SDFormat: A robot description format in constant evolution

Alejandro Hernández Cordero
Addisu Taddese (Intrinsic)

ROSCon 2023
Table of content

- What is SDFormat?
- How you can improve SDFormat
- Python bindings
- Converters for other robot description formats
- SDFormat in ROS 2
- New improvements in the SDFormat specification
Simulation Description Format (SDFormat) is an XML format for describing objects and environments for robot simulators. It describes objects and environments for robot simulators, visualization, and control. SDFormat is designed for scientific robot applications and is an extensible format describing all aspects of robots, static and dynamic objects, lighting, terrain, and physics.
Table of content

- What is SDFormat?
- How you can improve SDFormat
- Python bindings
- Converters for other robot description formats
- SDFormat in ROS 2
- New improvements in the SDFormat specification
How you can improve SDFormat

- Make a proposal
- Request changes
  - Direct conversations, GitHub issues, ROS Discourse, Gazebo Community, etc
- Write a formal proposal!
  - Open a pull request on http://bit.ly/github_sdf_tutorials
- Give feedback on new proposals
Table of content

- What is SDFormat?
- How you can improve SDFormat
- Python bindings
- Converters for other robot description formats
- SDFormat in ROS 2
- New improvements in the SDFormat specification
SDFormat python bindings

- pybind11
- Programmatically create/edit models
- Compatible with gz.math and gz.sim
import sdformat13 as sdf

root = sdf.Root()
try:
    root.load(input_file)
except sdf.SDFErrorsException as e:
    print(e, file=sys.stderr)

# Create a new element
world = root.world_by_index(0)
world.set_name('shapes')

...

with open('new_sdf.sdf', "w") as f:
    f.write(root.to_string())
Table of content

● What is SDFormat?
● How you can improve SDFormat
● Python bindings
● Converters for other robot description formats
● SDFormat in ROS 2
● New improvements in the SDFormat specification
● High-performance extensible software platform for animated 3D scenes
● Designed to meet the needs of large-scale film and visual effects production
● Expanding set of schemas, covering geometry, shading, lighting, and physics
SDFormat -> USD

$ sdf2usd warehouse.sdf warehouse.usd

Isaac Sim

Gazebo
USD -> SDFFormat

$ usd2sdf robot.usd robot.sdf
● XML model files
● MJCF models can represent complex dynamical systems with a wide range of features and model elements.
● Support joint, geom, site, camera and light.
  ○ Supported in the converter:
    ■ Bodies
    ■ Geoms
    ■ Sensors (Camera, Force torque, IMU)
    ■ Joints (Fixed, Free, Hinge, Slide)
    ■ Materials
SDFormat-MJCF: Installation

With `gz-garden` or higher version installed:

```
$ pip install sdformat-mjcf
```
SDFormat -> MJCF

$ sdf2mjcf input_file.sdf output_dir

Example: Pand Arm
MJCF -> SDFORMAT

$ mjcf2sdf input_file.xml output_dir

Example: Humanoid
Table of content

- What is SDFormat?
- How you can improve SDFormat
- Python bindings
- Converters for other robot description formats
- SDFormat in ROS 2
- New improvements in the SDFormat specification
SDFormat in ROS 2

- A urdf_parser_plugin for SDFormat
  - Parses SDFormat into URDF C++ data structures
  - SDFormat files can be loaded directly into robot_description

---

**ROS URDF**
Universal Robotic Description Format

---

![diagram showing ROS URDF components](image)
Table of content

- What is SDFormat?
- How you can improve SDFormat
- Python bindings
- Converters for other robot description formats
- SDFormat in ROS 2
- New improvements in the SDFormat specification
Improved Composition

● Using the `<include>` tag for modularity
● Models are self-contained/standalone components similar to parts in CAD assembly
● Frames, links, and joints in a model should be considered the public "API" of the model
  ○ Frames as primary interface elements (eg. use frames to define mounting points)
Model scope and the "::" syntax

- Only relative references are permitted
- References can access the current scope or child scopes

```xml
<model name="top_model">
  <link name="top_link"/>

  <frame name="top_to_bottom">
    <pose relative_to="mid_model::bottom_model::bottom_link"/> <!-- VALID -->
  </frame>

<model name="mid_model">
  <link name="mid_link">
    <pose relative_to="top_link"/> <!-- Error -->
  </link>

  <model name="bottom_model">
    <link name="bottom_link">
      <model>
      </model>
  </model>
</model>
```
Example: Frame Semantics with Nested References

<sdf version="1.8">
  <world name="default">
    <include>
      <uri>models/table</uri>
      <name>table1</name>
    </include>

    <include>
      <uri>models/Coke</uri>
      <pose relative_to="table1::top_surface"/>
    </include>
  </world>
</sdf>
Example: Placement Frame

```xml
dsdf version="1.8">
  <world name="default">
    <include>
      <uri>models/table</uri>
      <name>table1</name>
      <pose>0 0 0 0 -0.05 0</pose>
    </include>
    ...
    <include>
      <uri>models/big_sphere</uri>
      <placement_frame>pl_bottom</placement_frame>
      <pose relative_to="table1::top_surface">
        0.4 0 0 0 0 0
      </pose>
    </include>
  </world>
</sdf>

<placement_frame>pl_bottom</placement_frame>
```
Example: Robot arm assembly

```xml
<model name="robot_arm_with_gripper">
  <joint name="fix_to_world" type="fixed">
    <parent>world</parent>
    <child>arm</child>
  </joint>
  <include>
    <uri>models/panda_arm</uri>
    <name>arm</name>
  </include>
  <joint name="attach_gripper" type="fixed">
    <parent>arm::gripper_mount</parent>
    <child>gripper</child>
  </joint>
  <include>
    <uri>models/panda_hand</uri>
    <name>gripper</name>
    <placement_frame>mount_point</placement_frame>
    <pose relative_to="arm::gripper_mount"/>
  </include>
</model>
```
Example: Robot arm assembly

```
<model name="robot_arm_with_gripper">
  <joint name="fix_to_world" type="fixed">
    <parent>world</parent>
    <child>arm</child>
  </joint>
  <include>
    <uri>models/panda_arm</uri>
    <name>arm</name>
  </include>
  <joint name="attach_gripper" type="fixed">
    <parent>arm::gripper_mount</parent>
    <child>gripper</child>
  </joint>
  <include>
    <uri>models/simple_gripper</uri> <!--changed -->
    <name>gripper</name>
    <placement_frame>mount_point</placement_frame>
    <pose relative_to="arm::gripper_mount"/>
  </include>
</model>
```
Merging for Composition

- Include other models without introducing a new scope
New ways of specifying pose

• Option to specify the rotation representation.
• Currently

```xml
<pose>{xyz} {rpy_radians}</pose>
```

• New feature: Use Degrees

```xml
<pose degrees="true">{xyz} {rpy_degrees}</pose>
```

• New feature: Rotation format

```xml
<pose rotation_format="euler_rpy">{xyz} {rpy_radians}</pose>
<pose rotation_format="euler_rpy" degrees="true">{xyz} {rpy_degrees}</pose>
<pose rotation_format="quat_xyzw">{xyz} {quat_xyzw}</pose>
```

http://sdformat.org/tutorials?tut=better_pose_proposal
Fluid Added Mass

- The effective mass of a body increases when moving in a fluid

\[(M + \mu)\ddot{x} = \sum F(x, t)\]

- Newton’s second law:
  - where \(M\) is the body mass inertia matrix, \(\mu\) is the fluid added mass matrix

- New `<fluid_added_mass>` element added under `//link/inertial/

- Available in SDFFormat 1.10

Mimic Joint Actuation Constraint

- Gearbox joint type provides equivalent functionality but requires defining more joints and intermediate links
- Mimic constraint simplifies the definition

http://sdformat.org/tutorials?tut=mimic_proposal
Automatic Moments of Inertia Calculations

- Bad Inertia values cause bad simulation
- Default Mass Matrix
  - mass = 1.0 Kg
  - Diagonal Elements = (1, 1, 1)
- 2 major workflows for computing inertial properties
  - Using CAD software
  - Mesh Processing Software, such as MeshLab
Automatic Moments of Inertia Calculations

- SDFormat 1.11 introduced <inertial auto="true"/>

```xml
<link name="robot_link">
  <inertial auto="true"/>
  <collision name="capsule_collision">
    <density>2710</density>
    <geometry>
      <capsule>
        <radius>0.5</radius>
        <length>0.7</length>
      </capsule>
    </geometry>
  </collision>
  ...
</link>
```
Summary

- Offline converters (USD and MJCF)
- Python API
- ROS 2 support
- SDFormat new features
  - SDFormat 1.7: Frame semantics, Parameter passing (experimental)
  - SDFormat 1.8: Composition (nested models)
  - SDFormat 1.9: New ways of specifying pose (angles in degrees, quaternions)
  - SDFormat 1.10: Merge-includes, `<joint>` in world, Fluid added mass
  - SDFormat 1.11: Automatic computation of moments of inertia

Feedback:

- We invite everyone to try it and test it! We are happy to receive your feedback
ROS / ROS 2
with
Kubernetes and KubeEdge

Oct. 19th, 2023
ROSCon 2023 @ New Orleans, US
Agenda

• Who are we?
• Background
• Problems
• Goals / Requirements
• Kubernetes
• KubeEdge
• Sample Deployment
• What’s missing? Next-gen proposal
• Community
Who are we?

• Tomoya Fujita (Presenter)
  • Software Engineer, Sony R&D US Laboratory
  • ROS TSC (Technical Steering Committee)
  • KubeEdge SIG Robotics Chair
  • fujitatomoya@github, tomoyafujita@linkedin

• Co-Authors
  • Yin Ding (Engineering Manager, Google)
    o KubeEdge TSC, Co-Founder of KubeEdge Project
    o Leading the Kubernetes Hardening team
  • Kevin Wang (Lead of Cloud Native Open Source Team, Huawei)
    o KubeEdge TSC, Co-Founder of KubeEdge Project
    o CNCF Ambassador, TOC contributor
  • Fei Xu (Senior Engineer, Huawei)
    o KubeEdge TSC, Maintainer
Background

• Broad use cases.

• Distributed and Connected System.

• Collaborative and Orchestrated Application.

• Circulatory Functioning System and Development

• Specific Hardware Acceleration.

• Security. (Device, Data, Network)
What is the pain?

• Platform Dependencies.

• Proprietary hardware support.

• Application Modularity.

• System and Security Integration.

• Application Specific Network Bridge.

• Application Developer Friendly.
Goal / Requirements

• Flexible Application Deployment.

• Zero Trust Security Support.

• Application Agnostic Network Configuration.

• Extend Device Capability.

• System Global Observability.

• Platform Agnostic Device Abstraction.
Kubernetes (Service Mesh)

• Application Deployment and Orchestration.
• Device Capability and Label Control.
• Custom Resource Extension.
• Auto-Scaling and Healing.
• Roll Up/Down, Canary Test.
• Role Based Access Control.
• Device-Plugin / Container Device Interfaces.
• Container Network Interfaces.
• Traffic Management.
• Observability.
• Security Policy.
KubeEdge

is built upon Kubernetes and provides core infrastructure support for networking, application deployment and metadata synchronization between cloud and edge.

• Cloud-Edge Coordination
• Edge Computing
• Edge Autonomy
• Simplified Deployment
• Kubernetes-native Support
• Resource Efficient

https://kubeedge.io/docs/
Sample Deployment

Theory is good, but please see how it works in the flesh!

ROS Kubernetes Tutorials

Contribution(Issues/PRs) always welcome!
ROS Multi-Node Deployment
ROS 2 Localhost Only

LAN (Physical Network)

Layer 2 Emulation (WeaveNet)

ROS LOCALHOST ONLY=1

ROS LOCALHOST ONLY=1

ROS LOCALHOST ONLY=1

ROS LOCALHOST ONLY=1

Kubernetes API Server

Primary node

Pod

ros2 talker

ros2 listener

Pod

ros2 talker

ros2 listener

Pod

ros2 talker

ros2 listener

Pod

Kubelet (agent)

Worker node

ROS LOCALHOST ONLY=1

ROS LOCALHOST ONLY=1

ROS LOCALHOST ONLY=1

ROS LOCALHOST ONLY=1
ROS 2 Logical Partition / Multiple RMW Implementation

LAN (Physical Network)

Bind Host Network Interface

ROS_DOMAIN_ID=5
RMW_IMPLEMENTATION=rmw_fastdds_cpp

ROS_DOMAIN_ID=10
RMW_IMPLEMENTATION=rmw_cyclonedds_cpp

Kubernetes agnostic

Kubernetes API Server

Primary node

Worker node

Label: {nodetype: edgeserver}

Label: {nodetype: edgedevice}

ROS2 CLI

Docker

Pod

ros2 talker

ros2 listener

Kubelet (agent)
ROS 2 Deployment Intermediate

LAN (Physical Network)

Layer 2 Emulation (WeaveNet)

Primary node

Worker node

Kubernetes API Server

Label:
\{nodetype: edgeserver\}

Kubelet (agent)

Label:
\{nodetype: edgedevice\}

Pod

turtlesim

Pod

rgt

Pod

ros2 shell

Start turtlesim teleop app in this container
ROS 2 / Micro-ROS with KubeEdge (W.I.P)
Device-Abstraction (Device-Plugin, Container Device Interface)

- Kubernetes Custom Resource Definition, that allows us to plugin vendor specific hardware and device to the containers.
- After advertising the custom resource to Kubernetes, Kubernetes controls those resources with workload based on application requirements.
Support SROS 2 security enclaves via ConfigMap

Certificate to Join this entire distributed system,
Access permission for each topics and services

LAN

Administrator

Primary

App

::ROS

App Container

kubelet

User

Access Control

Worker

App

::ROS

App Container

kubelet

Worker

App

::ROS

App Container

kubelet

Registration

Load

ConfigMap & Secrets for Each ROS2 Application
What’s missing? Proposals?

• Device Abstraction Enhancement
  • KEP-3162: Add Deallocate and PostStopContainer to Device Manager API
  • Add CDI devices to device plugin API resolved this issue.
• KubeEdge CNI support (e.g. edgemesh, Cilium)
• More Edge Optimization / Configurable Options for Resource Constrained Device Support

• Cloud-Native Robotics Management Solution
  • `RoboDevOps` through Edge-Cloud Synergy
  • Cloud-Native Digital Twin for testing and data generation training
  • Robotics App Development Friendly
  • Cloud Robotics Custom Resource Definition and Operator Proposal
  • Edgemesh: adaptive cross-edge and edge-cloud data plane support
  • VSLAM algorithm with KubeEdge
  • Building a Robot-Oriented Intelligent Monitoring System
Community

- Kubernetes IoT Edge WG
- KubeEdge SIG Robotics
- ROS Kubernetes Tutorials