

[← Back \(/tutorials?cat=connect_ros\)](#)

ROS communication

Tutorial: ROS Communication

Gazebo provides a set of ROS API's that allows users to modify and get information about various aspects of the simulated world. In the following sections, we will demonstrate some of the utilities for manipulating the simulation world and objects. The complete list of ROS messages and services for gazebo can be found [here](#) also.

Prerequisites

If you would like to follow along with the examples make sure you have the RRBot setup as described in the [Using URDF in Gazebo \(http://gazebosim.org/tutorials/?tut=ros_urdf\)](http://gazebosim.org/tutorials/?tut=ros_urdf). In this tutorial we'll have the RRBot "kick" a coke can using various techniques.

We'll assume you have Gazebo already launched using:

```
roscore &
roslaunch gazebo_ros gazebo
```

You may occasionally need to restart Gazebo after different commands listed below.

Terminologies

In the following context, the pose and twist of a rigid body object is referred to as its "state". An object also has intrinsic "properties", such as mass and friction coefficients. In Gazebo, a "body" refers to a rigid body, synonymous to "link" in the URDF context. A Gazebo "model" is a conglomeration of bodies connected by "joints".

About the gazebo_ros_api_plugin

The `gazebo_ros_api_plugin` plugin, located with the `gazebo_ros` package, initializes a ROS node called "gazebo". It integrates the ROS callback scheduler (message passing) with Gazebo's internal scheduler to provide the ROS interfaces described below. This ROS API enables a user to manipulate the properties of the simulation environment over ROS, as well as spawn and introspect on the state of models in the environment.

This plugin is only loaded with `gzserver`.

About the gazebo_ros_paths_plugin

A secondary plugin named `gazebo_ros_paths_plugin` is available in the `gazebo_ros` package that simply allows Gazebo to find ROS resources, i.e. resolving ROS package path names.

This plugin is loaded with both `gzserver` and `gzclient`.

Gazebo Published Parameters

Parameters:

`/use_sim_time` : **Bool** - Notifies ROS to use published `/clock` topic for ROS time.

Gazebo uses the ROS parameter server to notify other applications, particularly Rviz, if simulation time should be used via the `/use_sim_time` parameter. This should be set automatically by Gazebo as true when you start `gazebo_ros`

`/use_sim_time` is true if gazebo_ros is publishing to the ROS `/clock` topic in order to provide a ROS system with simulation-synchronized time. For more info on simulation time, see ROS C++ Time (<http://www.ros.org/wiki/roscpp/Overview/Time>).

Checking the value

To see what the parameter is set as run:

```
rosparam get /use_sim_time
```

Gazebo Subscribed Topics

Topics:

`~/set_link_state` : `gazebo_msgs/LinkState` - Sets the state (pose/twist) of a link.

`~/set_model_state` : `gazebo_msgs/ModelState` - Sets the state (pose/twist) of a model.

Set Model Pose and Twist in Simulation via Topics

Topics (<http://www.ros.org/wiki/Topics>) can be used to set the pose and twist of a model rapidly without waiting for the pose setting action to complete. To do so, publish the desired model state message (<http://www.ros.org/doc/api/gazebo/html/msg/ModelState.html>) to `/gazebo/set_model_state` topic. For example, to test pose setting via topics, add a coke can to the simulation by spawning a new model from the online database:

```
roslaunch gazebo_ros spawn_model -database coke_can -gazebo -model coke_can -y 1
```

and set the pose of the coke can by publishing on the `/gazebo/set_model_state` topic:

```
rostopic pub -r 20 /gazebo/set_model_state gazebo_msgs/ModelState '{model_name: coke_can, pose: { position: { x: 1, y: 0, z: 2 }, orientation: {x: 0, y: 0.491983115673, z: 0, w: 0.870604813099 } }, twist: { linear: { x: 0, y: 0, z: 0 }, angular: { x: 0, y: 0, z: 0 } }, reference_frame: world }'
```

You should see the coke can hovering in front of the RRBot, just asking to be hit (we'll get there).



Gazebo Published Topics

Topics:

/clock : rosgraph_msgs /**Clock** - Publish simulation time, to be used with /use_sim_time parameter.

~/link_states : gazebo_msgs /**LinkStates** - Publishes states of all the links in simulation.

~/model_states : gazebo_msgs /**ModelState** - Publishes states of all the models in simulation.

Retrieving Model and Link States Using Topics

Gazebo publishes /gazebo/link_states and /gazebo/model_states topics, containing pose and twist information of objects in simulation with respect to the gazebo world frame. You can see these in action by running:

```
rostopic echo -n 1 /gazebo/model_states
```

or

```
rostopic echo -n 1 /gazebo/link_states
```

To reiterate, a "link" is defined as a rigid body with given inertial, visual and collision properties. Whereas, a "model" is defined as a collection of links and joints. The state of a "model" is the state of its canonical "link". Given that URDF enforces a tree structure, the canonical link of a model is defined by its root link.

Services: Create and destroy models in simulation

These services allow the user to spawn and destroy models dynamically in simulation:

~/spawn_urdf_model : gazebo_msgs /**SpawnModel** - Use this service to spawn a Universal Robotic Description Format (URDF)

~/spawn_sdf_model : gazebo_msgs /**SpawnModel** - Use this service to spawn a model written in Gazebo Simulation Description Format (SDF)

~/delete_model : gazebo_msgs /**DeleteModel** - This service allows the user to delete a model from simulation.

Spawn Model

A helper script called spawn_model is provided for calling the model spawning services offered by gazebo_ros. The most practical method for spawning a model using the service call method is with a roslaunch file. Details are provided in the tutorial Using roslaunch Files to Spawn Models (http://gazebo.org/tutorials/?tut=ros_roslaunch). There are many ways to use spawn_model to add URDFs and SDFs to Gazebo. The following are a few of the examples:

Spawn a URDF from file - first convert .xacro file to .xml then spawn:

```
roslaunch xacro xacro `rospack find rrbot_description`/urdf/rrbot.xacro >> `rospack find rrbot_description`/urdf/rrbot.xml
roslaunch gazebo_ros spawn_model -file `rospack find rrbot_description`/urdf/rrbot.xml -urdf -y 1 -model rrbot1 -robot_namespace rrbot
1
```

URDF from parameter server using roslaunch and xacro: See Using roslaunch Files to Spawn Models (http://gazebo.org/tutorials/?tut=ros_roslaunch)

SDF from local model database:

```
roslaunch gazebo_ros spawn_model -file `echo $GAZEBO_MODEL_PATH`/coke_can/model.sdf -sdf -model coke_can1 -y 0.2 -x -0.3
```

SDF from the online model database:

```
roslaunch gazebo_ros spawn_model -database coke_can -sdf -model coke_can3 -y 2.2 -x -0.3
```

To see all of the available arguments for spawn_model including namespaces, trimesh properties, joint positions and RPY orientation run:

```
roslaunch gazebo_ros spawn_model -h
```

Delete Model

Deleting models that are already in Gazebo is easier as long as you know the model name you gave the object. If you spawned a rrbot named "rrbot1" as described in the previous section, you can remove it with:

```
rosservice call gazebo/delete_model '{model_name: rrbot1}'
```

Services: State and property setters

These services allow the user to set state and property information about simulation and objects in simulation:

~/set_link_properties : gazebo_msgs /**SetLinkProperties**

~/set_physics_properties : gazebo_msgs /**SetPhysicsProperties**

```
~/set_model_state : gazebo_msgs/SetModelState
~/set_model_configuration : gazebo_msgs/SetModelConfiguration - This service allows the user to set model joint positions without invoking dynamics.
~/set_joint_properties : gazebo_msgs/SetJointProperties
~/set_link_state : gazebo_msgs/SetLinkState
~/set_link_state : gazebo_msgs/LinkState
~/set_model_state : gazebo_msgs/ModelState
```

Set Model State Example

Let's have some fun and have the RRBot hit a coke can using the `/gazebo/set_model_state` service.

If you have not already added a coke can to your simulation run

```
roslaunch gazebo_ros spawn_model -database coke_can -gazebo -model coke_can -y 1
```

This should be prepackaged with Gazebo or available via the online model database (internet connection required). Place the coke can anywhere in the scene, it doesn't matter where. Now we'll call a service request to move the coke can into position of the RRBot:

```
rosservice call /gazebo/set_model_state '{model_state: { model_name: coke_can, pose: { position: { x: 0.3, y: 0.2, z: 0 }, orientation: {x: 0, y: 0.491983115673, z: 0, w: 0.870604813099 } }, twist: { linear: {x: 0.0, y: 0, z: 0 }, angular: { x: 0.0, y: 0, z: 0.0 } } , reference_frame: world } }'
```

You should see the setup like this:



Now get the RRBot to start spinning using the following command:

```
rosservice call /gazebo/set_model_state '{model_state: { model_name: rrbot, pose: { position: { x: 1, y: 1, z: 10 }, orientation: {x: 0, y: 0.491983115673, z: 0, w: 0.870604813099 } }, twist: { linear: {x: 0.0, y: 0, z: 0 }, angular: { x: 0.0, y: 0, z: 0.0 } } , reference_frame: world } }'
```

With luck you'll have the coke can launched some variable distance :) If it fumbles you can always try again, this is a tutorial you know.



Services: State and property getters

These services allow the user to retrieve state and property information about simulation and objects in simulation:

```
~/get_model_properties : gazebo_msgs/GetModelProperties - This service returns the properties of a model in simulation.
~/get_model_state : gazebo_msgs/GetModelState - This service returns the states of a model in simulation.
~/get_world_properties : gazebo_msgs/GetWorldProperties - This service returns the properties of the simulation world.
~/get_joint_properties : gazebo_msgs/GetJointProperties - This service returns the properties of a joint in simulation.
~/get_link_properties : gazebo_msgs/GetLinkProperties - This service returns the properties of a link in simulation.
~/get_link_state : gazebo_msgs/GetLinkState - This service returns the states of a link in simulation.
~/get_physics_properties : gazebo_msgs/GetPhysicsProperties - This service returns the properties of the physics engine used in simulation.
~/link_states : gazebo_msgs/LinkStates - Publish complete link states in world frame
~/model_states : gazebo_msgs/ModelStates - Publish complete model states in world frame
```

Note:

`link_names` are in gazebo scoped name notation, `[model_name::body_name]`

Get Model State Example

Now that you've "kicked" the coke can some distance, you'll want to know how far it went. Building off of the previous example (with the same simulation running), we'll query the pose and twist of the coke can by using the service call:

```
rosservice call gazebo/get_model_state '{model_name: coke_can}'
```

Which depending on your robot's kicking skill could give you something like:

```
pose:
  position:
    x: -10.3172263825
    y: -1.95098702647
    z: -0.00413857755159
  orientation:
    x: -0.0218349987011
    y: -0.00515029763403
    z: 0.545795377598
    w: 0.83761811887
twist:
  linear:
    x: -0.000385525262354
    y: -0.000344915539911
    z: -0.00206406538336
  angular:
    x: -0.104256200218
    y: 0.0370371098566
    z: 0.0132837766211
success: True
```

My robot kicked the can 10 meters, how did you do?

Retrieving Simulation World and Object Properties

You can get a list of models (ground_plane, coke cane, rrbot) in the world by running:

```
rosservice call gazebo/get_world_properties
```

```
sim_time: 1013.366
model_names: ['ground_plane', 'rrbot', 'coke_can']
rendering_enabled: True
success: True
status_message: GetWorldProperties: got properties
```

and retrieve details of a specific model by

```
rosservice call gazebo/get_model_properties '{model_name: rrbot}'
```

```
parent_model_name: ''
canonical_body_name: ''
body_names: ['link1', 'link2', 'link3']
geom_names: ['link1_geom', 'link2_geom', 'link3_geom', 'link3_geom_camera_link', 'link3_geom_hokuyo_link']
joint_names: ['fixed', 'joint1', 'joint2']
child_model_names: []
is_static: False
success: True
status_message: GetModelProperties: got properties
```

Services: Force control

These services allow the user to apply wrenches and forces to bodies and joints in simulation:

- ~/apply_body_wrench : gazebo_msgs / **ApplyBodyWrench** - Apply wrench to a body in simulation. All active wrenches applied to the same body are cumulative.
- ~/apply_joint_effort : gazebo_msgs / **ApplyJointEffort** - Apply effort to a joint in simulation. All active efforts applied to the same joint are cumulative.
- ~/clear_joint_forces : gazebo_msgs / **JointRequest** - Clear applied efforts to a joint.
- ~/clear_body_wrenches : gazebo_msgs / **ClearBodyWrenches** - Clear applied wrench to a body.

Apply Wrenches to Links

To demonstrate wrench applications on a Gazebo body, let's spawn an object with gravity turned off. Make sure the coke can has been added to the simulation:

```
roslaunch gazebo_ros spawn_model -database coke_can -gazebo -model coke_can -y 1
```

Then to turn off gravity send a service call to /gazebo/set_physics_properties with no gravity in any of the axis:

```
rosservice call /gazebo/set_physics_properties "
time_step: 0.001
max_update_rate: 1000.0
gravity:
  x: 0.0
  y: 0.0
  z: 0.0
ode_config:
  auto_disable_bodies: False
  sor_pgs_precon_iters: 0
  sor_pgs_iters: 50
  sor_pgs_w: 1.3
  sor_pgs_rms_error_tol: 0.0
  contact_surface_layer: 0.001
  contact_max_correcting_vel: 100.0
  cfm: 0.0
  erp: 0.2
  max_contacts: 20"
```

Apply a 0.01 Nm torque at the coke can origin for 1 second duration by calling the `/gazebo/apply_body_wrench` service, and you should see the coke can spin up along the positive x-axis:

```
rosservice call /gazebo/apply_body_wrench '{body_name: "coke_can::link" , wrench: { torque: { x: 0.01, y: 0 , z: 0 } }, start_time:
10000000000, duration: 1000000000 }'
```

You should see your coke can spinning:



Apply a reverse -0.01 Nm torque for 1 second duration at the coke can origin and the can should stop rotating:

```
rosservice call /gazebo/apply_body_wrench '{body_name: "coke_can::link" , wrench: { torque: { x: -0.01, y: 0 , z: 0 } }, start_time:
10000000000, duration: 1000000000 }'
```

In general, torques with a negative duration persists indefinitely. To clear any active wrenches applied to the body, you can:

```
rosservice call /gazebo/clear_body_wrenches '{body_name: "coke_can::link"}'
```

Apply Efforts to Joints in Simulation

Call `/gazebo/apply_joint_effort` to apply torque to the joint

```
rosservice call /gazebo/apply_joint_effort "joint_name: 'joint2'
effort: 10.0
start_time:
  secs: 0
  nsecs: 0
duration:
  secs: 10
  nsecs: 0"
```

And the link should start rotating.



To clear efforts on joints for a specific joint, call

```
rosservice call /gazebo/clear_joint_forces '{joint_name: joint2}'
```

Services: Simulation control

These services allow the user to pause and unpause physics in simulation:

`~/pause_physics` : `std_srvs/Empty` - Pause physics updates.

`~/unpause_physics` : `std_srvs/Empty` - Resume physics updates.

`~/reset_simulation` : `std_srvs/Empty` - Resets the entire simulation including the time

`~/reset_world` : `std_srvs/Empty` - Resets the model's poses

Pausing and Unpausing Physics

Say you want to get a good screenshot of your soda can flying in the air. You can pause the physics engine by calling:

```
rosservice call gazebo/pause_physics
```

When simulation is paused, simulation time is stopped and objects become static. However, Gazebo's internal update loop (such as custom dynamic plugin updates) are still running, but given that the simulation time is not changing, anything throttled by simulation time will not update. To resume simulation, unpause the physic engine by calling:



```
rosservice call gazebo/unpause_physics
```

Next Steps

Learn how to create custom ROS plugins for Gazebo (http://gazebosim.org/tutorials/?cat=ros_plugins).

©2014 Open Source Robotics Foundation

Gazebo is open-source licensed under Apache 2.0
(<http://www.apache.org/licenses/LICENSE-2.0.html>)

 ([https://plus.google.com/u/0/115981436296571800301?](https://plus.google.com/u/0/115981436296571800301?prsrc=3)
prsrc=3) 
(<https://www.youtube.com/channel/UCJyqf9XJpDoM9>)