Accessing scene objects programmatically

When programming in and around CoppeliaSim, you will always need to reference scene objects. You do this with handles, that you obtain via sim.getObject, which expects an object path as input argument. The object path can be expressed in an absolute manner, but also in a relative manner in case of script object code.

In both cases, the path to the object can often be simplified. You may also use wildcards, and optionally specify the object sequence or order, in a given scene hierarchy. Or simply fetch the n-th object that matches a specific path/alias.

Access from *unassociated* code

Unassociated code is code that is not from a script object. This includes all the code written for plugins, add-ons, remote API clients, external ROS nodes, and the main script.

In that case, you simply specify the object's **absolute** path, in order to retrieve its handle. If the object alias is unique, the path to the object can be simplified. You may also use wildcards, and optionally specify the object sequence in a given scene hierarchy. Or simply fetch the n-th object that matches a specific path:

```
// e.g. from within a c/c++ plugin:

// using the full Object path:
int objectHandle = simGetObject("/Path/to/Object", -1, -1, 0);

// if object has unique alias Object:
int objectHandle = simGetObject("/Object", -1, -1, 0);

// handle of the first object with alias Robot, in a given hierarchy level:
int robotHandle = simGetObject("/Robot[0]", -1, -1, 0);

// handle of the second object with alias Robot, in a given hierarchy level:
int robotHandle = simGetObject("/Robot[1]", -1, -1, 0);

// handle of the fifth object with alias Robot, in the scene:
int robotHandle = simGetObject("/Robot{4}", -1, -1, 0);
// or
int robotHandle = simGetObject("/Robot", 4, -1, 0);

// find all objects starting with alias prefix Mobile:
int i = 0;
while (true)
{
    int objectHandle = simGetObject("/Mobile*", i++, -1, 1);
    if (objectHandle < 0)
        break;
}
```

```
# e.g. from within a Python ZeroMQ remote API client:

# using the full Object path:
objectHandle = sim.getObject("/Path/to/Object")

# if object has unique alias Object:
objectHandle = sim.getObject("/Object")

# handle of the first object with alias Robot, in a given hierarchy level:
robotHandle = sim.getObject("/Robot[0]")

# handle of the second object with alias Robot, in a given hierarchy level:
robotHandle = sim.getObject("/Robot[1]")

# handle of the fifth object with alias Robot, in the scene:
robotHandle = sim.getObject("/Robot{4}")
```

Access from a script object (i.e. from *associated* code)

Associated code is code that runs in a script object, i.e. code that runs in a simulation- or customization script.

In that case, objects can also be accessed in an **absolute** manner as described above, but additionally, object access can also happen in a **relative** manner (relative to the current script object (*./*), or relative to the model containing current script (*:/*)):

Python | Lua

```
#python

# from within a simulation- or customization script:

# returns the current script's handle:
objectHandle = sim.getObject(".")

# returns the parent of this script:
parentHandle = sim.getObject("..")

# returns the model this script is contained in:
modelHandle = sim.getObject(":")

# returns the parent model this script is contained in:
parentModelHandle = sim.getObject("::")

# returns the first object in the current hierarchy, that starts with Object:
objectHandle = sim.getObject("./Object*")

# returns the first object with alias Object, in the current model hierarchy:
objectHandle = sim.getObject(":/Object")

# returns the 4th object with alias Leg, within a same hierarchy level, in the current model hierarchy:
legHandle = sim.getObject(":/Leg[3]")

# returns the 5th object with alias Leg, in the current model hierarchy:
legHandle = sim.getObject(":/Leg{4}")

# parse through all Leg objects in the current model hierarchy:
i = 0
while True:
```

```
    legHandle = sim.getObject(":/Leg", {'index': i, 'noError': True})
    if legHandle < 0:
        break
    i = i + 1

# returns the first object with alias Object, in the hierarchy of anotherObject:
local objectHandle = sim.getObject("./Object", {'proxy': anotherObject})
```