Using the Python client

An explanation about how to install and use the Python client can be found here: Python

However, in this document we will create an example tutorial to learn how to work with the Python client in order to get the most out of it.

script1.py

```
# First, we need to import both the ClientConfiguration and the OpenCGAClient
from pyopencga.opencga_config import ClientConfiguration
from pyopencga.opencga_client import OpenCGAClient

# The main client-configuration.yml file has a 'host' section to point to the Rest OpenCGA endpoints.
# We need to either pass the path to the configuration file or a dictionary with the format of the file.
config = ClientConfiguration('/opt/opencga/conf/client-configuration.yml')
config = ClientConfiguration({
        "rest": {
                "host": "http://bioinfo.hpc.cam.ac.uk/opencga-demo"
        }
})

# And finally create an instance of the OpenCGAClient passing the configuration
oc = OpenCGAClient(config)

# Now we need to authenticate.
oc.login('myUser')               # If done this way, password will be prompted to the user so it is not
displayed but...
oc.login('myUser', 'myPassword') # ... it is also possible to pass the password directly as an additional
parameter

# Let's assume our installation already has been populated and we are interested in looking for
# all the families containing a concrete disorder: 'Rod-cone dystrophy'. To fetch this data, we will need to:
family_query_response = oc.families.search(study="study1", limit=10, disorders="Rod-cone dystrophy", include="
id,members.id")

# Running oc.families.search(disorders="Rod-cone dystrophy") with only the 'disorders' field would only work
# if only one project and one study has been defined. However, we expect that most of the OpenCGA installations
# will have more than one study, so we need to specify the families of which study we are looking for.

# Additionally, we are passing limit = 10 to limit the number of family results we want to fetch. Because this
# is an example, we are simply limiting the number of results to 10.

# Finally, if we don't specify anything else, all the values from the Family will be fetched. When writing
# scripts, we are normally interested in just a few fields of a whole entry, so adding the include/exclude
fields
# will definitely help us getting the results faster as we will avoid sending data we are going to discard
through
# the network. In this particular case, we are only interested in getting the Family id and the id of the
members
# of the family. To know what fields you can include/exclude, please follow the data models we have defined.

# family_query_response is an instance of the QueryResponse class defined in the Python library. To read the
fields,
# we could do the following:
family_query_response.time         # Get the time spent with the REST call
family_query_response.apiVersion   # Get the API version of the REST
family_query_response.queryOptions # Get the QueryOptions of the call (include/exclude, limit, skip, count...)
family_query_response.warning      # Get warning messages
family_query_response.error        # Get error messages
family_query_response.responses    # Get the responses (Array of QueryResults containing the data queried)

# We can iterate over all the results to print all the id's using the 'results()' method such as in the example
below:
for family in family_query_response.results():
        print (family['id'])

# We could have this same behaviour if we run the following script, which is why 'results()' is that useful.
for query_result in family_query_response.responses:
        for family in query_result['results']:
                print (family['id'])

# If we want to know exactly the amount of results obtained, we can run:
family_query_response.num_results()

# Or let's say that instead of querying the data, we only wanted to get the number of families in the study
with that disorder. In that case, we could:
family_count_response = oc.families.search(study="study1", disorders="Rod-cone dystrophy", count=True)

# And then get the number of matches by calling to the num_matches method:
family_count_response.num_matches()

# Now that we know how to work with the OpenCGA QueryResponse object, we will write a script to fetch all the
variants
# falling in the 'BMPR2' gene found in any member of the family. In this case, we will limit the variant query
to a maximum
# of 10 results excluding the sample information (sample information can be huge and would make this query much
slower).
for family in oc.families.search(study="study1",limit=10,disorders="Rod-cone dystrophy",include="id").results():
    print ("Family: " + family['id'])
    variant_response = oc.variant.query(family=family['id'], gene= 'BMPR2', study='study1',
includeSamples=None, limit=10)
    if variant_response.num_total_results() > 0:
        for variant in variant_response.results():
            print (variant['chromosome'] + ":" + str(variant['start']) + "-" + str(variant['end']) + '\t' +
variant['type'])
    else:
        print ("No variant results found")
    print()
```