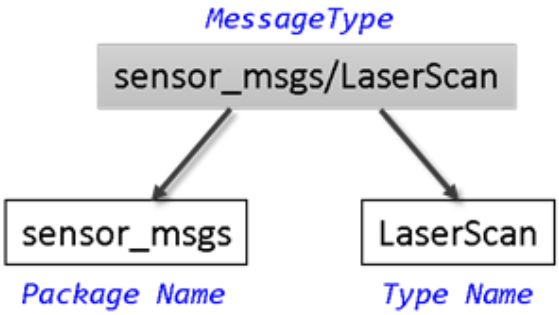
Work with Basic ROS Messages

Introduction

Try it in MATLAB

Messages are the primary container for exchanging data in ROS. Topics (see Exchange Data with ROS Publishers and Subscribers) and services (see Call and Provide ROS Services) use messages to carry data between nodes.

To identify its data structure, each message has a *message type*. For example, sensor data from a laser scanner is typically sent in a message of type `sensor_msgs/LaserScan`. Each message type identifies the data elements that are contained in a message. Every message type name is a combination of a package name, followed by a forward slash /, and a type name:



MATLAB® supports many ROS message types that are commonly encountered in robotics applications. In this example, you will examine some of the ways to create, explore, and populate ROS messages in MATLAB.

Prerequisites: Get Started with ROS, Connect to a ROS Network

Find Message Types

- Initialize the ROS master and global node.

```
rosinit
```

```
Initializing ROS master on http://bat6305glnxa64:45346/.
Initializing global node /matlab_global_node_91072 with NodeURI http://bat6305glnxa64:35325/
```

- Use `exampleHelperROSCreateSampleNetwork` to populate the ROS network with three additional nodes and sample publishers and subscribers.

```
exampleHelperROSCreateSampleNetwork
```

- There are various nodes on the network with a few topics and affiliated publishers and subscribers.
- You can see the full list of available topics by calling `rostopic list`. `/scan` is one of topics that is listed.

```
rostopic list
```

```
/pose
/rosout
/scan
```

- If you want to know more about the type of data that is sent through the `/scan` topic, use the `rostopic info /scan` command to examine it. `/scan` has a message type of `sensor_msgs/LaserScan`.

```
rostopic info /scan
```

```
Type: sensor_msgs/LaserScan

Publishers:
* /node_3 (http://bat6305glnxa64:39005/)
```

```
Subscribers:
* /node_2 (http://bat6305glnxa64:37099/)
* /node_1 (http://bat6305glnxa64:43907/)
```

The command output also tells you which nodes are publishing and subscribing to the topic. To learn about publishers and subscribers, see Exchange Data with ROS Publishers and Subscribers.

- To find out more about the topic's message type, create an empty message of the same type. Use the `rosmessage` function. `rosmessage` supports tab completion for the message type. To complete message type names, type the first few characters of the name you want to complete, and then press the **Tab** key.

```
scandata = rosmessage('sensor_msgs/LaserScan')
```

```
scandata =

  ROS LaserScan message with properties:

       MessageType: 'sensor_msgs/LaserScan'
            Header: [1x1 Header]
          AngleMin: 0
          AngleMax: 0
    AngleIncrement: 0
     TimeIncrement: 0
          ScanTime: 0
          RangeMin: 0
          RangeMax: 0
            Ranges: [0x1 single]
      Intensities: [0x1 single]

  Use showdetails to show the contents of the message
```

The created message `scandata` has many properties associated with data typically received from a laser scanner. For example, the minimum sensing distance is stored in the `RangeMin` property and the maximum sensing distance in `RangeMax`.

To see a complete list of all message types available for topics and services, use the `rosmsg list` command:

```
rosmsg list
```

```
ackermann_msgs/AckermannDrive
ackermann_msgs/AckermannDriveStamped
actionlib/TestAction
actionlib/TestActionFeedback
actionlib/TestActionGoal
actionlib/TestActionResult
actionlib/TestFeedback
actionlib/TestGoal
actionlib/TestRequestAction
actionlib/TestRequestActionFeedback
actionlib/TestRequestActionGoal
actionlib/TestRequestActionResult
actionlib/TestRequestFeedback
actionlib/TestRequestGoal
actionlib/TestRequestResult
actionlib/TestResult
actionlib/TwoIntsAction
actionlib/TwoIntsActionFeedback
actionlib/TwoIntsActionGoal
actionlib/TwoIntsActionResult
...
```

Explore Message Structure and Get Message Data

ROS messages are objects and the message data is stored in properties. MATLAB® features convenient ways to find and explore the contents of messages.

- If you subscribe to the `/pose` topic, you can receive and examine the messages that are sent.

```
posesub = rossubscriber('/pose')
```

```
posesub =

  Subscriber with properties:

        TopicName: '/pose'
      MessageType: 'geometry_msgs/Twist'
    LatestMessage: [0x1 Twist]
       BufferSize: 1
    NewMessageFcn: []
```

- Use `receive` to acquire data from the subscriber. Once a new message is received, the function will return it and store it in the `posedata` variable (the second argument is a time-out in seconds).

```
posedata = receive(posesub, 10)
```

```
posedata =

  ROS Twist message with properties:

    MessageType: 'geometry_msgs/Twist'
         Linear: [1x1 Vector3]
        Angular: [1x1 Vector3]

  Use showdetails to show the contents of the message
```

- The message has a type of `geometry_msgs/Twist`. There are two other fields in the message: `Linear` and `Angular`. You can see the values of these message fields by accessing them directly:

```
posedata.Linear
```

```
ans =

  ROS Vector3 message with properties:

    MessageType: 'geometry_msgs/Vector3'
              X: 0.0093
              Y: 0.0453
              Z: 0.0084

  Use showdetails to show the contents of the message
```

```
posedata.Angular
```

```
ans =

  ROS Vector3 message with properties:

    MessageType: 'geometry_msgs/Vector3'
              X: 0.0878
              Y: -0.0210
              Z: 0.0068

  Use showdetails to show the contents of the message
```

You can see that each of the values of these message fields is actually a message in itself. The message type for these is `geometry_msgs/Vector3`. `geometry_msgs/Twist` is a composite message made up of two `geometry_msgs/Vector3` messages.

- Data access for these nested messages works exactly the same as accessing the data in other messages. Access the `X` component of the `Linear` message using this command:

```
xpos = posedata.Linear.X
```

```
xpos =

    0.0093
```

- If you want a quick summary of all the data contained in a message, you can call the `showdetails` function. `showdetails` works on messages of any type and will recursively display all the message data properties.

```
showdetails(posedata)
```

```
  Linear
    X :  0.00932219052602273
    Y :  0.04526734562806031
    Z :  0.008449365172780367
  Angular
    X :  0.08782643913382959
    Y :  -0.02100276720970422
    Z :  0.006813318676002524
```

`showdetails` helps you during debugging and when you want to quickly explore the contents of a message.

Set Message Data

- You can also set message property values. Create a message with type `geometry_msgs/Twist`.

```
twist = rosmessage('geometry_msgs/Twist')
```

```
twist =

  ROS Twist message with properties:

    MessageType: 'geometry_msgs/Twist'
         Linear: [1x1 Vector3]
        Angular: [1x1 Vector3]

  Use showdetails to show the contents of the message
```

- The numeric properties of this message are initialized to "0" by default. You can modify any of the properties of this message. Make the `Linear.Y` entry equal to 5.

```
twist.Linear.Y = 5;
```

- You can view the message data to make sure that your change took effect.

```
twist.Linear
```

```
ans =

  ROS Vector3 message with properties:

    MessageType: 'geometry_msgs/Vector3'
              X: 0
              Y: 5
              Z: 0

  Use showdetails to show the contents of the message
```

Once a message is populated with your data, you can use it with publishers, subscribers, and services. See the Exchange Data with ROS Publishers and Subscribers and Call and Provide ROS Services examples.

Copy Messages

There are two ways to copy the contents of a message:

- You can create a *reference copy* in which the copy and the original messages share the same data
- You can create a *deep copy* in which the copy and the original messages each have their own data.

A reference copy is useful if you want to share message data between different functions or objects, whereas a deep copy is necessary if you want an independent copy of a message.

- Make a *reference copy* of a message by using the '=' sign. This creates a variable that references the same message contents as the original variable.

```
twistCopyRef = twist
```

```
twistCopyRef =

  ROS Twist message with properties:

    MessageType: 'geometry_msgs/Twist'
         Linear: [1x1 Vector3]
        Angular: [1x1 Vector3]

  Use showdetails to show the contents of the message
```

- Modify the `Linear.Z` field of `twistCopyRef`, and see that it changes the contents of `twist` as well:

```
twistCopyRef.Linear.Z = 7;
```

```
twist.Linear
```

```
ans =

  ROS Vector3 message with properties:

    MessageType: 'geometry_msgs/Vector3'
              X: 0
              Y: 5
              Z: 7

  Use showdetails to show the contents of the message
```

- Make a *deep copy* of `twist` so that you can change its contents without affecting the original data. Make a new message, `twistCopyDeep`, using the `copy` function:

```
twistCopyDeep = copy(twist)
```

```
twistCopyDeep =

  ROS Twist message with properties:

    MessageType: 'geometry_msgs/Twist'
         Linear: [1x1 Vector3]
        Angular: [1x1 Vector3]

  Use showdetails to show the contents of the message
```

- Modify the `Linear.X` property of `twistCopyDeep` and notice that the contents of `twist` remain unchanged.

```
twistCopyDeep.Linear.X = 100;
```

```
twistCopyDeep.Linear
```

```
ans =

  ROS Vector3 message with properties:

    MessageType: 'geometry_msgs/Vector3'
              X: 100
              Y: 5
              Z: 7

  Use showdetails to show the contents of the message
```

```
twist.Linear
```

```
ans =

  ROS Vector3 message with properties:

    MessageType: 'geometry_msgs/Vector3'
              X: 0
              Y: 5
              Z: 7

  Use showdetails to show the contents of the message
```

Save and Load Messages

You can save messages and store the contents for later use.

- Get a new message from the subscriber:

```
posedata = receive(posesub,10)
```

```
posedata =

  ROS Twist message with properties:

    MessageType: 'geometry_msgs/Twist'
         Linear: [1x1 Vector3]
        Angular: [1x1 Vector3]

  Use showdetails to show the contents of the message
```

- Save the pose data to a .mat file using MATLAB's `save` function.

```
save('posedata.mat','posedata')
```

- Before loading the file back into the workspace, clear the `posedata` variable.

```
clear posedata
```

- Now you can load the message data by calling the `load` function. This loads the `posedata` from above into the `messageData` structure. `posedata` is a data field of the struct.

```
messageData = load('posedata.mat')
```

```
messageData =

  struct with fields:

    posedata: [1x1 Twist]
```

- Examine `messageData.posedata` to see the message contents.

```
messageData.posedata
```

```
ans =

  ROS Twist message with properties:

    MessageType: 'geometry_msgs/Twist'
         Linear: [1x1 Vector3]
        Angular: [1x1 Vector3]

  Use showdetails to show the contents of the message
```

- You can now delete the MAT file with

```
delete('posedata.mat')
```

Object Arrays in Messages

Some messages from ROS are stored in Object Arrays (MATLAB). These should be handled differently from typical data arrays.

- In your workspace, the variable `tf` contains a sample message (the `exampleHelperROSCreateSampleNetwork` script created the variable). In this case, it is a message of type `tf/tfMessage` used for coordinate transformations.

```
tf
```

```
tf =

  ROS tfMessage message with properties:

    MessageType: 'tf/tfMessage'
     Transforms: [53x1 TransformStamped]

  Use showdetails to show the contents of the message
```

`tf` has two fields: `MessageType` contains a standard data array and `Transforms` contains an object array. There are 53 objects stored in `Transforms`, and all of them have the same structure.

- Expand `tf` in `Transforms` to see the structure:

```
tf.Transforms
```

```
ans =

  53x1 ROS TransformStamped message array with properties:

    MessageType
    Header
    ChildFrameId
    Transform
```

- Each object in `Transforms` has four properties. Expand to see the `Transform` field of `Transforms`.

```
tf.Transforms.Transform
```

```
ans =

  ROS Transform message with properties:

    MessageType: 'geometry_msgs/Transform'
    Translation: [1x1 Vector3]
       Rotation: [1x1 Quaternion]

  Use showdetails to show the contents of the message
```

```
ans =

  ROS Transform message with properties:

    MessageType: 'geometry_msgs/Transform'
    Translation: [1x1 Vector3]
       Rotation: [1x1 Quaternion]

...
```

- Notice that the output returns 53 individual answers, since each object is evaluated and returns the value of its `Transform` field. This format is not always useful, so you can convert it to a cell array with the following command:

```
cellTransforms = {tf.Transforms.Transform}
```

```
cellTransforms =

  1x53 cell array

  Columns 1 through 4

    {1x1 Transform}    {1x1 Transform}    {1x1 Transform}    {1x1 Transform}

  Columns 5 through 8

    {1x1 Transform}    {1x1 Transform}    {1x1 Transform}    {1x1 Transform}

  Columns 9 through 12

    {1x1 Transform}    {1x1 Transform}    {1x1 Transform}    {1x1 Transform}

  Columns 13 through 16

    {1x1 Transform}    {1x1 Transform}    {1x1 Transform}    {1x1 Transform}

...
```

This puts all 53 object entries in a cell array, which allows you to access them with indexing.

- In addition, you can access object array elements the same way you access standard MATLAB vectors:

```
tf.Transforms(5)
```

```
ans =

  ROS TransformStamped message with properties:

     MessageType: 'geometry_msgs/TransformStamped'
          Header: [1x1 Header]
    ChildFrameId: '/imu_link'
       Transform: [1x1 Transform]

  Use showdetails to show the contents of the message
```

- You can access the properties of individual array elements:

```
tf.Transforms(5).Transform.Translation
```

```
ans =

  ROS Vector3 message with properties:

    MessageType: 'geometry_msgs/Vector3'
              X: 0.0599
```

```
    Y: 0
    Z: -0.0141

  Use showdetails to show the contents of the message
```

This exposes the translation component of the fifth transform in the list of 53.

Shut Down ROS Network

- Remove the sample nodes, publishers, and subscribers from the ROS network.

```
exampleHelperROSShutDownSampleNetwork
```

- Shut down the ROS master and delete the global node.

```
rosshutdown
```

```
Shutting down global node /matlab_global_node_91072 with NodeURI http://bat6305glnxa64:35325/
Shutting down ROS master on http://bat6305glnxa64:45346/.
```

Next Steps

- See Work with Specialized ROS Messages for examples of handling images, point clouds, and laser scan messages.
- For application examples, see the Get Started with Gazebo and a Simulated TurtleBot or Get Started with a Real TurtleBot examples.