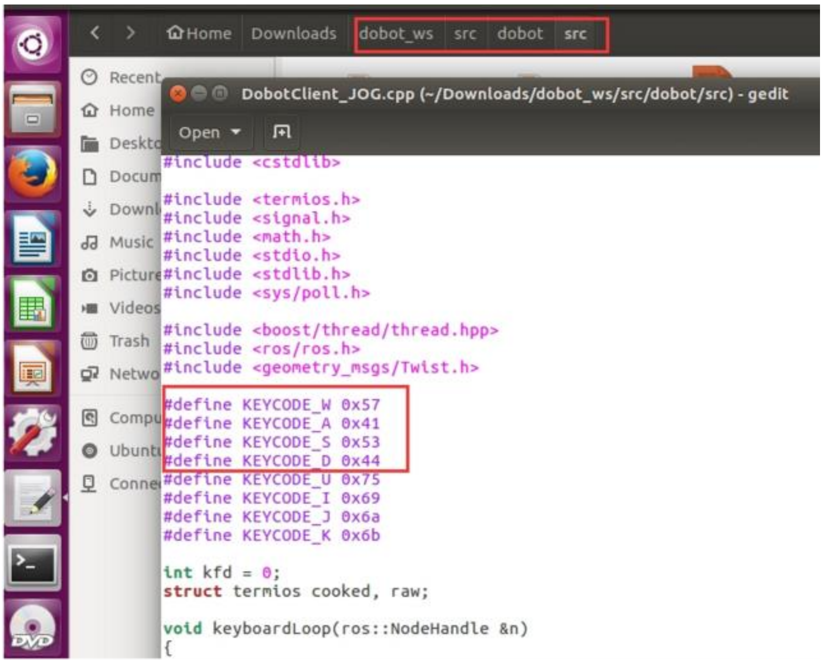

Figure 1.9 Modify the ASCII codes of the keys

Also, you can run other commands on the client node after pressing **Crtil+C** to stop jogging command. For example, run the PTP command.

```
$ rosrun dobot DobotClient_PTP
```

After running this PTP command, Dobot Magician will move automatically back and forth along X-axis.

1.2.2 Code Description

- In this demo, queue mode is used. Figure 1.10 shows the demo directory structure.

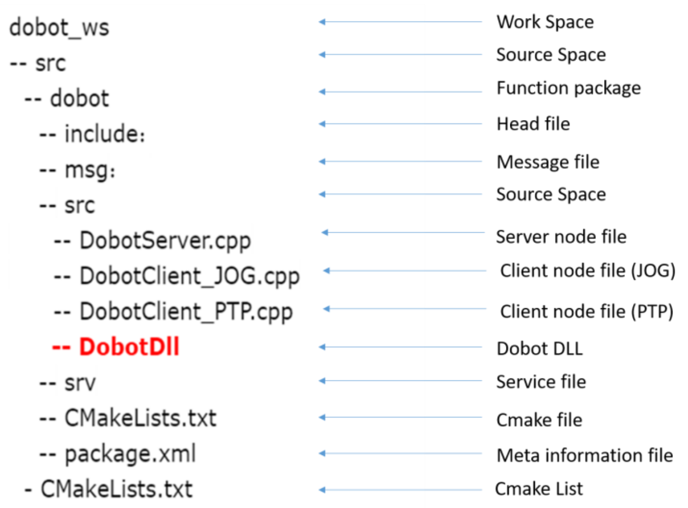

Figure 1.10 Demo directory structure

- There are four common communication styles in the ROS.
  - Topic
  - Service
  - Parameter Service
  - Actionlib

  In this ROS demo, the Service style is used. Request/Reply communication between nodes is done via Service, which is defined by a pair of messages: one for the request and one for reply, as shown in Figure 1.11.

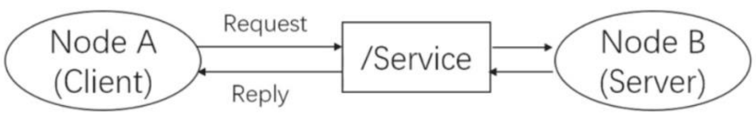

Figure 1.11 Service communication

  Where, Node B is a server, which offers a service under a string name. Node A is a client, which calls the service by sending the request message and awaiting the reply.

- Figure 1.12 shows the Service programing process.

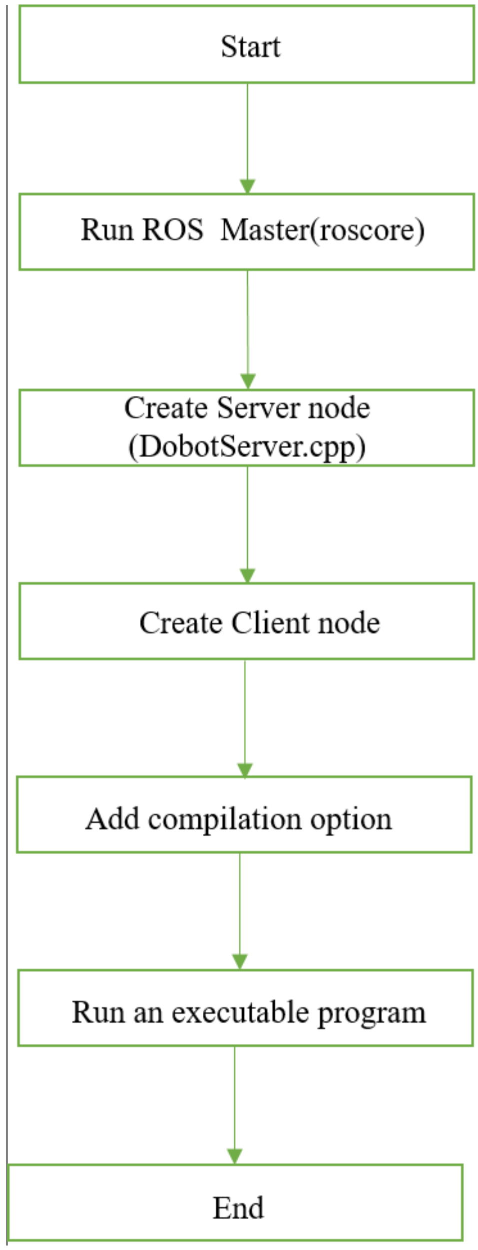

Figure 1.12 Service programing process

- Figure 1.13 shows the process of creating a server.

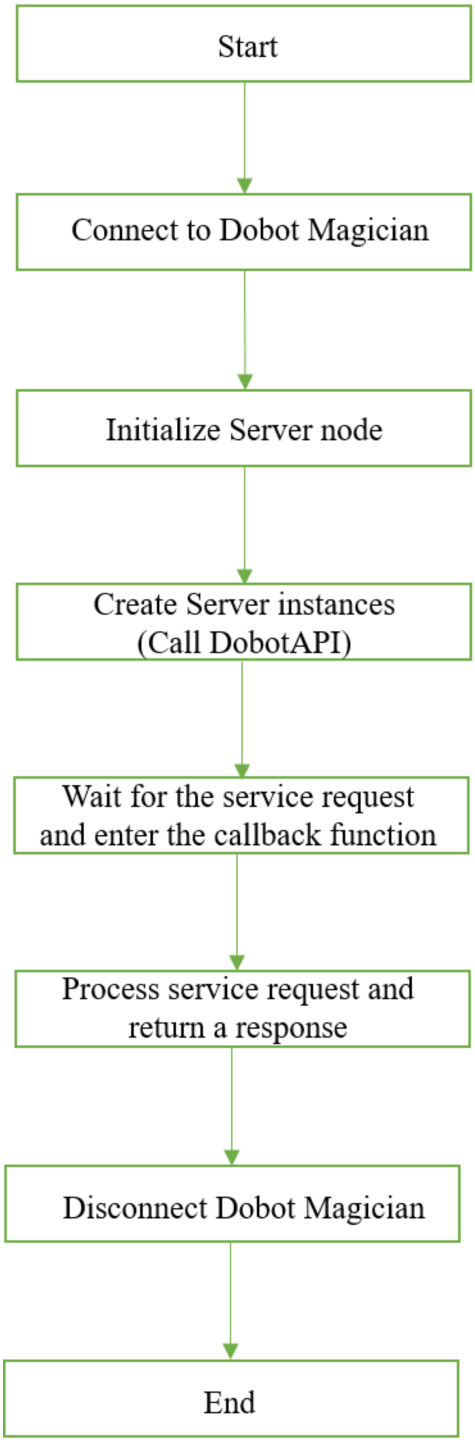

Figure 1.13 Process of create a server

（1） Load ROS and Dobot DLL.

Program 1.1 Load ROS and Dobot DLL

```
#include "ros/ros.h"     //Load ROS
#include "std_msgs/String.h“
```

```
#include "std_msgs/Float32MultiArray.h“
#include "DobotDll.h"               //Load Dobot DLL
```

(2) Connect to Dobot Magician

Program 1.2 Connect to Dobot Magician

```
if (argc < 2)
    {
        ROS_ERROR("[USAGE]Application portName");
        return -1;
    }
    printf("---------%s",argv[1]);
    //Connect to Dobot Magician
    int result = ConnectDobot(argv[1], 115200, 0, 0);
```

(3) Initialize Server node.

Program 1.3 Initialize Server node

```
ros::init(argc, argv, "DobotServer");   //Initialize Server node called DobotServer
ros::NodeHandle n;          // Create the node handle
```

(4) Create Server instances.

Program 1.4 Create Server instances

```
    //Create a Server vector
    std::vector<ros::ServiceServer> serverVec;
   //Initialize Server instances and register the callback function in the right initialization function
    InitCmdTimeoutServices(n, serverVec);
    InitDeviceInfoServices(n, serverVec);
    InitPoseServices(n, serverVec);
    InitAlarmsServices(n, serverVec);
    InitHOMEServices(n, serverVec);
    InitTRIGServices(n, serverVec);
    InitEIOServices(n, serverVec);
    InitQueuedCmdServices(n, serverVec);
    … ....
    void InitQueuedCmdServices(ros::NodeHandle &n, std::vector<ros::ServiceServer> &serverVec)
    {
```

```
    ros::ServiceServer server;
//Register the callback function
    server = n.advertiseService("/DobotServer/SetQueuedCmdStartExec", SetQueuedCmdStartExecService);
    serverVec.push_back(server);
    server = n.advertiseService("/DobotServer/SetQueuedCmdStopExec", SetQueuedCmdStopExecService);
    serverVec.push_back(server);
    …. ….
    }
```

（5） Wait for service request and enter the callback function after receive a request.

Program 1.5 Enter the callback function

```
    ROS_INFO("Dobot service running...");
    ros::spin();
```

（6） Process the service request and return a response.

Program 1.6 Process the service request and return a response

```
     bool SetQueuedCmdStartExecService(dobot::SetQueuedCmdStartExec::Request &req,
dobot::SetQueuedCmdStartExec::Response &res)
    {
        res.result = SetQueuedCmdStartExec();
        return true;
    }
```

（7） Disconnect Dobot Magician

Program 1.7 Disconnect Dobot Magician

```
    ROS_INFO("Dobot service exiting...");
    DisconnectDobot();
```

- Figure 1.14 shows the process of creating a client (take **DobotClient_PTP.cpp** as an example).

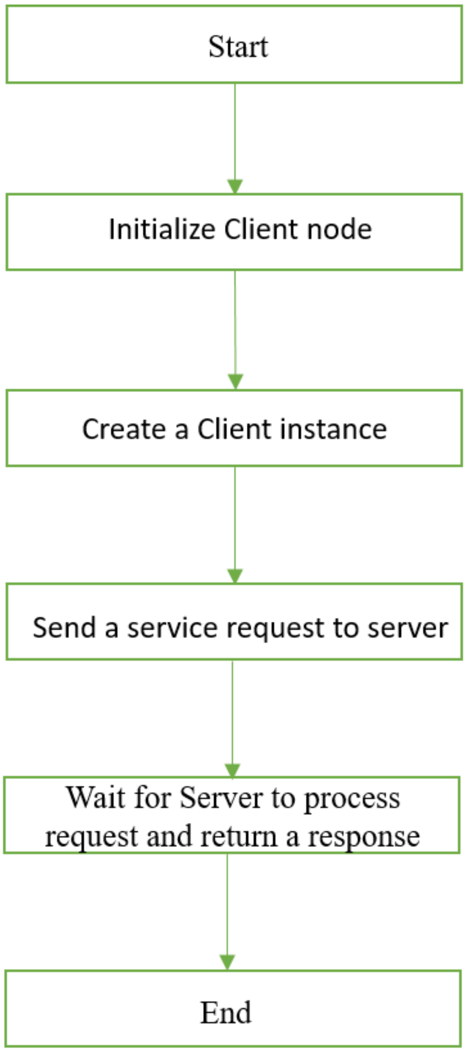

Figure 1.14 Process of creating a client

（1） Initialize Client node.

Program 1.8 Initialize Client node

```
ros::init(argc, argv, "DobotClient");        // Initialize Client node called DobotClient
ros::NodeHandle n;      //Create the handle node
```

（2） Create a Client instance, send a service request, and wait for the Server to process the request and return a response.

Program 1.9 Create a Client instance and complete service request process

```
ros::ServiceClient client;    //Create a Client instance
… …
```



```
// Set PTP common parameters
do {
        // The service request is dobot::SetPTPCommonParams
        client = n.serviceClient<dobot::SetPTPCommonParams>("/DobotServer/SetPTPCommonParams");
        dobot::SetPTPCommonParams srv;

        srv.request.velocityRatio = 50;
        srv.request.accelerationRatio = 50;
        // Send the service request and wait for the Server to process the request and return a response
        client.call(srv);
    } while (0);
```