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

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.

1. Creating a ROS msg and srv

Description: This tutorial covers how to create and build msg and srv files as well as the rosmsg (/rosmsg), rossrv and roscp commandline tools.

Tutorial Level: BEGINNER

Next Tutorial: Writing a simple publisher and subscriber (python) (/ROS/Tutorials/WritingPublisherSubscriber%28python%29) (c++) (/ROS/Tutorials/WritingPublisherSubscriber%28c%2B%2B%29)

catkin	rosbuild

Contents

1.1 Introduction to msg and srv

- msg (/msg): msg files are simple text files that describe the fields of a ROS message. They are used to generate source code for messages in different languages.
- srv (/srv): an srv file describes a service. It is composed of two parts: a request and a response.

msg files are stored in the `msg` directory of a package, and srv files are stored in the `srv` directory.

msgs are just simple text files with a field type and field name per line. The field types you can use are:

- int8, int16, int32, int64 (plus uint*)
- float32, float64
- string
- time, duration
- other msg files
- variable-length array[] and fixed-length array[C]

There is also a special type in ROS: `Header`, the header contains a timestamp and coordinate frame information that are commonly used in ROS. You will frequently see the first line in a msg file have `Header header`.

Here is an example of a msg that uses a Header, a string primitive, and two other msgs :

```
  Header header
  string child_frame_id
  geometry_msgs/PoseWithCovariance pose
  geometry_msgs/TwistWithCovariance twist
```

srv files are just like msg files, except they contain two parts: a request and a response. The two parts are separated by a '---' line. Here is an example of a srv file:

```
int64 A
int64 B
---
int64 Sum
```

In the above example, `A` and `B` are the request, and `Sum` is the response.

1.2 Using msg

1.2.1 Creating a msg

Let's define a new msg in the package that was created in the previous tutorial.

```
$ roscd beginner_tutorials
$ mkdir msg
$ echo "int64 num" > msg/Num.msg
```

The example above is the simplest, where the `.msg` file contains only 1 line. You can, of course, create more complex files by adding multiple elements per line like this:

```
string first_name
string last_name
uint8 age
uint32 score
```

There's one more step, though. We need to make sure that the msg files are turned into source code for C++, Python, and other languages. Open `CMakeLists.txt` in your favorite text editor (rosed (/ROS/Tutorials/UsingRosEd) from the previous tutorial is a good option) and remove # to uncomment the following line:

```
# rosbuild_genmsg()
```

1.3 Using rosmsg

That's all you need to do to create a msg. Let's make sure that ROS can see it using the `rosmsg show` command.

Usage:

```
$ rosmsg show [message type]
```

Example:

```
$ rosmsg show beginner_tutorials/Num
```

You will see:

```
int64 num
```

In the previous example, the message type consists of two parts:

- `beginner_tutorials` -- the package where the message is defined
- `Num` -- The name of the msg `Num`.

If you can't remember which Package a msg is in, you can leave out the package name. Try:

```
$ rosmsg show Num
```

You will see:

```
[beginner_tutorials/Num]:
int64 num
```

1.4 Using srv

1.4.1 Creating a srv

Let's use the package we just created to create a srv:

```
$ roscd beginner_tutorials
$ mkdir srv
```

Instead of creating a new srv definition by hand, we will copy an existing one from another package.

For that, `roscp` is a useful commandline tool for copying files from one package to another.

Usage:

```
$ roscp [package_name] [file_to_copy_path] [copy_path]
```

Now we can copy a service from the rospy_tutorials (/rospy_tutorials) package:

```
$ roscp rospy_tutorials AddTwoInts.srv srv/AddTwoInts.srv
```

There's one more step, though. We need to make sure that the srv files are turned into source code for C++, Python, and other languages.

Once again, open CMakeLists.txt and remove # to uncomment the following line:

```
# rosbuild_gensrv()
```

1.4.2 Using rossrv

That's all you need to do to create a srv. Let's make sure that ROS can see it using the `rossrv show` command.

Usage:

```
$ rossrv show <service type>
```

Example:

```
$ rossrv show beginner_tutorials/AddTwoInts
```

You will see:

```
int64 a
int64 b
---
int64 sum
```

Similar to `rosmsg`, you can find service files like this without specifying package name:

```
$ rossrv show AddTwoInts
[beginner_tutorials/AddTwoInts]:
int64 a
int64 b
---
int64 sum

[rospy_tutorials/AddTwoInts]:
int64 a
int64 b
---
int64 sum
```

1.5 Common step for msg and srv

Now that we have made some new messages we need to make our package again:

```
$ rosmake beginner_tutorials
```

Any .msg file in the msg directory will generate code for use in all supported languages. The C++ message header file will be generated in `~/catkin_ws/devel/include/beginner_tutorials/`. The Python script will be created in `~/catkin_ws/devel/lib/python2.7/dist-packages/beginner_tutorials/msg`. The lisp file appears in `~/catkin_ws/devel/share/common-lisp/ros/beginner_tutorials/msg/`.

Similarly, any .srv files in the srv directory will have generated code in supported languages. For C++, this will generate header files in the same directory as the message header files. For Python and Lisp, there will be an 'srv' folder beside the 'msg' folders.

The full specification for the message format is available at the Message Description Language (/ROS/Message_Description_Language) page.

1.6 Getting Help

We've seen quite a few ROS tools already. It can be difficult to keep track of what arguments each command requires. Luckily, most ROS tools provide their own help.

Try:

```
$ rosmsg -h
```

You should see a list of different `rosmsg` subcommands.

```
Commands:
  rosmsg show Show message description
  rosmsg users  Find files that use message
  rosmsg md5  Display message md5sum
  rosmsg package  List messages in a package
  rosmsg packages List packages that contain messages
```

You can also get help for subcommands

```
$ rosmsg show -h
```

This shows the arguments that are needed for rosmsg show:

```
Usage: rosmsg show [options] <message type>

Options:
  -h, --help  show this help message and exit
  -r, --raw   show raw message text, including comments
```

1.7 Review

Let's just list some of the commands we've used so far:

- rospack = ros+pack(age) : provides information related to ROS packages
- roscd = ros+cd : **c**hanges **d**irectory to a ROS package or stack
- rosls = ros+ls : **l**ist**s** files in a ROS package
- roscp = ros+cp : **c**o**p**ies files from/to a ROS package
- rosmsg = ros+msg : provides information related to ROS message definitions
- rossrv = ros+srv : provides information related to ROS service definitions
- catkin_make : makes (compiles) a ROS package
    - rosmake = ros+make : makes (compiles) a ROS package (if you're not using a catkin workspace)

1.8 Next Tutorial

Now that you've made a new ROS msg and srv, let's look at writing a simple publisher and subscriber (python) (/ROS/Tutorials/WritingPublisherSubscriber%28python%29) (c++) (/ROS/Tutorials/WritingPublisherSubscriber%28c%2B%2B%29).

Wiki: ROS/Tutorials/CreatingMsgAndSrv (last edited 2016-01-21 22:51:18 by AustinHendrix (/AustinHendrix))

(http://www.osrfoundation.org)