

ROS Filesystem Concepts (/ROS/Concepts): *Packages (/Packages) | Metapackages (/Metapackages) | Manifest (/Manifest) | msg | srv (/srv)*

For rosbuilt, see: rosbuilt/msg (/rosbuilt/msg)

目录

1. Command-line Tools

2. Message Description Specification

1. Fields

2. Constants

3. Building .msg Files

4. Client Library Support

ROS uses a simplified messages description language for describing the data values (aka messages (/Messages)) that ROS nodes (/Nodes) publish. This description makes it easy for ROS tools to automatically generate source code for the message type in several target languages. Message descriptions are stored in .msg files in the msg/ subdirectory of a ROS package (/Packages).

There are two parts to a .msg file: fields and constants. Fields are the data that is sent inside of the message. Constants define useful values that can be used to interpret those fields (e.g. enum-like constants for an integer value).

Message types are referred to using package resource names (/Names). For example, the file geometry_msgs/msg/Twist.msg is commonly referred to as geometry_msgs/Twist.

1. Command-line Tools

rosmmsg (/rosmmsg) prints out message definition information and can find source files that use a message type.

2. Message Description Specification

The format of this language is simple: a message description is a list of data field descriptions and constant definitions on separate lines.

2.1 Fields

Each field consists of a type and a name, separated by a space, i.e.:

```
fieldtype1 fieldname1
fieldtype2 fieldname2
fieldtype3 fieldname3
```

For example:

```
int32 x
int32 y
```

2.1.1 Field Types

Field types can be:

1. a built-in type, such as "float32 pan" or "string name"

2. names of Message descriptions defined on their own, such as "geometry_msgs/PoseStamped"

3. fixed- or variable-length arrays (lists) of the above, such as "float32[] ranges" or "Point32[10] points"





4. the special Header (/msg#headerSect) type, which maps to std_msgs/Header


When embedding other Message descriptions, the type name may be relative (e.g. "Point32") if it is in the same package; otherwise it must be the full Message type (e.g. "std_msgs/String"). The only exception to this rule is Header (/msg#headerSect).

NOTE: you must not use the names of built-in types or Header when constructing your own message types.

Built-in types:

Primitive Type	Serialization	C++	Python2	Python3
bool (1)	unsigned 8-bit int	uint8_t (2)		bool
int8	signed 8-bit int	int8_t		int
uint8	unsigned 8-bit int	uint8_t		int (3)

int16	signed 16-bit int	int16_t		int
uint16	unsigned 16-bit int	uint16_t		int
int32	signed 32-bit int	int32_t		int
uint32	unsigned 32-bit int	uint32_t		int
int64	signed 64-bit int	int64_t	long	int
uint64	unsigned 64-bit int	uint64_t	long	int
float32	32-bit IEEE float	float		float
float64	64-bit IEEE float	double		float
string	ascii string (4)	std::string	str	bytes
time	secs/nsecs unsigned 32-bit ints	 <code>ros::Time</code> (http://docs.ros.org/api/rostime/html/classros__1_1Time.html)		 <code>rospy.Time</code> (http://www.ros.org/doc/api/rospy/html/rospy.rostime.Time-class.html)
duration	secs/nsecs signed 32-bit ints	 <code>ros::Duration</code> (http://docs.ros.org/api/rostime/html/classros__1_1Duration.html)		 <code>rospy.Duration</code> (http://www.ros.org/doc/api/rospy/html/rospy.rostime.Duration-class.html)

1. bool was introduced in ROS 0.9
2. bool in C++ is aliased to uint8_t because of array types: `std::vector<bool>` is in fact a specialized form of vector that is not a container. See  <http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2007/n2160.html> (<http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2007/n2160.html>) for more information.
3. uint8 has special meaning in Python. `uint8[]` is treated as a Python `bytes` so that it is compatible with other byte-oriented APIs in Python.
4. unicode strings are currently not supported as a ROS data type. utf-8 should be used to be compatible with ROS string serialization. In python 2, this encoding is automatic for `unicode` objects, but decoding must be done manually. In python 3, when using `str`, both encoding and decoding using 'utf-8' in the generated message code.

Deprecated:

- char: deprecated alias for uint8
- byte: deprecated alias for int8

Array handling

Array Type	Serialization	C++	Python2	Python3
fixed-length	no extra serialization	0.11+: <code>boost::array<T, length></code> , otherwise: <code>std::vector<T></code>		tuple (1)
variable-length	uint32 length prefix	<code>std::vector<T></code>		tuple (1)
uint8[]	see above	as above	str	bytes (2)
bool[]	see above	<code>std::vector<uint8_t></code>		list of bool

1. In rospy (/rospy), arrays are deserialized as tuples for performance reasons, but you can set fields to tuples and lists interchangeably.
2. rospy (/rospy) treats `uint8[]` data as a `bytes`, which is the Python representation for byte data. In Python 2, this is the same as `str`.

rospy (/rospy) can also deserialize arrays into `numpy` data structures. Please consult the rospy (/rospy) documentation for more information.

2.1.2 Field Names

The field name determines how a data value is referenced in the target language. For example, a field called 'pan' would be referenced as 'obj.pan' in Python, assuming that 'obj' is the variable storing the message.

Field names must be translated by message generators to several target languages, so we restrict field names to be an alphabetical character followed by any mixture of alphanumeric and underscores, i.e. `[a-zA-Z][a-zA-Z1-9_]*`. It is recommended that you avoid using field names that correspond to keywords in common languages -- although those names are legal, they create confusion as to how a field name is translated.

2.1.3 Header

ROS provides the special `Header` type to provide a general mechanism for setting frame IDs for libraries like tf (/tf). While `Header` is not a built-in type (it's defined in `std_msgs/msg/Header.msg`), it is commonly used and has special semantics. If the first field of your `.msg` is:

Header header

It will be resolved as `std_msgs/Header`.

Header.msg:

```
#Standard metadata for higher-level flow data types
#sequence ID: consecutively increasing ID
uint32 seq
#Two-integer timestamp that is expressed as:
# * stamp.secs: seconds (stamp_secs) since epoch
# * stamp.nsecs: nanoseconds since stamp_secs
# time-handling sugar is provided by the client library
time stamp
#Frame this data is associated with
string frame_id
```

The special nature of Header is mainly for historical reasons, such as preserving recorded data.

2.2 Constants

Each constant definition is like a field description, except that it also assigns a value. This value assignment is indicated by use of an equal '=' sign, e.g.

```
constanttype1 CONSTANTNAME1=constantvalue1
constanttype2 CONSTANTNAME2=constantvalue2
```

For example:

```
int32 X=123
int32 Y=-123
string FOO=foo
string EXAMPLE="#comments" are ignored, and leading and trailing whitespace removed
```

Notes:

- With the exception of time and duration, you may declare any built-in type as a constant.
- string constants are assigned the value of everything to the right of the equals sign, with leading and trailing whitespace removed. As such, you cannot leave a comment on a string constant definition.
- Integer constants must be specified in decimal (base 10).

3. Building .msg Files

The ROS Client Libraries (/Client%20Libraries) implement message generators that translate .msg files into source code. These message generators must be invoked from your build script, though most of the gory details are taken care of by including some common build rules. By convention, all .msg files are stored in a directory within your package called "msg," and you can build all of them by editing the CMakeLists.txt (/CMakeLists) file and the catkin/package.xml (/catkin/package.xml) file.

Open package.xml, and make sure these two lines are in it:

```
<build_depend>message_generation</build_depend>
<run_depend>message_runtime</run_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 line to your CMakeLists.txt, modify the existing line
find_package(catkin REQUIRED COMPONENTS roscpp rospy std_msgs message_generation)
```

You may notice 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 build 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
)
```

Find the following block of code:

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

Uncomment it by removing the # symbols and then replace std_msgs with the messages your messages depend on, such that it looks like this:

```
generate_messages(
  DEPENDENCIES
  std_msgs
)
```

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.

Also see: [catkin/CMakeLists.txt#msgs_srvs_actions \(/catkin/CMakeLists.txt#msgs_srvs_actions\)](#)

4. Client Library Support

In Python, the generated Python message file (e.g. std_msgs.msg.String) provides nearly all the information you might want about a .msg file. You can examine the `__slots__` and `__slot_types` and other fields to introspect information about messages.

Except where otherwise noted, the ROS wiki is licensed under the Creative Commons Attribution 3.0 (<http://creativecommons.org/licenses/by/3.0/>)

Wiki: msg (2019-01-13 18:15:57由AustinHendrix (/AustinHendrix)编辑)

Brought to you by:  Open Source Robotics Foundation

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