COMMUNICATION PROTOCOL AT P04, PETRA III

(V1: abgespeckte quick-and-dirty Version für PIPE & Berliner)
(Update V2: einige neue Commands/Parser, Dummy Server, free Client, Communication-Logfile, remove OTF-script, Keithley reading, introduce status und Commandolistung)
(Update V3: viel Text gelöscht, nur noch Fokus auf die Commandoliste, weil User das bevorzugen
Update V4: einige send/check-Befehle, Screen, Mesh, Block, I0, Undulator-factor, alle Servervariablen im Memory in EINEM Array allokiert)
Update V5: PS2, exitslit right/left, pressure, undulator energy, status2

Gregor Hartmann, 2016-07-12

Readme for ComServer_V37.pyc

Schematics of the communication

The aim of this protocol is to have ONE protocol for all users which is capable of using the high performance of P04 at its best while adding new features is simple without compromising the established communication. The new features are added by adding parser arguments or new alias possibilities in the following scheme.
If you are not so interested in the ideas and motivation, you can directly continue with the section '**Command list as a series of examples**' which simply shows all your options.

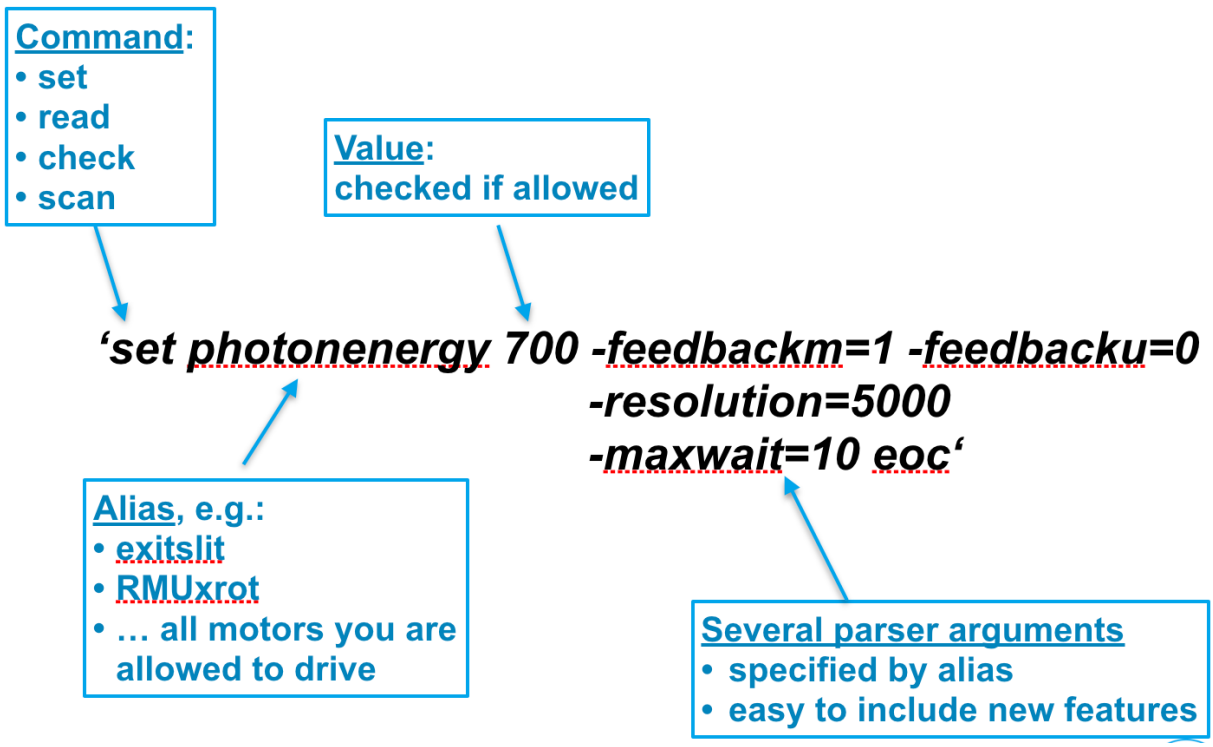


Figure from User's Meeting 2016

Endings of the communication strings

The string you send via the TCP(/IP) socket from your user-PC to our beamline-PC must always end with the following 4 chars (‘eoc' means End Of Command)
‘ eoc’ (space+eoc),
for instance:
‘set mono 700 eoc’
Any answer from the communication server will end with the following 4 chars (‘eoa' means End Of Answer):
‘ eoa’ (space+eoa),
for instance:
‘done eoa’
This protocol allows to send strings with variable length without running into any problems in the communication since it is always clear when one party has finished its communication. You have to strictly follow this protocol.

Establishing a connection

When the server is running at aimIP, port 3001 the connection is very easily established as for example in python with the following class which acts as a client:

```
# -*- coding: utf-8 -*-

import socket
import time
import sys

class server:
    def __init__(self,aimIP):
        self.aimIP=aimIP
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.server_address = (self.aimIP, 3001)
        print 'connecting to '+ str(self.server_address)
        self.sock.connect(self.server_address)
    def __call__(self,string):
        self.string=string
        self.sock.sendall(self.string)
        self.char=self.sock.recv(3)
        while True:
            self.curchar = self.sock.recv(1);
            self.char+=self.curchar
            if self.char[-4:]==' eoa':
                break
        print 'reply:'+self.char
        if 'bye!' in self.char:
            self.sock.close()

aimIP='131.169.225.245'
serv=server(aimIP)
```

This class establishes the connection when it is initialized. Calling it with a string leads to sending this command to the beamline communication server. Then it is waiting for the answer (ending with ‘ eoa’) which is self.char. If an error occurs or the connection is ended, the server will send you a ‘bye! eoa’ which leads to a closing process of the connection in the client.
If you want to close the connection, just send the command:
'closeconnection eoc’

Testing at home

In order to give you a test environment to check your communication before the beamtime, there is a Dummy-Server which you can run without being in the DESY-Network. It is a .py file and only needs python 2.7.xx and the following packages to be executed:

- numpy
- socket
- sys
- random
- time
- os
- subprocess

Please note and understand that the ‘real’ server at the beamline will always be a .pyc-file (compiled python) because of security reasons. This Dummy-server has only one device (the monochromator in dummy-device mode) meaning you can only set and read this device which is absolutely enough to test your client. In case you run into problems, you can simply start with the client of the previous section which is also available as a .py file.

Example:
open two terminal T1 and T2:

T1:

```
Gregors-MacBook-Pro:Dummy gregorhartmann$ python ComServer_V20_dummy.py
server on:192.168.0.101
waiting for connection
```

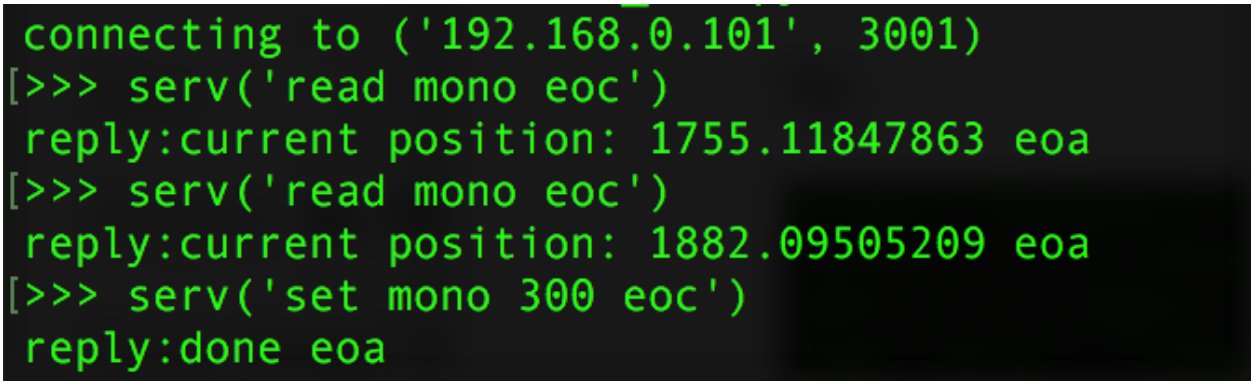
T2:

```
Gregors-MacBook-Pro:Dummy gregorhartmann$ python
Python 2.7.11 |Anaconda 2.3.0 (x86_64)| (default, Dec  6 2015, 18:57:58)
[GCC 4.2.1 (Apple Inc. build 5577)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
Anaconda is brought to you by Continuum Analytics.
Please check out: http://continuum.io/thanks and https://anaconda.org
>>> execfile('NewClient_V20.py')
connecting to ('192.168.0.101', 3001)
>>>
```

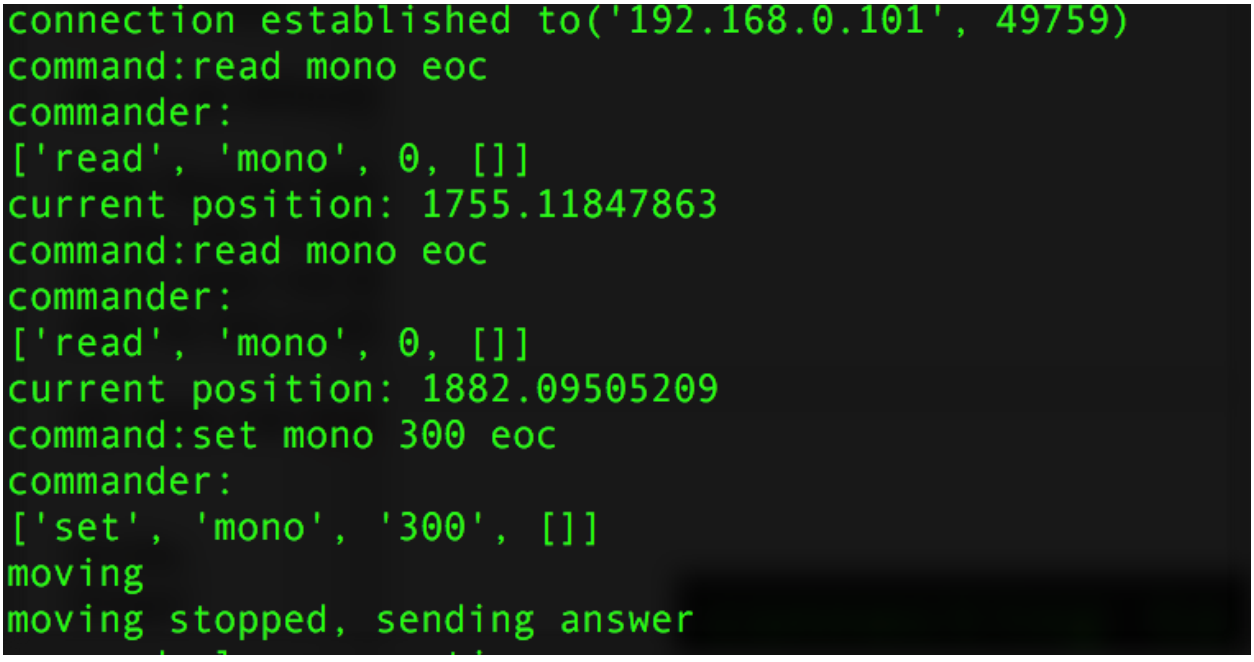
T1 will show:

```
waiting for connection
connection established to('192.168.0.101', 49759)
```

continue sending command in T2:



while T1 is printing:



Logging the communication

In order to find errors, all communications are logged with a timestamp.

Restarting the communication server
In case of a crash, the communication server will be restarted automatically.

Running a Makro environment at P04
If you prefer this syntax over the beamline panel, you can also run it in a (I)python console. Therefore, start a console via 'python'. Now, execute the following command:
execfile('/PETRAIII/p04/beamline/PythonLib/NewCom/NewClient_V24.py')
(directory will be changed to common/…)
After this you can run the commands with in a class called 'serv()' (see above). You can use this for single commands and for loops as well, for instance:

```
for k in np.linspace(250,1200,3000):
    serv('set mono '+str(k)+' -resolution=50000 eoc')
```

Remark on parsers
If you accidentally use a parser which is not assigned to the used alias it will be simply totally ignored.

Out-of-range/not-allowed commands
If you send a device to a position which is out of range, you will get the answer:
'out-of-range eoa'
and the device is not moved at all. If you are trying to set a device which you can only read as for instance the ring current or the undulator shift, you will get:
'not-allowed eoa'

COMMAND LIST AS A SERIES OF EXAMPLES

- 'set photonenergy 300 eoc'
  sets the photon energy to 300 eV by moving the undulator and the monochromator to the desired position and waiting until BOTH devices are not moving anymore. Returns:
  'done eoa'

- 'set photonenergy 300 -resolution=5000 -maxwait=10 eoc'
  same command as before, but you wait until the monochromator reaches a resolution of 5000 or until 10s are passed. Returns:
  'done eoa'

- 'set photonenergy 300 -feedbacku=0 eoc'
  sends the monochromator to the desired position and checks if the undulator is moving. If so, it is not touching the undulator AT ALL. If not it is sending the undulator to the desired position as well. Anyway, it is ONLY waiting until the monochromator is at the correct position. This is much faster and very useful for small energy steps. Roughly speaking: If you perform an energy scan, your energy steps are smaller than 0.4% of the photon energy and you don't

care of having only 90% flux for ~0.5-1.5 s of your measurement time than you should use this mode since it is much faster. For a scan, it is CRUCIALLY essential that you wait for the undulator at the first energy position for obvious reasons. Returns:
'done eoa'

- 'send photonenergy 300 eoc'
  checks if the monochromator and undulator are moveable and sends them to 300eV. Returns 'started eoa' or 'error eoa'
  depending on if the devices can be moved of not. It is not waiting for anything so you can 'immediately' (after 'started eoa' or 'error eoa') send new commands. You should combine the 'send'-command with the following 'check'-command or use the 'set'-command which will handle this for you.

- 'check photonenergy eoc'
  returns '1 eoa' if the devices (mono and undu) are ready to move or '0 eoa' if not. The 'send'-command will return 'error eoa' if the status is '0 eoa'.

- 'send photonenergy 300 -feedbacku=0 eoc'
  same as before but without undulator feedback as in 'set photonenergy 300 -feedbacku=0 eoc'

- 'check photonenergy -feedbacku=0 eoc'
  same as before but without undulator feedback. This command exists for consistency, but it is the same command as 'check mono eoc'

- 'check photonenergy -resolution=5000 eoc'
  checks if the desired resolution is achieved within 1s. returns '1 eoa' or '0 eoa'

- 'send exitslit 300 eoc' and 'check exitslit eoc' are handled as the previous example.

- 'send screen X eoc'
  moves a holder at the beamline exit with:
  - X=0 : closes the beamline
  - X=1 : a mesh is moved into the beamline which you can use as I0 (to get this value read the keithley2)
  - X=2 : opens the beamline (without mesh)
  depending on the start and end values this movement needs ~2-5s, returns:
  'done eoa'

- 'read exitslit eoc'
  Returns:
  'current position: 1499.99934 eoa'
  In the very same way, you can use 'read photonenergy eoc', 'read mono eoc', 'read undugap'

- 'set exitslit 20 eoc'
  will set the exitslit gap to the desired position and wait until it is there, followed by:

‘done eoa’

- ‘set undugap 15000 eoc’
  will set the undulator gap to the desired position and wait until it is there, followed by:
  ‘done eoa’

- ‘set mono 300 eoc’
  will set the monochromator to the desired position and wait until it is there, followed by:
  ‘done eoa’

- ‘read keithley2 eoc’
  gives the keithley current (‘2’ since ‘1’ is the beamline Keithley while ‘2’ is dedicated to users)

- You can also perform on-the-fly photon energy scans meaning the undulator and monochromator are moving in parallel in terms of photon energies. You HAVE to submit the following values:
  - start energy in eV
  - stop energy in eV
  - scan speed in eV/s
  meaning for example
  ‘OTF photonenergy 300 -endposition=400 -speed=1 eoc’
  The range for the speed is limited to 0.5eV/s<scan speed<10eV/s. If the desired speed is lower than 0.5eV/s, the scan will be performed step-by-step and not on the fly.
  Assuming the start energy is lower than the end energy the scan will start 10*speed*s. Following the example above, the scan will start 300eV-10*1eV/s*s=290eV.
  Returns:
  ‘started eoa’

- During an OTF-scan you can check the status with
  ‘read status eoc’
  which will send you the answer:
  ‘timestamp monoenergy undulatorenergy keithley2value ringcurrent eoa’

- The helicity can be changed by
  ‘set helicity 1 eoc’
  or
  ‘set helicity -1 eoc’
  which will start a script to close the PS2 and move the undulator shift.
  You can also
  ‘read helicity eoc’
  which returns +1 or -1 and 0 if the state is undefined (‘current position: 1 eoa’ for example). If you are interested in the current undulatorshift value, you can also
  ‘read undushift eoc’

- ‘send helicity 1/-1 eoc’
  is also supported since ‘ComServer_V25.pyc’.

- ‘set undufactor 0.995 eoc’
  scales the undulator energy to 99.5% of the monochromator energy (in the photonenergy alias). The default value is 1, of course. When you change it one time, it will remain the new value until you change it again or the server is restarted.
  followed by:
  ‘done eoa’

- ‘read undufactor eoc'
  Returns:
  ‘current position: 1.0 eoa’

- ‘read ringcurrent eoc’
  will give you the ring current in mA
  ‘current position: 99.8765 eoa’

- ‘read pressure eoc’
  will give you the experiment-pressure

- ‘closeconnection eoc’
  disconnects your client from the beamlineserver. 0.1s afterwards, the server is ready for a new connection.

- New alias in V37:
  ‘slt2hleft'
  'slt2hright'
  'slt2vgap'
  ‘slt2voffset’
  'exsu2bpm'
  'exsu2baffle'
  ‘status2' (same as status but both keithleys)

- Please contact me if you need new parsers or commands!