How to call Axel from a python Application?

Thanks to a JSON RPC protocol, using qjsonrpc library, Axel can be called from a python Application.
For further information about what's a JSON RPC protocol you can take a look here.

This document explains how you can use this functionality.

1) **How I can get this functionality?**

   This protocol was added in the version 2.3.2 of Axel.
   If you get the sources you must have installed qjsonrpc and put the cmake variable JSON_RPC to ON. That's way Axel embeds a server which listens to all messages send by axlClient data.

2) **Requirements**

   To communicate with Axel, the python application must use a client to send messages to the Axel server. The client must be of type axlClient defined in Axel kernel application.
   To use those kind of data you should import some part of Axel such as described below:

   ```python
   import sys
   sys.path.append("your_path_to_axel_modules")
   # To get some axel data
   import axlcore
   from axlcore import *
   # To get axlClient methods
   import axlrpc
   from axlrpc import *
   ```

   You'll find Axel modules folder in the build of Axel if you are using sources. Otherwise it must be installed on your computer system, for instance C:\Program Files (x86)\axel-2.3.2\modules on Windows.
   You should also initialize factories for getting Axel plugins data and processes.

   ```python
   #Initialize factories for plugins
   dtkPluginManager = axlcore.dtkPluginManager.instance()
   dtkPluginManager.initializeApplication()
   dtkPluginManager.initialize()
   dtkDataFactory = axlcore.dtkAbstractDataFactory.instance()
   dtkProcessFactory = axlcore.dtkAbstractProcessFactory.instance()
   ```
A more comprehensive example is given in section 4 of this document.

3) **What kind of communication is possible?**

The communication is bi-directionnal as you can get data created in Axel with your python application and sending python application data to Axel view. Your python application can also ask Axel to compute an axel algorithm/process, with specific inputs and parameters, and getting the result(s).

**A) Sending data to the Axel View**

- `sendData(axlAbstractData *data)`, send the object in the Axel view.
- `modifyData(axlAbstractData *data)`, modify the properties of the object in Axel view.
- `deleteData(axlAbstractData *data)`, delete the object representation in Axel view.

**B) Getting data from Axel**

- `update(axlAbstractData *data)`, update the properties of the corresponding object in the python application. If the object was deleted in Axel view delete it.
- `getData(QString name)`, an object was created in Axel, create the corresponding object in python application.

**C) Asking Axel to compute an algorithm**

- `callProcess("processName",listInput, listParam)`, Axel computes the algorithm with the inputs list and parameters given. Returns the output. ListInput and listParameter are of type axlAbstractDataComposite.

4) **A python script example**

```python
#!/usr/bin/python
import sys
sys.path.append("your_path_to_axel_modules")

# To get some axel data
import axlcore
from axlcore import *
# To get axlClient methods
import axlrpc
from axlrpc import *
import subprocess

#Initialize factories for plugins
dtkPluginManager = axlcore.dtkPluginManager.instance()
```
dtkPluginManager.initializeApplication()

dtkPluginManager.initialize()

dtkDataFactory = axlcore.dtkAbstractDataFactory.instance()

dtkProcessFactory = axlcore.dtkAbstractProcessFactory.instance()

# To run Axel. Be careful sometimes Axel open too late. You should run Axel apart if so.
# or wait more than 5 seconds by changings sleep 5 with another value.
pid = subprocess.Popen(['your_path_to_axel_application' , '--verbose'],shell=True)
p = subprocess.Popen('sleep 5', shell=True)# to wait axel opening.
p.wait()# to wait axel opening.

# Create a client to communicate with Axel.
c = axlClient()
# Create an axel point
point = axlPoint(2,6,5)
print point.description()
# Send it to the Axel view.
c.sendData(point)

#Modify the coordinates
point.setValues(1.8,2.5,0.0)
print point.description()
# Communicate the new values to Axel.
c.modifyData(point)

# If a new object was created in Axel, with name axlCone, the python application can get it.
#c.getData("axlCone") #suppose un axlCone de nom axlCore est cree a partir d'Axel.
#Delete the point representation in Axel view.
#c.deleteData(point)

# Here is an example to explain how to call a process in Axel and get the result. The process
called
# must be initialized and exist in Axel not in python application.
point1 = axlPoint(1.5,0.5,2)
point2= axlPoint(2,0,2)
line = axlLine(point1,point2)
listInput = axlAbstractDataComposite()
listInput.add(line)
valueParam = axlDouble(0.25)
listParam = axlAbstractDataComposite();
listParam.add(valueParam);
# Method to call process with specific inputs and parameters.
data = c.callProcess("axlBarycenterProcess",listInput, listParam )
print data.description()

p2 = subprocess.Popen('sleep 5', shell=True)
p2.wait()