
CSCE574 – Robotics

Spring 2012 – Notes on ROS

The goal of this document is to fill in and flesh out some details about ROS, based on your questions. It may be edited and expanded throughout the semester.

1 How to publish messages

- Step 1: Determine the name and data type of the correct topic.
 1. Start `roscore`.
 2. Start the node that you would like to talk to.
 3. Start `rxgraph`.
 4. Click “All Topics.”
 5. Find the topic you want and hover your mouse pointer over its rectangle.
 6. The *name* of the topic is the text that appears in the rectangle.
 7. The *data type* of the topic appears in the column to the right. This data type has a *package name* and a *type name*, separated by a slash.
 8. Remember the topic name, the package name, and the type name, because you’ll need them in the next steps.

- Step 2: Make sure your package depends on the package for the data type you need.

1. In your package directory, open the file `manifest.xml`.
2. Look for a line that reads:

```
<depend package="packageName" />
```

3. If this line does not exist, add it between the `<package>` and `</package>` tags.

This step is needed to ensure that `rosmake` makes the correct header files available when you compile your program.

- Step 3: Include the header file for the data type you need.

1. Near the top of your C++ code, include a line like this:

```
#include "packageName/typeName.h"
```

2. Example: If `rxgraph` showed “`turtlesim/Velocity`”, then your code would say:

```
#include "turtlesim/Velocity.h"
```

3. Example: If `rxgraph` showed “`sensor_msgs/LaserScan`”, then your code would say:

```
#include "sensor_msgs/LaserScan.h"
```

- Step 4: Create a publisher.

1. Shortly after you create your `ros::NodeHandle` object, use it to create a `Publisher` object, like this:

```
ros::Publisher pub = n.advertise<packageName::typeName>(
    "topicName", 1000);
```

2. Example: If `rxgraph` showed a topic named `/turtle1/command_velocity` with type `turtlesim/Velocity` then your code would say:

```
ros::Publisher pub = n.advertise<turtlesim::Velocity>(
    "/turtle1/command_velocity", 1000);
```

3. Example: If `rxgraph` showed a topic named `/ik` with type `kinematics_msgs/PositionIKRequest`, then your code would say:

```
ros::Publisher pub =
    n.advertise<kinematics_msgs::PositionIKRequest>("ik", 1000);
```

- Step 5: Create a message object.

1. Declare a variable of type `packageName::typeName`.
2. Example:

```
turtlesim::Velocity velMsg;
```

3. Example:

```
sensor_msgs::LaserScan scanMsg;
```

4. Example:

```
kinematics_msgs::PositionIKRequest ikReqMsg;
```

- Step 6: Fill in the data members of your message object.

1. At the command line, use `rosmg show packageName/typeName` to get a list of the data members.
2. In your C++ code, assign a value to each of these data members.
3. Example: The output of `rosmg show turtlesim/Velocity` is

```
float32 linear
float32 angular
```

So your C++ code must provide floating point values for the `linear` and `angular` members of the message object:

```
turtlesim::Velocity velMsg;
velMsg.linear = 1.0;
velMsg.angular = 0.0;
```

- Step 7: Use your Publisher object to publish your message.

1. Use the `publish` method of your publisher object.
2. Example:

```
pub.publish(velMsg);
```

- Step 8: Give ROS a chance to send the message.

1. Call `ros::spinOnce()`.

2 How to call services

- Step 1: Determine the name of the correct service.
 1. Start `roscore`.
 2. Start the node that you think might provide the service you want.
 3. Start `rxgraph`.
 4. Find your node and hover your mouse pointer over its oval.
 5. Information about this node, include a list of services that it provides, will appear in the column to the right.
 6. Remember the service name, because you'll need it in the next steps.
 7. Example: The standard `turtlesim` node provides a service called `/turtle1/set_pen`.
- Step 2: Determine the data type used by your service.
 1. At the command line, say `rosservice type serviceName`. The output is the data type used by the service. This data type has a *package name* and a *type name*, separated by a slash.
 2. Example: The output of `rosservice type /turtle1/set_pen` is `turtlesim/SetPen`. The package name is `turtlesim` and the type name is `SetPen`.
- Step 3: Make sure your package depends on the package for the data type you need.
 1. In your package directory, open the file `manifest.xml`.
 2. Look for a line that reads:

```
<depend package="packageName" />
```

3. If this line does not exist, add it between the `<package>` and `</package>` tags.

This step is needed to ensure that `rosmake` makes the correct header files available when you compile your program.

- Step 3: Include the header file for the data type you need.
 1. Near the top of your C++ code, include a line like this:

```
#include "packageName/typeName.h"
```

2. Example: If `rosservice type` showed `"turtlesim/SetPen"`, then your code would say:

```
#include "turtlesim/SetPen.h"
```

- Step 4: Create a client.

1. Shortly after you create your `ros::NodeHandle` object, use it to create a `ServiceClient` object, like this:

```
ros::ServiceClient client =  
    n.serviceClient<packageName::typeName>("serviceName");
```

2. Example: If `rxgraph` showed a service named `/turtle1/set_pen` with type `turtlesim/SetPen`, then your code would say:

```
ros::ServiceClient client =  
    n.serviceClient<turtlesim::SetPen>("/turtle1/set_pen");
```

-
- Step 5: Create a service request (“srv”) object.
 1. Declare a variable of type `packageName::typeName`.
 2. Example:

```
turtlesim::SetPen setPenSrv;
```

- Step 6: Fill in the data members of your srv, if any.
 1. At the command line, use `rossrv show packageName/typeName` to get a list of the data members. The output will show data members that are part of the *request* that you will send, followed by a line, followed by the data members that are part of the *response* that the server will send back.
 2. In your C++ code, assign a value to each of these data members of the request part.
 3. Example: The output of `rosmmsg show turtlesim/SetPen` is

```
uint8 r
uint8 g
uint8 b
uint8 width
uint8 off
---
```

So your C++ code must provide integer values for the `r`, `g`, `b`, `width`, and `off` members of the angular members of the srv:

```
turtlesim::SetPen setPenSrv;
setPenSrv.request.r = 128;
setPenSrv.request.g = 128;
setPenSrv.request.b = 128;
setPenSrv.request.width = 3;
setPenSrv.request.off = 0;
```

4. Example: The output of `rosmmsg show std_srvs/Empty` is:

```
---
```

This means that there are no data members to fill in. Just create the `std_srvs::Empty` object, and it’s ready to go.

- Step 7: Use your ServiceClient object to call the service.
 1. Use the `call` method of your client object.
 2. Example:

```
bool success = client.call(setPenSrv);
```

3. This call will wait until the node provide the service has completed our request.

- Step 8: Inspect the data members in the *response* part of your srv, if any.
 1. Remember that the data members that you’ll have access to are listed in the output of `rossrv show`.

-
2. Example: The `set_pen` service does not have any data members in its response, so we know there's no information coming back from the `turtlesim` node; there's nothing to do for this step.
 3. Example: Some nodes provide a service of type `diagnostic_msgs/SelfTest`. The output of `rossrv show diagnostic_msgs/SelfTest` is:

```
---
string id
byte passed
diagnostic_msgs/DiagnosticStatus[] status
byte OK=0
...
```

Each line after the `---` shows a data member of the response part of your `srv`, so you could, for example, say something like:

```
client.call(diagnosticSrv);
if(!diagnosticSrv.response.passed) {
    ROS_INFO("Oh no! Diagnostic failed!");
}
```

3 How to set parameters

- Step 1: Determine the name of the parameter you would like to set.

1. At the command line, say

```
rosparam list
```

to get a list of parameters that have been set.

2. Example: After running the `turtlesim`, you'll see a list something like this:

```
/background.b
/background.g
/background.r
/rosdistro
/rosversion
```

Each of these is a parameter that you can set.

- Step 2: Use the `setParam` method of your `NodeHandle` to change the value of that parameter.

1. Just one line is enough:

```
n.setParam("paramName", value);
```

2. The *value* can be a `bool`, an `int`, a `double` or a `string`.

3. Example: To set the background color used by `turtlesim` to black:

```
n.setParam("background.r", 0);
n.setParam("background.g", 0);
n.setParam("background.b", 0);
```

Then call the `clear` service (See Section 1 to get `turtlesim` to read the new values).