

**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) (<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.

# Creating a ROS msg and srv

**Description:** This tutorial covers how to create and build msg and srv files as well as the rosmmsg (/rosmmsg), 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

roscpp

## Contents

1. Introduction to msg and srv
2. Using msg
  1. Creating a msg
3. Using rosmmsg
4. Using srv
  1. Creating a srv
  2. Using rossrv
5. Common step for msg and srv
6. Getting Help
7. Review
8. Next Tutorial

## 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.

## 2. Using msg

### 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 .msg file above contains only 1 line. You can, of course, create a more complex file by adding multiple elements, one 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 `package.xml`, and make sure these two lines are in it and  **uncommented** (<http://www.htmlhelp.com/reference/wilbur/misc/comment.html>):

```
<build_depend>message_generation</build_depend>
<exec_depend>message_runtime</exec_depend>
```

Note that at build time, we need "message\_generation", while at runtime, we only need "message\_runtime".

Open CMakeLists.txt in your favorite text editor (rosed (/ROS/Tutorials/UsingRosEd) from the previous tutorial is a good option).

Add the message\_generation dependency to the find\_package call which already exists in your CMakeLists.txt so that you can generate messages. You can do this by simply adding message\_generation to the list of COMPONENTS such that it looks like this:

```
# Do not just add this to your CMakeLists.txt, modify the existing text to add m
message_generation before the closing parenthesis
find_package(catkin REQUIRED COMPONENTS
  roscpp
  rospy
  std_msgs
  message_generation
)
```

You may notice that sometimes your project builds fine even if you did not call find\_package with all dependencies. This is because catkin combines all your projects into one, so if an earlier project calls find\_package, yours is configured with the same values. But forgetting the call means your project can easily break when built in isolation.

Also make sure you export the message runtime dependency.

```
catkin_package(
  ...
  CATKIN_DEPENDS message_runtime ...
  ...)
```

Find the following block of code:

```
# add_message_files(
#   FILES
#   Message1.msg
#   Message2.msg
# )
```

Uncomment it by removing the # symbols and then replace the stand in Message\*.msg files with your .msg file, such that it looks like this:

```
add_message_files(
  FILES
  Num.msg
)
```

By adding the .msg files manually, we make sure that CMake knows when it has to reconfigure the project after you add other .msg files.

Now we must ensure the generate\_messages ( ) function is called.

*For ROS Hydro and later, you need to uncomment these lines:*

```
# generate_messages(  
#   DEPENDENCIES  
#   std_msgs  
# )
```

so it looks like:

```
generate_messages(  
  DEPENDENCIES  
  std_msgs  
)
```

*In earlier versions*, you may just need to uncomment one line:

```
generate_messages()
```

Now you're ready to generate source files from your msg definition. If you want to do so right now, skip next sections to Common step for msg and srv (/ROS/Tutorials/CreatingMsgAndSrv#Common\_step\_for\_msg\_and\_srv).

### 3. Using rosmmsg

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

Usage:

```
$ rosmmsg show [message type]
```

Example:

```
$ rosmmsg 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:

```
$ rosmmsg show Num
```

You will see:

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

## 4. Using srv

### 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 cmdline 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.

Unless you have done so already, open `package.xml`, and make sure these two lines are in it and  uncommented (<http://www.htmlhelp.com/reference/wilbur/misc/comment.html>):

```
<build_depend>message_generation</build_depend>
<exec_depend>message_runtime</exec_depend>
```

As before, note that at build time, we need "message\_generation", while at runtime, we only need "message\_runtime".

Unless you have done so already for messages in the previous step, add the message\_generation dependency to generate messages in `CMakeLists.txt`:

```
# Do not just add this line to your CMakeLists.txt, modify the existing line
find_package(catkin REQUIRED COMPONENTS
  roscpp
  rospy
  std_msgs
  message_generation
)
```

(Despite its name, message\_generation works for both msg and srv.)

Also you need the same changes to package.xml for services as for messages, so look above for the additional dependencies required.

Remove # to uncomment the following lines:

```
# add_service_files(  
#   FILES  
#   Service1.srv  
#   Service2.srv  
# )
```

And replace the placeholder `Service*.srv` files for your service files:

```
add_service_files(  
  FILES  
  AddTwoInts.srv  
)
```

Now you're ready to generate source files from your service definition. If you want to do so right now, skip next sections to Common step for msg and srv (/ROS/Tutorials/CreatingMsgAndSrv#Common\_step\_for\_msg\_and\_srv).

## 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 `rosmmsg`, 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
```

Here, two services are shown. The first is the one you just created in the `beginner_tutorials` package, and the second is the pre-existing one from the `rospy_tutorials` package.

## 5. Common step for msg and srv

Unless you have already done this in the previous steps, change in `CMakeLists.txt` :

```
# generate_messages(  
#   DEPENDENCIES  
#   std_msgs # Or other packages containing msgs  
# )
```

Uncomment it and add any packages you depend on which contain `.msg` files that your messages use (in this case `std_msgs`), such that it looks like this:

```
generate_messages(  
  DEPENDENCIES  
    std_msgs  
)
```

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

```
# In your catkin workspace  
$ roscd beginner_tutorials  
$ cd ../..  
$ catkin_make install  
$ cd -
```

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.

If you are building C++ nodes which use your new messages, you will also need to declare a dependency between your node and your message, as described in the [catkin msg/srv build documentation](#) ([http://docs.ros.org/latest/api/catkin/html/howto/format2/building\\_msgs.html](http://docs.ros.org/latest/api/catkin/html/howto/format2/building_msgs.html)).

## 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:

```
$ rosmmsg -h
```

You should see a list of different rosmmsg subcommands.

Commands:

```
rosmmsg show      Show message description
rosmmsg list      List all messages
rosmmsg md5       Display message md5sum
rosmmsg package   List messages in a package
rosmmsg packages  List packages that contain messages
```

You can also get help for subcommands

```
$ rosmmsg show -h
```

This shows the arguments that are needed for rosmmsg show:

Usage: rosmmsg show [options] <message type>

Options:

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

## 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 : **changes** directory to a ROS package or stack
- rosls = ros+ls : **lists** files in a ROS package
- roscp = ros+cp : **copies** files from/to a ROS package
- rosmmsg = 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)

## 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).

Except where otherwise

noted, the ROS wiki is

Wiki: ROS/Tutorials/CreatingMsgAndSrv (last edited 2017-10-26 15:55:56 by DirkThomas (/DirkThomas))

licensed under the

Creative Commons Attribution 3.0 (<http://creativecommons.org/licenses/by/3.0/>) | Find us on Google+ (<https://plus.google.com/113789706402978299308>)



Brought to you by:  Open Source Robotics Foundation

(<http://www.osrfoundation.org>)