Developing on ROS Framework

ROS packages and facilities

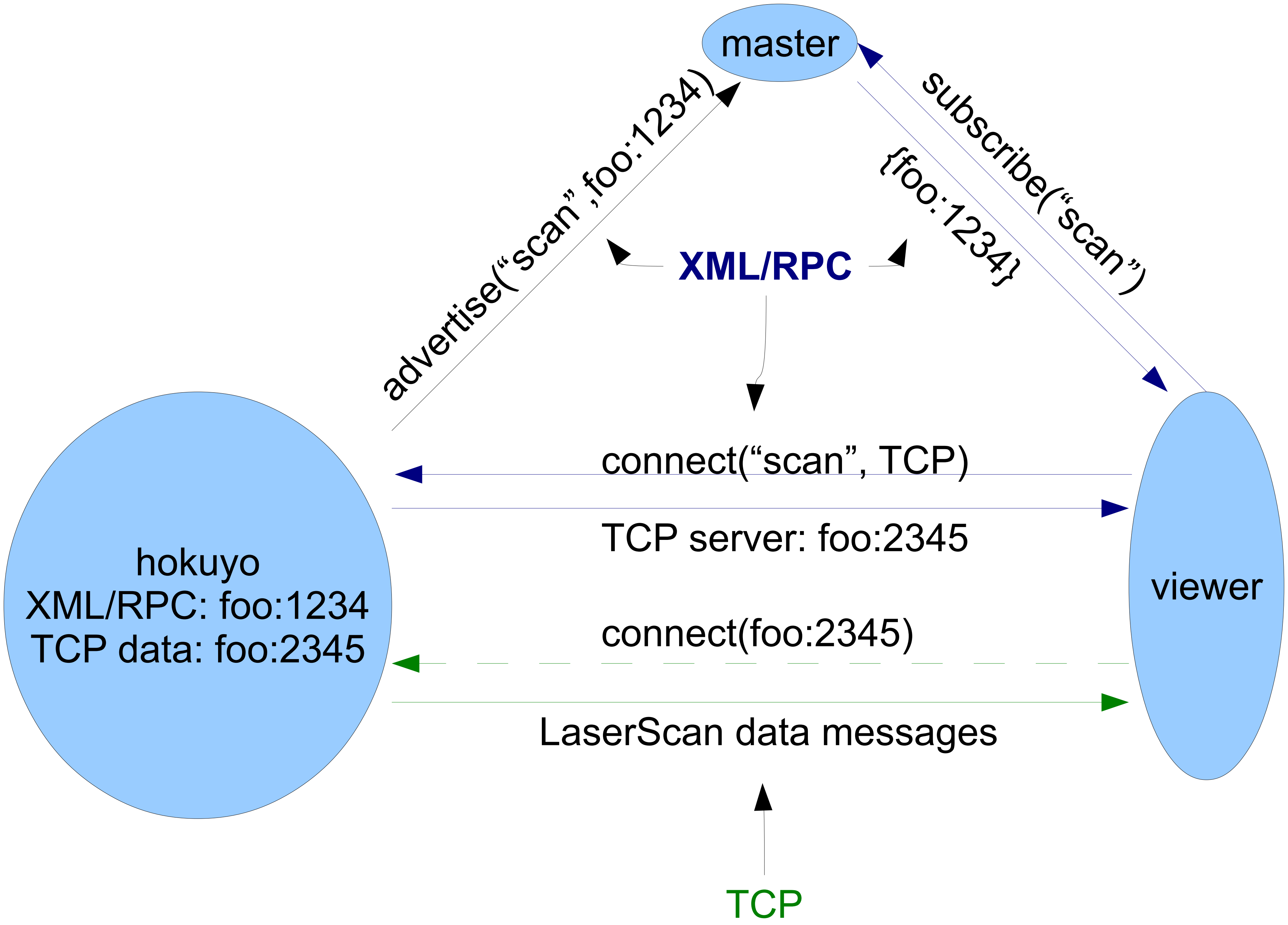
part I

Rodrigo Ventura
João Reis
Institute for Systems and Robotics
Instituto Superior Técnico, Lisboa

Lisbon, 23-26 June 2013

Sponsored by project FP7-ICT-2011-9-601033 (MOnarCH)

How ROS works inside?



Crash-course in XML

- Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable

- Historical origin: HTML tags

  In fact, HTML is now XML, the standard being called XHTML

- Building blocks:

  - tags: <xpto> data... </xpto> or <xpto/> if an empty tag

    example in HTML: <b>this is bold face</b> → **this is bold face**

  - attributes: <course name=”SCDTR”> data... </course>

    example in HTML: <a href=”http://ist.eu”>Instituto Superior Técnico</a>

    → Instituto Superior Técnico

Crash-course in XML

- XML files are hierarchical
  - example of an “hello world” HTML file:

```
<html>
    <body>
        <b>Hello world</b>
        <br/>
        <img src=”lenna.jpg” width=”100” height=”100”/>
    </body>
</html>
```

- Comments are enclosed by a “<!--” and a “-->” marker

```
<!-- this is a comment and it’s ignored by machines -->
```

roslaunch tool

- Inpractical to launch manually many ROS nodes

- roslaunch allows automatic launch of nodes from a single shell command
- roslaunch is configured using XML files
- Launch files are typically stored in the launch/ directory of a package

- roslaunch tool arguments:

  roslaunch [package] filename [arg_name:=value]*

Launch files

- Minimal launch file for the xpto package

```
# file l1.launch
<launch>
        <node pkg="xpto" name="node1" type="publisher.py"/>
</launch>
```

  - **attributes:**

    pkg="package_name"

    name="node_name"

    type="executable_filename"

Launch files

- Running...

```
$ roslaunch xpto l1.launch
[...]
core service [/rosout] found
process[node1-1]: started with pid [23774]
```

```
$ rosnode list
/node1
/rosout

$ rostopic list
/abc
/rosout
/rosout_agg
```

Launch files

- Several nodes

```
# file l2.launch
<launch>
        <node pkg="xpto" name="publisher" type="publisher.py"/>
        <node pkg="xpto" name="subscriber" type="subscriber.py"
         output="screen"/>
</launch>
```

```
$ roslaunch xpto l2.launch
[...]
process[publisher-1]: started with pid [23919]
process[subscriber-2]: started with pid [23920]
Received 'hello world #1'
Received 'hello world #2'
Received 'hello world #3'
```

Launch files

- Other <node> arguments:
  - launch node on a different machine

    machine="hostname"

    (use <machine> tags to declare machine names)
  - restart node whenever it quits

    respawn="true"
  - start node in a different namespace

    ns="namespace"
  - pass arguments to node

    args="arg1 arg2 arg3 ..."
  - ...

Launch files

- Tags allowed inside the <node> tag:
  - set environment variables

    <env name=”variable” value=”value”/>
  - remap names (nodes, topics, parameters)

    <remap from="original" to="new"/>
  - handle ROS parameters

    <rosparam command="load|dump|delete" file=”...”/>
  - send parameters to parameters server

    <param name=”...” type=”...” value=”...”/>
- *These tags can also be used in other scopes, i.e., globally scoped within the launch file*

Launch files

- Other relevant tags:
  - include launch files

    ```
    <include file=”filename”/>
    ```
  - group tags within a scope

    ```
    <group name=”…”>
    …
    </group>
    ```
  - declare machines

    ```
    <machine name="..." address="..." user="...” ...>
    …
    </machine>
    ```

Launch files

- Substitution arguments (i.e., macros)
  - package path name
    $(find package_name)
  - evaluates to value declared with tag <arg>
    $(arg argument_name)
  - evaluates to an environment variable that has to exist
    $(env variable_name)
  - same as $(env ...) but defaults to a given value if undefined
    $(optenv variable_name)
  - generate a unique (anonymous) name
    $(anon base_name)

Launch files

- Simple example:

```
# file l3.launch
<launch>
        <include ns="foo" file="$(find xpto)/launch/l2.launch"/>
        <include ns="bar" file="$(find xpto)/launch/l2.launch"/>
</launch>
```

```
$ rosnode list
/bar/publisher
/bar/subscriber
/foo/publisher
/foo/subscriber
/rosout

$ rostopic list
/bar/abc
/foo/abc
/rosout
/rosout_agg
```

Launch files

- Example top level organization

```
<launch>
  <group name="wg">
    <include file="$(find pr2_alpha)/$(env ROBOT).machine" />
    <include file="$(find 2dnav_pr2)/config/new_amcl_node.xml" />
    <include file="$(find 2dnav_pr2)/config/base_odom_teleop.xml" />
    <include file="$(find 2dnav_pr2)/config/lasers_and_filters.xml" />
    <include file="$(find 2dnav_pr2)/config/map_server.xml" />
    <include file="$(find 2dnav_pr2)/config/ground_plane.xml" />

    <!-- The navigation stack and associated parameters -->
    <include file="$(find 2dnav_pr2)/move_base/move_base.xml" />
  </group>
</launch>
```

Launch files

- Example from ISR-CoBot

```
<launch>
    <!-- LIDAR node -->
    <include file="lidar.launch"/>

    <!-- Run the map server -->
    <node name="map_server" pkg="map_server" type="map_server" args="$(find
maps)/sala_corredor.yaml"/>

    <!-- Odometry node -->
    <include file="$(find scout_odometry)/launch/odometry.launch"/>

    <!--- AMCL -->
    <include file="$(find amcl)/examples/amcl_diff.launch"/>

    <!-- Navigation -->
    <node name="navigation" pkg="scout_navigation" type="navigator"/>

    <!-- Speech synth -->
    <node name="speech" pkg="speech" type="server"/>

    <!-- Web console -->
    <node name="webconsole" pkg="webconsole" type="server"/>
</launch>
```

Message type definition

- Message types are defined in simple text files in the msg/ directory
- Sintax:

  # this is a comment

  fieldtype1 fieldname1

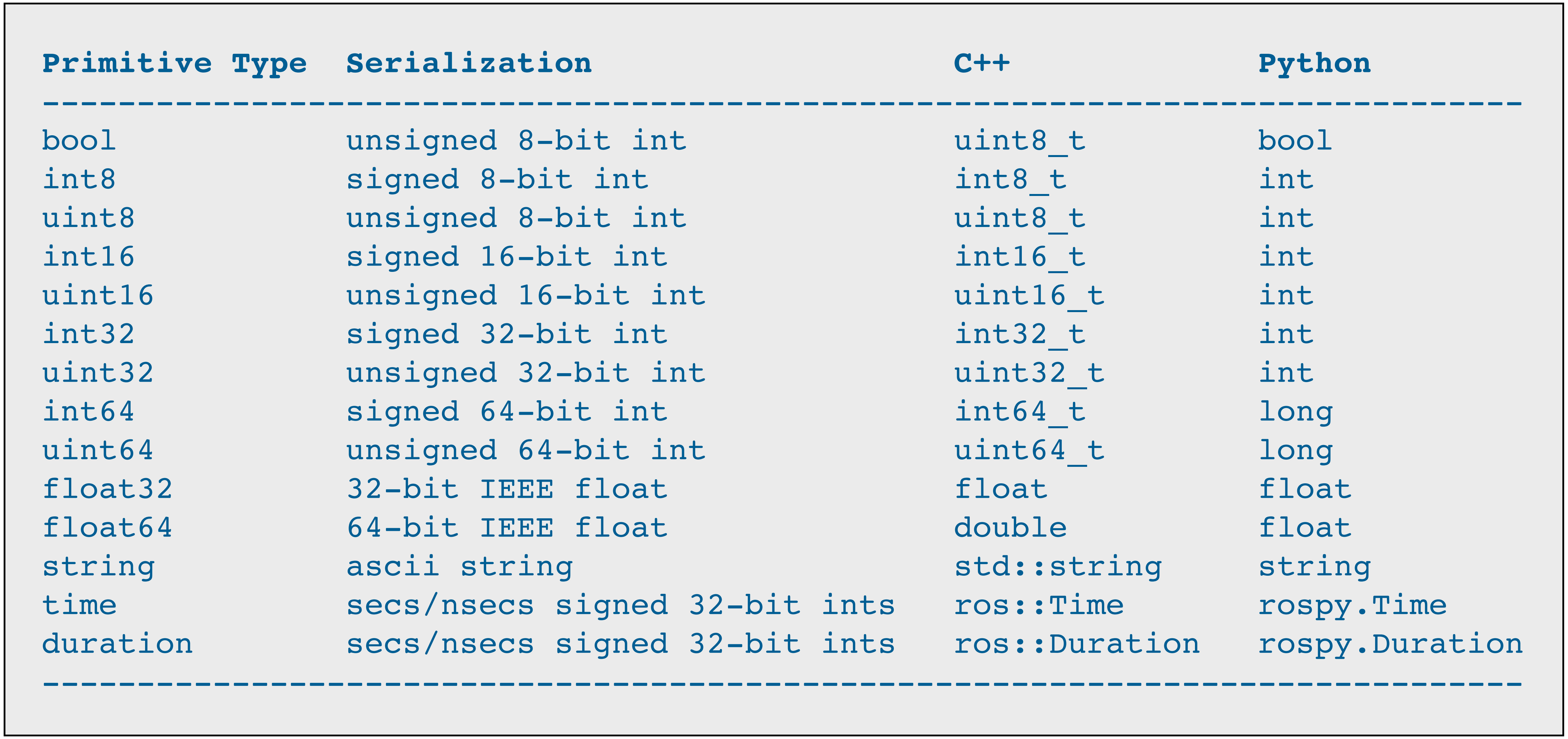
  fieldtype2 fieldname2

  ...
- Example:

```
float64 x
float64 y
float64 z
```

Message field types

- Built-in types:



```
Primitive Type  Serialization                    C++              Python
----------------------------------------------------------------------------
bool            unsigned 8-bit int               uint8_t          bool
int8            signed 8-bit int                 int8_t           int
uint8           unsigned 8-bit int               uint8_t          int
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
uint64          unsigned 64-bit int              uint64_t         long
float32         32-bit IEEE float                float            float
float64         64-bit IEEE float                double           float
string          ascii string                     std::string      string
time            secs/nsecs signed 32-bit ints    ros::Time        rospy.Time
duration        secs/nsecs signed 32-bit ints    ros::Duration    rospy.Duration
----------------------------------------------------------------------------
```

  - use '[]' after type to denote an array
    example: float64[] *is a string of float64's*

Message field types

- **Examples from** geometry_msgs **package**
  - Point.msg

```
# This contains the position of a point in free space
float64 x
float64 y
float64 z
```

  - Quaternion.msg

```
# This represents an orientation in free space in quaternion form.
float64 x
float64 y
float64 z
float64 w
```

Message field types

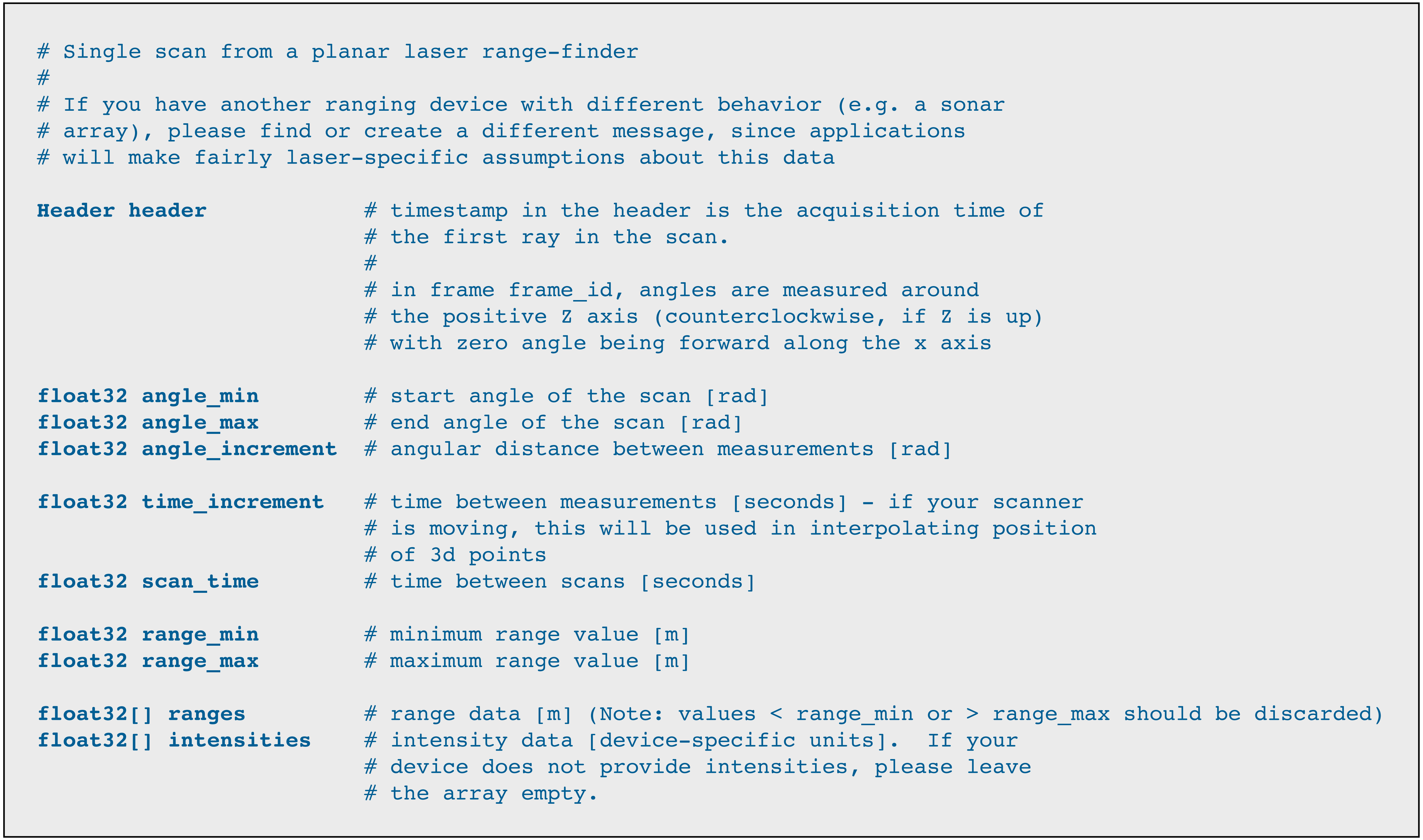
- Message types are themselves field types that can be used in another message type definitions
  - **example:** Header **type, defined in** std_msgs/Header.msg

```
# Standard metadata for higher-level stamped data types.
# This is generally used to communicate timestamped data
# in a particular coordinate frame.
#
# 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
```

  - this type is almost always used in other message types

Message field types

- **Example from** sensor_msgs **package:** LaserScan.msg



```
# Single scan from a planar laser range-finder
#
# If you have another ranging device with different behavior (e.g. a sonar
# array), please find or create a different message, since applications
# will make fairly laser-specific assumptions about this data

Header header            # timestamp in the header is the acquisition time of
                         # the first ray in the scan.
                         #
                         # in frame frame_id, angles are measured around
                         # the positive Z axis (counterclockwise, if Z is up)
                         # with zero angle being forward along the x axis

float32 angle_min        # start angle of the scan [rad]
float32 angle_max        # end angle of the scan [rad]
float32 angle_increment  # angular distance between measurements [rad]

float32 time_increment   # time between measurements [seconds] - if your scanner
                         # is moving, this will be used in interpolating position
                         # of 3d points
float32 scan_time        # time between scans [seconds]

float32 range_min        # minimum range value [m]
float32 range_max        # maximum range value [m]

float32[] ranges         # range data [m] (Note: values < range_min or > range_max should be discarded)
float32[] intensities    # intensity data [device-specific units].  If your
                         # device does not provide intensities, please leave
                         # the array empty.
```

Message field types

- **Examples from** geometry_msgs **package**
  - Pose.msg

```
# A representation of pose in free space, composed of postion and orientation.
Point position
Quaternion orientation
```

  - PoseWithCovariance.msg

```
# This represents a pose in free space with uncertainty.

Pose pose

# Row-major representation of the 6x6 covariance matrix
# The orientation parameters use a fixed-axis representation.
# In order, the parameters are:
# (x, y, z, rotation about X axis, rotation about Y axis, rotation about Z axis)
float64[36] covariance
```

Available message types

- **from** std_msgs **package:**

```
Bool.msg                Header.msg              String.msg
Byte.msg                Int16.msg               Time.msg
ByteMultiArray.msg      Int16MultiArray.msg     UInt16.msg
Char.msg                Int32.msg               UInt16MultiArray.msg
ColorRGBA.msg           Int32MultiArray.msg     UInt32.msg
Duration.msg            Int64.msg               UInt32MultiArray.msg
Empty.msg               Int64MultiArray.msg     UInt64.msg
Float32.msg             Int8.msg                UInt64MultiArray.msg
Float32MultiArray.msg   Int8MultiArray.msg      UInt8.msg
Float64.msg             MultiArrayDimension.msg UInt8MultiArray.msg
Float64MultiArray.msg   MultiArrayLayout.msg
```

Available message types

- **from** geometry_msgs **package:**

```
Point.msg                          QuaternionStamped.msg
Point32.msg                        Transform.msg
PointStamped.msg                   TransformStamped.msg
Polygon.msg                        Twist.msg
PolygonStamped.msg                 TwistStamped.msg
Pose.msg                           TwistWithCovariance.msg
Pose2D.msg                         TwistWithCovarianceStamped.msg
PoseArray.msg                      Vector3.msg
PoseStamped.msg                    Vector3Stamped.msg
PoseWithCovariance.msg             Wrench.msg
PoseWithCovarianceStamped.msg      WrenchStamped.msg
Quaternion.msg
```

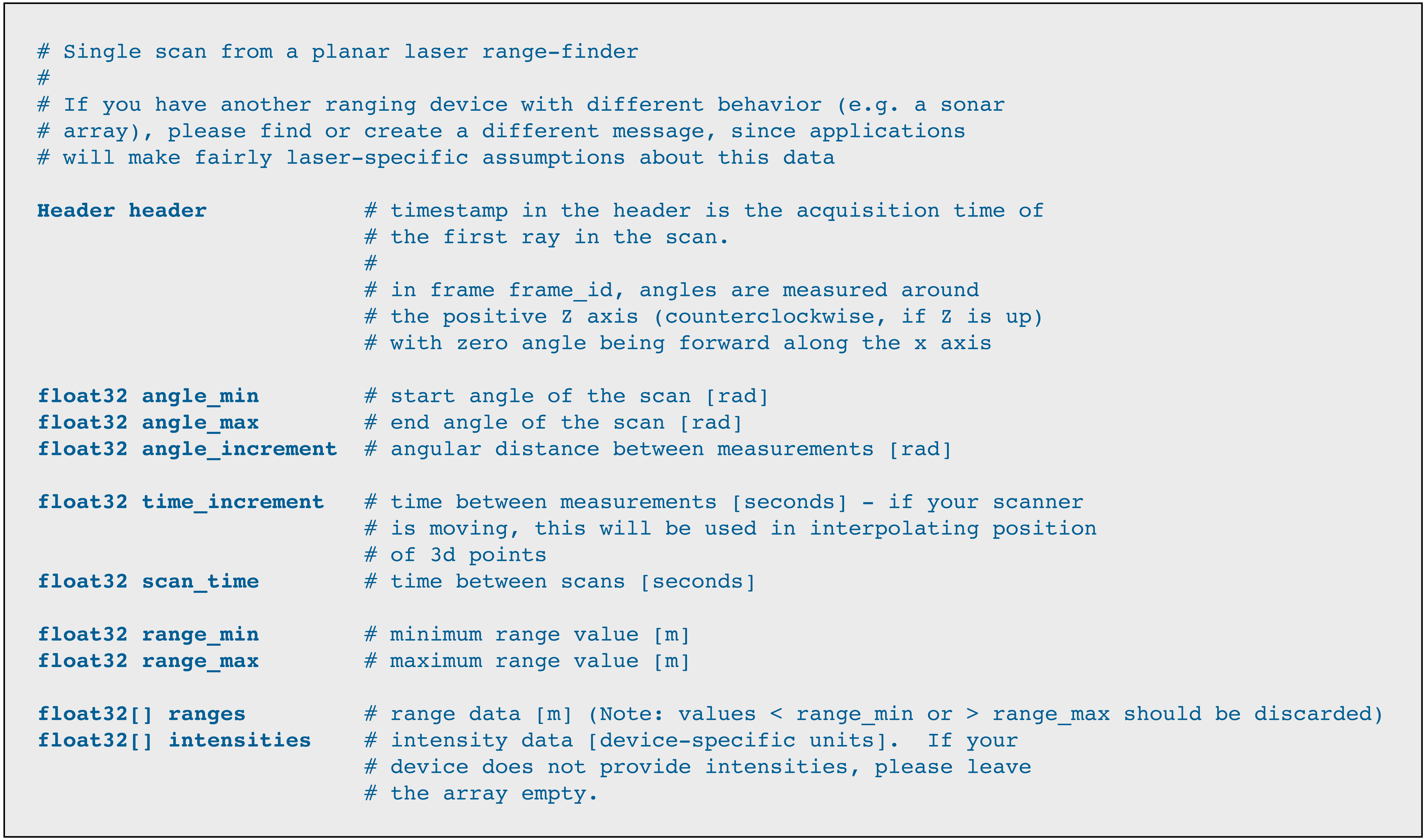
Available message types

- **from** sensor_msgs **package:**

```
CameraInfo.msg         JoyFeedback.msg        PointCloud2.msg
ChannelFloat32.msg     JoyFeedbackArray.msg   PointField.msg
CompressedImage.msg    LaserEcho.msg          Range.msg
FluidPressure.msg      LaserScan.msg          RegionOfInterest.msg
Illuminance.msg        MagneticField.msg      RelativeHumidity.msg
Image.msg              MultiEchoLaserScan.msg Temperature.msg
Imu.msg                NavSatFix.msg          TimeReference.msg
JointState.msg         NavSatStatus.msg
Joy.msg                PointCloud.msg
```

Available message types

- **Example from** sensor_msgs **package:** LaserScan.msg



```
# Single scan from a planar laser range-finder
#
# If you have another ranging device with different behavior (e.g. a sonar
# array), please find or create a different message, since applications
# will make fairly laser-specific assumptions about this data

Header header            # timestamp in the header is the acquisition time of
                         # the first ray in the scan.
                         #
                         # in frame frame_id, angles are measured around
                         # the positive Z axis (counterclockwise, if Z is up)
                         # with zero angle being forward along the x axis

float32 angle_min        # start angle of the scan [rad]
float32 angle_max        # end angle of the scan [rad]
float32 angle_increment  # angular distance between measurements [rad]

float32 time_increment   # time between measurements [seconds] - if your scanner
                         # is moving, this will be used in interpolating position
                         # of 3d points
float32 scan_time        # time between scans [seconds]

float32 range_min        # minimum range value [m]
float32 range_max        # maximum range value [m]

float32[] ranges         # range data [m] (Note: values < range_min or > range_max should be discarded)
float32[] intensities    # intensity data [device-specific units].  If your
                         # device does not provide intensities, please leave
                         # the array empty.
```

Service type definition

- Similarly to messages, service types are defined by simple text files in the srv/ directory
- Syntax:

```
# this is a comment
fieldtype1 request_fieldname1
fieldtype2 request_ fieldname2
...
---
fieldtype1 response_fieldname1
fieldtype2 response_ fieldname2
...
```

Available service types

- **Example from** std_srvs **package:** Empty.srv

```
---
```

- **Example from** sensor_msgs **package:** SetCameraInfo.srv

```
# This service requests that a camera stores the given CameraInfo
# as that camera's calibration information.
#
# The width and height in the camera_info field should match what the
# camera is currently outputting on its camera_info topic, and the camera
# will assume that the region of the imager that is being referred to is
# the region that the camera is currently capturing.

sensor_msgs/CameraInfo camera_info # The camera_info to store
---
bool success           # True if the call succeeded
string status_message # Used to give details about success
```

Example service type

- From ISR-CoBot:
  - service to control Guidance module

```
uint8   cmd

# SET GOAL
#   cmd=1 for direct
#   cmd=2 for path planning
float32 x
float32 y
float32 t

---

bool success
string result
```