ROS Documentation

# Initialising ROS:

Ros core has to be initialised/run on a terminal before starting any other process.

command : roscore

# Topics:

Nodes communicate on a topic in ros.

rostopic bw display bandwidth used by topic

rostopic delay display delay for topic which has header

rostopic echo print messages to screen

rostopic find find topics by type

rostopic hz display publishing rate of topic

rostopic info print information about active topic

rostopic list print information about active topics

rostopic pub publish data to topic

rostopic type print topic type

Resource: <http://wiki.ros.org/rostopic>

## Rqt Graph:

Rqt graphs can be used to visualise the nodes, subscribers etc on a given topic.

# Nodes:

They can be considered as the basic building block of any ROS based application. they can contain publishers, subscribers , services and clients.

**Initialising a node:**

rospy.init\_node(‘node\_name’, anonymous=True)

\*anonymous command creates a randomised name for the node to keep its name unique.

## Publisher:

A publisher publishes the required data on a rostopic.

**Initialising a publisher:**

publisher = rospy.Publisher(‘topic\_name’, data\_type, queue\_size=10).

**Publishing data using a publisher:**

publisher.publish(message)

Data can be published at a particular rate using the rate function in rospy as follows:

Rate = rospy.rate(Hz)

Rate.sleep() //Creates a delay of desired time in the loop.

## Subscriber:

It subscribes to a particular topic and receives the data (Messages) published on it.

**Initialising a subscriber:**

rospy.Subscriber('topic\_name', data\_type, callback\_function)

A subscriber requires a predefined callback function for processing the received data.

## Node Activation:

A node can be activated in the terminal using the following command:

rosrun /topic\_name node\_name

## Spin functionality:

It runs the subscriber/receiver node until its terminated by the used (it makes the node persistent).

Command: rospy.spin()

# Messages:

Notes communicated on a topic using messages, they can be considered as the data packets used for communication within ROS.Standard message types: string, float, boolean, array, char, time etc.

Resource: <http://wiki.ros.org/std_msgs>

**Publishing a message on a topic:**

A message can be published on a running topic without the need of initialising the node (Given the data types of the message and topic are the same).

rostopic pub -r rate /topic\_name message/type message

The publishing of data through a messages depends on the parameters accepted by it.

Eg: A String message can only transfer string data type.

Message = String()

Message.data = ‘It’s a string.’

For creating a message data type, an object of the required datatype has to be instantiated.

# Service:

Services are used for bi-directional data flow on a topic. eg: Suppose a weather service is created and a client makes a request, the server can receive the request as well as transmit the desired data back to the client.

Resource: <http://wiki.ros.org/Services>

## Server:

A ROS server accepts the client’s request and transmits the data accordingly.

Initialisation:

service = rospy.Service('/Service\_Name',Service\_Type,handler)

Handler function is required to handle the service request and transmit/return the desired data. It receives the client request as the parameter and returns the appropriate data.

There are predefined services in ROS like Addition of the input numbers, but can also be customised.

## Client:

A client transmits data/request to the server and waits until it gets a response from it.

Hence the services in ROS are used for quick communication purposes, as it suspends the client activity until it gets a response from the server.

Initialisation:

client\_name = rospy.ServiceProxy("/",Service\_Type)

response = Service\_Name (requested\_data)

# Log:

Logging functionality can be used to print/log a message/command on the running terminal.

**Logging info:**

rospy.loginfo(data)

# Custom Messages:

Custom message definitions can be created in ROS in the .msg format.

Example:

float64 temprature

bool motor\_status

string debug\_msg

The message parameters can be set using the variable names specified in the .msg file.

\*The message file has to be created in the msg folder of a package.

Eg: message.temprature = 102.04

## Custom Message Compilation:

Custom message definitons need to be compiled in the CMakeLists.txt and package.xml file.

**Changes to be made in CMakeLists.txt:**

1: Adding message generation in find\_package

find\_package(catkin REQUIRED COMPONENTS

roscpp

rospy

std\_msgs

message\_generation **# Adding message generation in find\_package**

**)**

2. Adding message file name in add\_message\_file (The message file should be in msg folder of the package)

add\_message\_files(

FILES

HardwareStatus.msg **# Adding message file name in add\_message\_file**

)

3. Uncommenting generate\_messages

generate\_messages(

DEPENDENCIES

std\_msgs

)

4. Adding message\_runtime to catkin\_depend

catkin\_package(

# INCLUDE\_DIRS include

# LIBRARIES Robot\_Messages

CATKIN\_DEPENDS roscpp rospy std\_msgs message\_runtime

# DEPENDS system\_lib

)

**Changes to be made in Package.xml:**

1. Adding message\_generation and message\_runtime:

<build\_depend>message\_generation</build\_depend>

<exec\_depend>message\_runtime</exec\_depend>