***ROS Graph Concepts (/ROS/Concepts)**: Nodes (/Nodes) | Topics (/Topics) | Services (/Services) | Messages | Bags (/Bags) | Master (/Master) | Parameter Server (/Parameter%20Server)*

Nodes (/Nodes) communicate with each other by publishing **messages** to topics (/Topics). A message is a simple data structure, comprising typed fields. Standard primitive types (integer, floating point, boolean, etc.) are supported, as are arrays of primitive types. Messages can include arbitrarily nested structures and arrays (much like C structs).

Nodes can also exchange a *request* and *response* message as part of a ROS service (/Services) call. These request and response messages are defined in srv files (/srv).

1. msg files

msg (/msg) files are simple text files for specifying the data structure of a message. These files are stored in the `msg` subdirectory of a package. For more information about these files, including a type specification, see the msg format (/msg).

2. Message Types

Message types use standard ROS naming (/Names) conventions: the name of the package + / + name of the `.msg` file. For example, `std_msgs/msg/String.msg` has the message type `std_msgs/String`.

In addition to the message type, messages are versioned by an MD5 sum of the `.msg` file. Nodes can only communicate messages for which both the message type and MD5 sum match.

3. Building

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." If you have msgs defined there, you simply have to add the line `rosbuild_genmsg()` to your CMakeLists.txt (/CMakeLists) file. Here is an example:

```
cmake_minimum_required(VERSION 2.6)
include(rosbuild)
rosbuild_init()
rosbuild_genmsg()
```

4. Header

A message may include a special message type called 'Header', which includes some common metadata fields such as a timestamp and a frame ID. The ROS Client Libraries (/Client%20Libraries) will automatically set some of these fields for you if you wish, so their use is highly encouraged.

There are three fields in the header message shown below. The `seq` field corresponds to an id that automatically increases as messages are sent from a given publisher. The `stamp` field stores time information that should be associated with data in a message. In the case of a laser scan, for example, the stamp might correspond to the time at which the scan was taken. The `frame_id` field stores frame information that should be associated with data in a message. In the case of a laser scan, this would be set to the frame in which the scan was taken.

```
#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
# 0: no frame
# 1: global frame
string frame_id
```

5. rosmsg

rosmsg (/rosmsg) is a command-line tool for displaying information about messages, such as displaying the `.msg` data structures. See rosmsg (/rosmsg) for documentation on how to use this tool.