Properties

In CoppeliaSim, all internal variables or states can be handled as properties: those can be queried or modified via dedicated API functions.

Properties relate to a *target*, such as CoppeliaSim in general (sim.handle_app), a scene (sim.handle_scene), or a specific object (e.g. a scene object):

Python Lua

```
#python

sim.getStringProperty(sim.handle_app, 'settingsPath') # get location of user settings file usrset.txt
sim.setVector3Property(sim.handle_scene, 'gravity', [0.0, 0.0, 0.0]) # set zero-gravity
sim.getFloatArrayProperty(jointHandle, 'interval') # get joint lower and upper limits
sim.getIntArrayProperty(shapeHandle, 'meshes') # get handles of meshes composing the shape
sim.getFloatArrayProperty(meshHandle, 'vertices') # get vertices of a mesh
```

This page lists all available targets and properties. A property usually is readable and writable, but depending on the property, it can also be read-only or write-only. Other properties can be removable:

Python Lua

```
#python

sim.getIntArrayProperty(shapeHandle, 'meshes') # read-only property
sim.setBoolProperty(shapeHandle, 'applyCulling', true) # write-only property
sim.removeProperty(objectHandle, 'customData.myFloatArray') # removable property
```

Properties can conveniently be inspected via the Property Explorer add-on located at [Modules > Developer tools > Property explorer].

Most properties, when changed (explicitly or internally by CoppeliaSim), trigger an event, which can be listened to via an event callback function.

There are particular types of properties:

- custom data
- signals
- named parameters
- engine-related properties

Custom data

Custom data properties can be seen as persistent data (stored to disk) with different scopes: globally (i.e., sim.handle_app), within the current scene (sim.handle_scene), or locally, specific to a scene object (where the *target* is the scene object's handle). Custom data properties are identified with a "customData."-prefix, and they can be created/set, modified and removed. For example:

Python Lua

```
#python

# script 1 writes the data to a scene object:

myData = [1, 2, ["Hello", "world", True, {"value1": 63, "value2": "aString"}]]
sim.setBufferProperty(objectHandle, "customData.myTag", sim.packTable(myData))
```

Python Lua

```
#python

# script 2 reads the data from the scene object:

myData = sim.getBufferProperty(objectHandle, "customData.myTag", {'noError' : True})
if myData:
    myData = sim.unpackTable(myData)
    # do something with the data
```

Unlike signals, custom data is not volatile nor removed if the script state that created/modified it is destroyed.

Signals

Signal properties can be seen as variables with different scopes: globally (i.e., sim.handle_app), within the current scene (sim.handle_scene), or locally, specific to a scene object (where the *target* is the scene object's handle). Signal properties are identified with a "signal."-prefix, and they can be created/set, modified and removed. For example:

Python Lua

```
#python

# script 1 writes the data to int signal 'myIntSignal':

sim.setIntProperty(sim.handle_scene, "signal.myIntSignal", 37)
```

Python Lua

```
#python

# script 2 reads the data from int signal 'myIntSignal':

myData = sim.getIntProperty(sim.handle_scene, "signal.myIntSignal", {'noError' : True})
if myData != None:
    # do something with the data
```

If the script state that created/modified a signal is destroyed, then that signal is automatically removed.

Named parameters

Named parameters are volatile string properties related to CoppeliaSim in general (i.e. target is sim.handle_app). Named parameters are usually user settings, or user arguments set via the -G command line switch. For instance:

Python Lua

```
#python

arg = sim.getStringProperty(sim.handle_app, "namedParam.myCustomArgument")
```

Engine-related properties

Engine-related properties are properties relative to physics engines. Those properties can easily be identified from their "bullet.", "ode.", "newton.", "vortex.", or "mujoco." prefix.