Note: This tutorial assumes that you have completed the previous tutorials: Beginner Tutorials (/ROS/Tutorials).

Please ask about problems and questions regarding this tutorial on answers.ros.org (http://answers.ros.org). Don't forget to include in your question the link to this page, the versions of your OS & ROS, and also add appropriate tags.

Writing Publisher/Subscriber with Parameters, Dynamic Reconfigure and Custom Messages (C++)

Description: This tutorial covers writing a publisher and a subscriber in C++. The use of custom messages and a publisher with a dynamic reconfigure server are covered.

Tutorial Level: INTERMEDIATE

Next Tutorial: Using C++ and Python Nodes Together (/ROSNodeTutorialC%2B%2BPython)

Contents

1. Writing a Publisher/Subscriber with Dynamic Reconfigure and Parameter Server (C++)

This tutorial will show you how to combine several beginner level tutorials to create publisher and subscriber nodes (/Nodes) that are more fully-featured than the previously created nodes. This page will describe how to create a publisher node that:

- Initializes several variables from either a launch file or using the command line by making use of the parameter server (/Parameter%20Server).
- Makes several variables available to be modified during run-time using a dynamic reconfigure server (/dynamic_reconfigure).
- Publishes data on a topic (/Topics) using custom messages (/Messages).

A subscriber node will be created to work with the publisher node that

- Sets up a subscriber to listen for the custom message on the specified topic and prints out the message data.

The variables that are used, and the callback functions used by the publisher and subscriber, will be part of a class that is created for this purpose as described in this tutorial (/roscpp_tutorials/Tutorials/UsingClassMethodsAsCallbacks).

It is assumed that all of the beginner tutorials (/ROS/Tutorials) will have been completed before using this one.

1.1 How to Run Examples

There are two programs created here, very similar to what is found at the ROS Publisher/Subscriber tutorial (/ROS/Tutorials/WritingPublisherSubscriber%28c%2B%2B%29). They are even called the same names, `talker` and `listener`. If you haven't already checked out the code for this tutorial then you can get it into your home directory (represented as ~ in Unix-based systems) using

```
git clone https://github.com/tdenewiler/node_example.git
```

Make sure that the directory where you have the example code is in your `~/.bashrc` file under `ROS_PACKAGE_PATH`, similar to what is done when you install ROS (/diamondback/Installation/Ubuntu) initially. Assuming that `node_example` is checked out to your home directory you will need to add, near the bottom of your `~/.bashrc` file, the line

```
export ROS_PACKAGE_PATH=~/node_example:$ROS_PACKAGE_PATH
```

After adding that line either (i) restart terminal or (ii) run `source ~/.bashrc`. Those commands will both let the system know about this new environment variable and allow ROS to find the new node.

Then, make the example nodes using

```
cd ~/node_example
cmake .
rosmake
```

The command `cmake .` will create a `Makefile` and any other auto-generated code. The auto-generated code is in header files and is created when `rosbuild_genmsg()` and `gencfg()` are invoked from `CMakeLists.txt`. The rest of `rosmake` will generate the two nodes (binaries) that we will be running, both located in `~/node_example/bin/`. Using four terminals (or four tabs in a single terminal -- use `<ctrl-shift-t>` to start a new tab in an existing terminal, and `<alt-#>` to easily switch between tabs) run the following four commands.

```
roscore
rosrun node_example talker
rosrun node_example listener
rosrun rqt_reconfigure rqt_reconfigure
```

You should see output from the terminal running the `listener` node. If you change the `message` variable or either of the integer values in the `reconfigure_gui` then the output of `listener` should change to match it.

Alternatively, you can run everything using a launch file by running

```
roslaunch node_example c++_node_example.launch
```

The launch file starts `talker`, `listener`, `reconfigure_gui` and `rxconsole`. The output of `listener` will only appear in `rxconsole` and not in the terminal when using the launch file. See the rxconsole (/rxconsole) page for more details on how to use that tool.

The following sections go into more detail about how the nodes work.

1.2 Creating a Custom Message

This tutorial (/ROS/Tutorials/CreatingMsgAndSrv) describes messages in more detail. The custom message for these nodes contains

```
string message
int32 a
int32 b
```

After creating that file in the `msg/` directory and using `rosbuild_genmsg()` in `CMakeLists.txt` nothing else needs to be done for the custom message.

1.3 Creating Configuration for Dynamic Reconfigure Server

This tutorial (/dynamic_reconfigure/Tutorials/SettingUpDynamicReconfigureForANode) describes dynamic reconfigure in more detail. The variables available for use with the dynamic reconfigure server (that we will create later) are specified in the following file called `node_example_params.cfg` and located in the `cfg/` directory.

Toggle line numbers

```
#! /usr/bin/env python

PACKAGE='node_example'
import roslib
roslib.load_manifest(PACKAGE)

from dynamic_reconfigure.parameter_generator import *

gen = ParameterGenerator()
#       Name       Type      Level Description     Default Min   Max
gen.add("message", str_t,    0,    "The message.", "hello")
gen.add("a",       int_t,    0,    "First number.", 1,     -100, 100)
gen.add("b",       int_t,    0,    "First number.", 2,     -100, 100)

exit(gen.generate(PACKAGE, "node_example", "node_example_params"))
```

This means we will be able to modify a message and two integers. Eventually, we will be able to modify these values and see the results while our nodes are running. Make sure the file `node_example_params.cfg` is executable by doing

```
chmod 755 ~/node_example/cfg/node_example_params.cfg
```

Then, after using `gencfg()` in `CMakeLists.txt` we will have everything we need to use a dynamic reconfigure server.

2. ROS Node Template

There are four files used to create the example nodes. There is one source and one header file that describe the class that is shared by `listener` and `talker`. Then, there is one source file to implement each of `listener` and `talker`. Note that the code style follows the ROS C++ style guide (/CppStyleGuide).

2.1 The talker node

The `node_example/src/talker.cpp` source file contains

Toggle line numbers

```
   1 #include "node_example_core.h"
   2
   3 /*------------------------------------------------------------------
----
   4  * main()
   5  * Main function to set up ROS node.
   6  *------------------------------------------------------------------
--*/
   7
   8 int main(int argc, char **argv)
   9 {
  10   // Set up ROS.
  11   ros::init(argc, argv, "talker");
  12   ros::NodeHandle n;
  13
  14   // Create a new NodeExample object.
  15   NodeExample *node_example = new NodeExample();
  16
  17   // Set up a dynamic reconfigure server.
  18   // This should be done before reading parameter server values.
  19   dynamic_reconfigure::Server<node_example::node_example_paramsCon
fig> dr_srv;
  20   dynamic_reconfigure::Server<node_example::node_example_paramsCon
fig>::CallbackType cb;
  21   cb = boost::bind(&NodeExample::configCallback, node_example, _1,
_2);
  22   dr_srv.setCallback(cb);
  23
  24   // Declare variables that can be modified by launch file or comm
and line.
  25   int a;
  26   int b;
  27   string message;
  28   int rate;
  29   string topic;
  30
  31   // Initialize node parameters from launch file or command line.
  32   // Use a private node handle so that multiple instances of the n
ode can
  33   // be run simultaneously while using different parameters.
  34   // Parameters defined in the .cfg file do not need to be initial
ized here
  35   // as the dynamic_reconfigure::Server does this for you.
  36   ros::NodeHandle private_node_handle_("~");
  37   private_node_handle_.param("rate", rate, int(40));
  38   private_node_handle_.param("topic", topic, string("example"));
  39
```

```
  40   // Create a publisher and name the topic.
  41   ros::Publisher pub_message = n.advertise<node_example::node_exam
ple_data>(topic.c_str(), 10);
  42
  43   // Tell ROS how fast to run this node.
  44   ros::Rate r(rate);
  45
  46   // Main loop.
  47   while (n.ok())
  48   {
  49     // Publish the message.
  50     node_example->publishMessage(&pub_message);
  51
  52     ros::spinOnce();
  53     r.sleep();
  54   }
  55
  56   return 0;
  57 } // end main()
  58
```

2.2 The listener node

The `node_example/src/listener.cpp` source file contains

Toggle line numbers

```
  1 #include "node_example_core.h"
  2
  3 /*-----------------------------------------------------------------
----
  4  * main()
  5  * Main function to set up ROS node.
  6  *-----------------------------------------------------------------
--*/
  7
  8 int main(int argc, char **argv)
  9 {
 10   // Set up ROS.
 11   ros::init(argc, argv, "listener");
 12   ros::NodeHandle n;
 13
 14   // Declare variables that can be modified by launch file or comm
and line.
 15   int rate;
 16   string topic;
 17
 18   // Initialize node parameters from launch file or command line.
 19   // Use a private node handle so that multiple instances of the n
ode can be run simultaneously
 20   // while using different parameters.
 21   ros::NodeHandle private_node_handle_("~");
 22   private_node_handle_.param("rate", rate, int(40));
 23   private_node_handle_.param("topic", topic, string("example"));
 24
 25   // Create a new NodeExample object.
 26   NodeExample *node_example = new NodeExample();
 27
 28   // Create a subscriber.
 29   // Name the topic, message queue, callback function with class n
ame, and object containing callback function.
 30   ros::Subscriber sub_message = n.subscribe(topic.c_str(), 1000, &
NodeExample::messageCallback, node_example);
 31
 32   // Tell ROS how fast to run this node.
 33   ros::Rate r(rate);
 34
 35   // Main loop.
 36   while (n.ok())
 37   {
 38     ros::spinOnce();
 39     r.sleep();
 40   }
 41
```

```
42   return 0;
43 } // end main()
44
```

2.3 The NodeExample class

The source file for the `node_example/src/node_example_core.cpp` class contains

Toggle line numbers

```
   1 #include "node_example_core.h"
   2
   3 /*-----------------------------------------------------------------
----
   4  * NodeExample()
   5  * Constructor.
   6  *-----------------------------------------------------------------
--*/
   7
   8 NodeExample::NodeExample()
   9 {
  10 } // end NodeExample()
  11
  12 /*-----------------------------------------------------------------
----
  13  * ~NodeExample()
  14  * Destructor.
  15  *-----------------------------------------------------------------
--*/
  16
  17 NodeExample::~NodeExample()
  18 {
  19 } // end ~NodeExample()
  20
  21 /*-----------------------------------------------------------------
----
  22  * publishMessage()
  23  * Publish the message.
  24  *-----------------------------------------------------------------
--*/
  25
  26 void NodeExample::publishMessage(ros::Publisher *pub_message)
  27 {
  28   node_example::node_example_data msg;
  29   msg.message = message;
  30   msg.a = a;
  31   msg.b = b;
  32
  33   pub_message->publish(msg);
  34 } // end publishMessage()
  35
  36 /*-----------------------------------------------------------------
----
  37  * messageCallback()
  38  * Callback function for subscriber.
  39  *-----------------------------------------------------------------
--*/
```

```
  40
  41 void NodeExample::messageCallback(const node_example::node_example
_data::ConstPtr &msg)
  42 {
  43   message = msg->message;
  44   a = msg->a;
  45   b = msg->b;
  46
  47   // Note that these are only set to INFO so they will print to a
terminal for example purposes.
  48   // Typically, they should be DEBUG.
  49   ROS_INFO("message is %s", message.c_str());
  50   ROS_INFO("sum of a + b = %d", a + b);
  51 } // end publishCallback()
  52
  53 /*-----------------------------------------------------------------
----
  54  * configCallback()
  55  * Callback function for dynamic reconfigure server.
  56  *-----------------------------------------------------------------
--*/
  57
  58 void NodeExample::configCallback(node_example::node_example_params
Config &config, uint32_t level)
  59 {
  60   // Set class variables to new values. They should match what is
input at the dynamic reconfigure GUI.
  61   message = config.message.c_str();
  62   a = config.a;
  63   b = config.b;
  64 } // end configCallback()
  65
```

2.4 The NodeExample header

The header file `node_example/include/node_example_core.h` for the class contains

Toggle line numbers

```
 1 #ifndef SR_NODE_EXAMPLE_CORE_H
 2 #define SR_NODE_EXAMPLE_CORE_H
 3
 4 // ROS includes.
 5 #include "ros/ros.h"
 6 #include "ros/time.h"
 7
 8 // Custom message includes. Auto-generated from msg/ directory.
 9 #include "node_example/node_example_data.h"
10
11 // Dynamic reconfigure includes.
12 #include <dynamic_reconfigure/server.h>
13 // Auto-generated from cfg/ directory.
14 #include <node_example/node_example_paramsConfig.h>
15
16 using std::string;
17
18 class NodeExample
19 {
20 public:
21   //! Constructor.
22   NodeExample();
23
24   //! Destructor.
25   ~NodeExample();
26
27   //! Callback function for dynamic reconfigure server.
28   void configCallback(node_example::node_example_paramsConfig &con
fig, uint32_t level);
29
30   //! Publish the message.
31   void publishMessage(ros::Publisher *pub_message);
32
33   //! Callback function for subscriber.
34   void messageCallback(const node_example::node_example_data::Cons
tPtr &msg);
35
36   //! The actual message.
37   string message;
38
39   //! The first integer to use in addition.
40   int a;
41
42   //! The second integer to use in addition.
43   int b;
44 };
45
```

```
46 #endif // SR_NODE_EXAMPLE_CORE_H
47
```

2.5 Building the code

After modifying the contents of `node_example/msg/node_example_data.msg` to add, remove or rename variables it is necessary to run

```
cd ~/node_example
cmake .
rosmake node_example
```

Invoking CMake again will auto-generate the new header files that contain information about the changes that were made to the message structure.

3. Using Parameter Server and Dynamic Reconfigure

3.1 Parameter Server

There are several ways of setting variables to initial values through the use of the parameter server (/Parameter%20Server). One is through a launch file, and another is from the command line.

In `node_example/c++_node_example.launch` the `talker` node is started with four parameters set, `message`, `a`, `b` and `rate`. To do the same thing from the command line `talker` could be started using

```
rosrun node_example talker _message:="Hello world!" _a:=57 _b:=-15 _rat
e:=1
```

Note that the ~ has been replaced by an underscore when modifying the private node handle parameters from the command line.

Then run

```
rosrun node_example listener
```

to see the differences.

Note that our nodes use private node handles (/roscpp/Overview/NodeHandles) for the parameter server. This is important because it helps to prevent name collisions when nodes are remapped to have unique node names. For instance, you may want to start two separate instances of a single node. With private node handles the separate nodes can have different

values for the same parameter name. This comes in handy when you have multiple cameras of the same type but want to run them at different frame rates (this is just one example out of many for using private node handles).

3.2 Dynamic Reconfigure

The dynamic reconfigure tools are awesome, because they allow users to modify variable *during* runtime, not just at the start of runtime. We have already created a file specifying which variables are available to the dynamic reconfigure server. Note that the file `node_example/manifest.xml` has to have the line

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

The `gencfg()` line in `CMakeLists.txt` causes the file `node_example/cfg/cpp/node_example/node_example_paramsConfig.h` to be auto-generated and this file is included in `node_example/include/node_example_core.h`.

4. Review - What to Change

To use this example code as a new node you have to change several variables.

1. Edit `manifest.xml` to depend on any other packages you need.
2. Edit `CMakeLists.txt` to change the name of the executable(s) to build, and the source files that are used.
3. Rename `cfg/node_example_params.cfg` to match the name of your node.
    - Change the PACKAGE= line to match the name of your node.
    - Change the last line to use the name of your node in the two places where `node_example` is present.
    - Modify the variables you make available to the dynamic reconfigure server.
4. Rename `msg/node_example_data.msg` to match the name of your node.
    - Modify the variables you want in the new message.
5. Rename `include/node_example_core.h`.
    - Modify the `#inlcude` lines to use your newly generated header files from the `.cfg` and `.msg` files.
    - Modify your class name, functions and variables.
6. Rename `src/node_example_core.cpp`.
    - Use your new include file for your node.
    - Modify the class name, functions and variables.
7. Modify `src/talker.cpp` or `src/listener.cpp`, rename them if you want.
    - Set up only the parts you want to use from the examples.
    - It is possible to combine initial configuration parameters, dynamic reconfigure server, publisher(s) and subscriber(s) all into a single node if desired.

Wiki: ROSNodeTutorialC++ (last edited 2014-07-24 03:33:29 by ThomasDenewiler (/ThomasDenewiler))