User manual search (offline, online), forum search

ZeroMQ remote API

The ZeroMQ remote API is one of several ways an application can connect with CoppeliaSim.

The ZeroMQ remote API allows to control a simulation (or the simulator itself) from an external application or a remote hardware (e.g. real robot, remote computer, etc.). It offers all API functions also available via a CoppeliaSim script: this includes all regular API functions (i.e. sim.* -type functions), but also all API functions provided by plugins (e.g. simOMPL.*, simUI.*, simIK.*, etc.), if enabled.

The ZeroMQ remote API functions are interacting with CoppeliaSim via ZeroMQ and its interface plugin to CoppeliaSim and the ZMQ remote API add-on. All this happens in a hidden fashion to the user. The remote API can let one or several external applications interact with CoppeliaSim in a stepping (i.e. *synchronized* with each simulation step) or non-stepping way (i.e. the normal operation mode), and even remote control of the simulator is supported (e.g. remotely loading a scene, starting, pausing or stopping a simulation for instance).

See programming/zmqRemoteApi folder or its related repository for examples.

Python client

Install client package:

```
$ python3 -m pip install coppeliasim-zmqremoteapi-client
```

(the ZeroMQ and CBOR dependencies will be automatically installed with the above command).

Following is a very simple example ZeroMQ remote API client code, which starts then runs a stepping simulation for 3 seconds:

```
from coppeliasim_zmqremoteapi_client import RemoteAPIClient

client = RemoteAPIClient()
sim = client.require('sim')

sim.setStepping(True)

sim.startSimulation()
while (t := sim.getSimulationTime()) < 3:
    print(f'Simulation time: {t:.2f} [s]')
    sim.step()
sim.stopSimulation()
```

C++ client

Any C++ client requires the jsoncons and cppzmq package: those are automatically downloaded and used when compiling via cmake. The boost and ZMQ libraries need to be installed. For details see *programming/zmqRemoteApi/clients/cpp/*, which contains several examples.

Build them with:

```
$ mkdir build
$ cd build
$ cmake ..
$ cmake --build . --config Release
```

Following is a very simple C++ ZeroMQ remote API client code, which starts then runs a stepping simulation for 3 seconds:

```
#include "RemoteAPIClient.h"

int main(int argc,char* argv[])
{
    RemoteAPIClient client;
    auto sim = client.getObject().sim();

    sim.setStepping(true);

    sim.startSimulation();
    double t = 0.0;
    do
    {
        t = sim.getSimulationTime();
        printf("Simulation time: %.2f [s]\n", t);
        sim.step();
    } while (t < 3.0);
    sim.stopSimulation();
    return(0);
}
```

Java client

Any Java client requires Apache Maven. For details see *programming/zmqRemoteApi/clients/java/src/main/java*, which contains several examples.

Make sure that your folder names do not contain any spaces, and have CoppeliaSim running (API function are fetched from CoppeliaSim). Build with:

```
$ export COPPELIASIM_ROOT_DIR=path/to/CoppeliaSim/exec/folder/or/resources/folder/on/macOS
$ cd zmqRemoteApi/clients/java
$ mvn package -D"GENERATE_INCLUDE_OBJECTS=sim,simIK"
```

In above, only the sim and simIK namespaces are considered, you may add more when needed. Following is a very simple Java ZeroMQ remote API client code, which starts then runs a stepping simulation for 3 seconds:

```
import java.util.Arrays;

import com.coppeliarobotics.remoteapi.zmq.*;

public class Example
{
    public static void main(String[] _args) throws java.io.IOException, co.nstant.in.cbor.CborException
    {
        var client = new RemoteAPIClient();
        var sim = client.getObject().sim();

        sim.setStepping(true);
        sim.startSimulation();

        double simTime = 0.0;
        while((simTime = sim.getSimulationTime()) < 3)
        {
            System.out.printf("Simulation time: %.2f [s]%n", simTime);
            sim.step();
        }
        sim.stopSimulation();
    }
}
```

Matlab & Octave clients

Matlab clients require the bundled JeroMQ, which installs automatically if not yet present.

Octave clients require Octave 6.4+, the octave communications and zeromq packages. Those can be installed with:

```
pkg install -forge communications
pkg install -forge zeromq
```

Following is a very simple Matlab/Octave ZeroMQ remote API client code, which starts then runs a stepping simulation for 3 seconds:

```
client = RemoteAPIClient();
sim = client.require('sim');

sim.setStepping(true);

sim.startSimulation();
while true
    t = sim.getSimulationTime();
    if t >= 3; break; end
    fprintf('Simulation time: %.2f [s]\n', t);
    sim.step();
end
sim.stopSimulation();
```

Lua client

Currently, a Lua client is only supported from within a CoppeliaSim script, e.g. in order to connect 2 or more CoppeliaSim instances.

Following is a very simple Lua ZeroMQ remote API client code, which synchronizes the simulation steps with another CoppeliaSim instance:

```
function sysCall_init()
    remoteApiClient = require('luaZmqRemoteApi')
    remoteApiClient.init('127.0.0.1', 23002)
    simx = remoteApiClient.getObject('sim')

    remoteApiClient.setStepping(true)
    simx.startSimulation()
end

function sysCall_sensing()
    remoteApiClient.step()
end

function sysCall_cleanup()
    simx.stopSimulation()
    remoteApiClient.cleanup()
end
```

Rust client

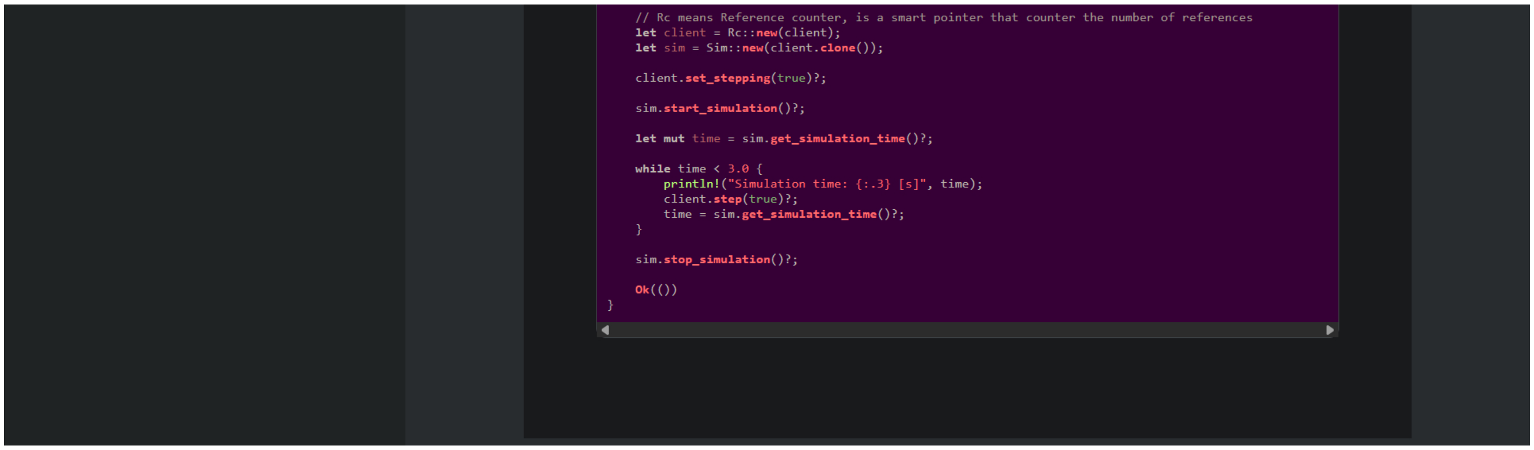
The Rust ZeroMQ remote API is courtesy of Samuel Cavalcanti. Following is a very simple example ZeroMQ remote API client code, which starts then runs a stepping simulation for 3 seconds:

```
use std::rc::Rc;
use zmq_remote_api::{sim::Sim, RemoteAPIError, RemoteApiClientParams};

/*
    Example based on Example.cpp
*/

fn main() -> Result<(), RemoteAPIError> {
    // use the env variable RUST_LOG="trace" or RUST_LOG="debug" to observe the zmq communication
    env_logger::init();

    let client = zmq_remote_api::RemoteApiClient::new(RemoteApiClientParams {
        host: "localhost".to_string(),
        ..RemoteApiClientParams::default()
    })?;
```



```rust
    // Rc means Reference counter, is a smart pointer that counter the number of references
    let client = Rc::new(client);
    let sim = Sim::new(client.clone());

    client.set_stepping(true)?;

    sim.start_simulation()?;

    let mut time = sim.get_simulation_time()?;

    while time < 3.0 {
        println!("Simulation time: {:.3} [s]", time);
        client.step(true)?;
        time = sim.get_simulation_time()?;
    }

    sim.stop_simulation()?;

    Ok(())
}
```