Instrument Server Documentation

Release 0.1

Aquiles Carattino

Sep 01, 2017

Contents:

The instrument server is built on top of Flask while the client relies on the requests module. The client is not necessary, but outlines the basic communication with the server.

The server uses as inputs the devices already instantiated, meaning that one passes as arguments variables that already hold the communication with the device. This enables to pass several devices that rely on the same class. The devices are registered in an internal dictionary of the server, avoiding the possibility of triggering actions more than once.

The measurements are done in separated threads to avoid timeouts in the server and to allow the user to trigger several actions simultaneously. The thread relies on python threading capability and therefore should not be used with computer intensive operations, but with device intensive operations.

CHAPTER 1

How to use it

You need to import the instserver package into your own project and load the devices you want to have available.

CHAPTER 2

Server side

In the server side you run just the server and the classes that correspond to the devices. If you need more personalization, you can always check the documentation at Flask, specially regardin the listening IP and port:

```python
from instserver.server import InstServer
from instserver.dummyDevice import dummyDevice
# First instantiate the device
dev = dummyDevice()

# Now is time for the server:
server = InstServer(__name__)
server.add_device(dev,'dev')
server.run(debug=True)
```

CHAPTER 3

Client side

The code is:

```
from instserver.client import InstClient
# First instantiate the client with the IP address and port of the server
c = InstClient('http://127.0.0.1:5000')
# Let's print a list of the available devices and methods on the server
print(c.listdevices())
```

instserver

instserver package

Submodules

instserver.client module

instserver.dummyDevice module

dummyDevice.py

Dummy device for testing the server and client. The dummy device defines three methods, initialize with returns True, idn that sleeps for two seconds and returns a string and measure that takes one argument and returns a numpy array after sleeping.

Section author: Aquiles Carattino <aquiles@aquicarattino.com>

class `instserver.dummyDevice.`**`DummyDevice`**

Bases: `object`

Dummy Device class

idn()
Identifies the dummy device via a serial number :return: Serial number

initialize()
Initializes the dummy device :return: True

measure(*time*)
Simulates a measurement with number of points defined by input and sleeping the same number of milliseconds.

Parameters **time** (*int*) –

Returns random numpy array

instserver.instThread module

instserver.messages module

Dictionary to keep track of the messages passed from the server to the client. This is useful to catch errors and handle them.

The current version is only a sketch that has to be further developed.

Section author: Aquiles Carattino <aquiles@aquicarattino.com>

instserver.server module

server.py

The server is the core of the package and therefore has to be the first place to look for errors or improvements. The server is built on top of Flask, a lightweight framework that allows to rapidly build and deploy solutions. The server will open a connection on a specified port on a specified listening IP address. Care has to be taken to have the port open on the computer, check the firewall rules if errors happen.

The code to run the server should be similar to:

```
from instserver.server import InstServer
from instserver.dummyDevice import dummyDevice
# First instantiate the device
dev = dummyDevice()

# Now is time for the server:
server = InstServer(__name__)
server.add_device(dev,'dev')
server.run(debug=True)
```

Section author: Aquiles Carattino <aquiles@aquicarattino.com>

class instserver.server.**InstServer**(*import_name*)
Bases: flask.app.Flask

Server class to run tasks over the network. It allows to load devices instantiated and to trigger their methods. The task after triggering happens in a separate thread, meaning the several tasks can be executed concurrently.

add_device(*device*, *name*)
Registers a new device into the server class

Parameters

- **`device`** – device previously initiated.
- **`name`** – name of the device to be used for identification.

Returns dictionary of devices

`deviceThread`(*device*, *name*, *method*, *args=None*)

Function to be run in a separate thread to trigger a mesaurement or an action on a device. It relies on the threading capabilities of Python, therefore is not running in a different core but on a different interpreter. Heavy load activities on the computer side will affect performance. .. todo:: The way data is exchanged to the main process is highly non-recommended, since it overwrites a dictionary belonging to the main thread. It can be improved by using signaling.

Parameters

- **`device`** – Device in which to trigger the action
- **`name`** – name given to the device when registering it
- **`method`** – method to trigger in the device
- **`args`** – arguments to pass to the method. They can be none

Returns updates an internal variable to the class

`get_data`()

Gets the data from a specific method on a device if it is available. After this, the data is not destroyed.

`list_devices`()

Lists the devices registered

Returns json dump of a dictionary

`main_device`(*name*)

Returns the methods of the device. It is mostly for debugging options, since it outputs text readable from a web explorer.

Parameters **`name`** – name of the device

Returns methods

`mainpage`()

Results in a message if one checks that the server is running.

Returns

`threadFinish`(*name*, *method*, *r*)

Method to identify when a device is done with an acquisition. This is called internally by the working thread and should not be accessed from the outside of the class.

Parameters

- **`name`** – Name of device running
- **`method`** – Method in the device running
- **`r`** – output of the method

Returns

`trigger_device`()

Triggers a specific method of a device with arguments. The device is identified by its name when registered into the server. Makes a connection of the signal of the thread to a function that registers the completion of the task.

Returns json dump of a message

instserver.testServer module

Module contents

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index