

Channel Access Clients

Kay Kasemir
Jan. 2019

ORNL is managed by UT-Battelle, LLC for the US Department of Energy

Channel Access Operations

- 'connect'
- 'get', 'get-callback', 'monitor'
- 'put', 'put-callback'
- 'disconnect'

'connect'

- Based on channel name, UDP search, ..
 - See earlier Channel Access overview
- Connection might take a few seconds
- Network interruptions do happen
 - CA client libraries will automatically re-connect
 - API for being notified about connect/reconnect

'disconnect'

- A long-running program might
 - Connect PVs "A", "B", "C", use them for a while, disconnect
 - Connect PVs "X", "Y", "Z", use them for a while, disconnect

Cleanup resources on the CA server/IOC, reduce network traffic

Quick and dirty script may just connect, use, quit

‘get’, ‘get-callback’, ‘monitor’

- read, subscribe

- ‘get’
 - Request value
 - Wait for value (with timeout of ~ seconds)
- ‘get-callback’
 - Request value
 - Callback invoked once as value arrives
- ‘monitor’
 - Register for updates
 - Callback invoked whenever a value arrives, until monitor cancelled

'put', 'put-callback'

- write

- 'put'
 - Send a value
 - Error in case channel not connected right now, or no write access. Otherwise done.
- 'put-callback'
 - Send a value
 - Callback invoked once the value has been received by CA server and 'completed'.

For PV related to motor record on IOC, this can mean motor drive has been energized, motor moved to appropriate target location, then performed N adjustments to perfectly reach the desired position, and finally motor drive was de-energized.

Higher Level CA Client Bindings

- **Cached PVs**
 - When first asked for a channel, some PV object is created which connects and tracks connection state.
 - When your program later asks for the same PV, it gets that cached PV which is already connected
- **Subscribed**
 - PV always monitors. If you read the value of a PV, you get the most recent monitor right away
- **Write**
 - Writing a PV tends to be the plain 'put', a fire-and-forget that doesn't delay your program and will eventually reach the CA server/IOC

➔ Convenient and performant for most clients

Python Example

```
# Example for interacting with the 'fishtank' IOC
from time import sleep
from epics import caget, caput

sp = caget('training:setpoint')
rb = caget('training:tank')
print("Temperature setpoint is at %f and tank temperature is %f" % (sp, rb))

if sp < 40:
    caput('training:setpoint', 40)
    print("Changed setpoint to 40...")
    while True:
        rb = caget('training:tank')
        print("Tank temperature is %f" % rb)
        if rb > 39.5:
            break
        sleep(1)
else:
    caput('training:setpoint', 30)
    print("Changed setpoint to 30...")
    while True:
        rb = caget('training:tank')
        print("Tank temperature is %f" % rb)
        if rb < 30.5:
            break
        sleep(1)
print("Got there, I think")
```


Automation: This doesn't cut it

write 10 to "motor:setpoint"

read detector data

save data in file for "position10"

Your motor might still be moving while you take data


Automation: Not ideal, either

```
write 10 to "motor:setpoint"
```

```
while abs(read("motor:position") - 10) < 0.1:  
    sleep 1 second
```

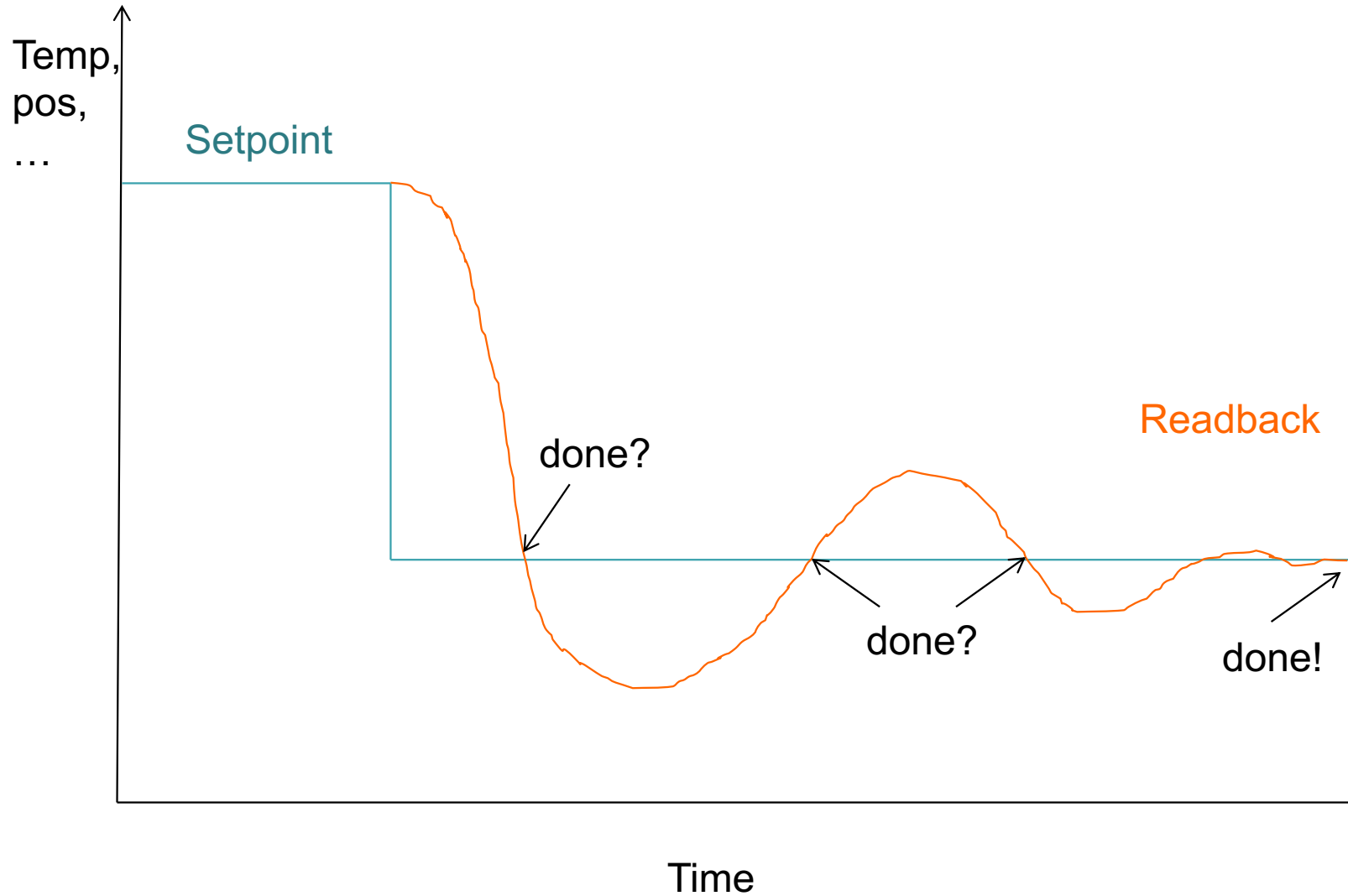
```
# OK, we're at the desired position  
read detector data
```

```
save data in file for "position10"
```



Maybe,
maybe not

Automation: How do you know you're "done"?



Automation: Use 'put-callback'

- Only logic on IOC really knows when it's done
 - Motor record: Performed initial backlash-compensated move, then retries, now motor is at rest and we're "done"
 - Lakeshore controller: Calc records, busy record, sequence knows when it's "done"
- With 'put-callback', IOC will tell you when it's "done"

Alas, no way to tell from the outside if a PV actually supports put-callback.

If not, put-callback returns right away even though the device is not "done".

Automation: Use explicit 'get', 'get-callback'

- In a higher level library, 'PV.read()' might only tell you about the last received monitor.
The actual record might have a new value that you will soon receive via a monitor, but unclear when exactly. Based on MDEL, you might never get the exact value.
- Explicit 'get' or 'get-callback' will force a round-trip read/response, fetching the most recent value.

Another Python Example

```
# Example for interacting with the 'fishtank' IOC
from time import sleep
from epics import caget, caput, camonitor, camonitor_clear

# The first call actually creates the PV and establishes a 'monitor'
print("Tank temperature is %f" % caget('training:tank'))

# Calling it again just returns the last received value
print("Tank temperature is %f" % caget('training:tank'))

# To force a 'get':
print("Tank temperature is %f" % caget('training:tank', use_monitor=False))

# Caput defaults to fire-and-forget
caput('training:setpoint', 30)

# To use put-callback and await completion:
caput('training:setpoint', 30, wait=True)

# To receive updates on received values
def handle_value_update(pvname, value, **kw):
    print("%s = %s" % (pvname, str(value)))
    # print("Stuff: " + str(kw))

camonitor('training:tank', callback=handle_value_update)
sleep(5)
camonitor_clear('training:tank')
```

Yet Another Python Example

More:

<https://github.com/pyepics/pyepics>

```
# Lower level API
import time, epics

# Create 3 PVs
pv1 = epics.get_pv('training:setpoint')
pv2 = epics.get_pv('training:room')
pv3 = epics.get_pv('training:sensor')

# Assert that we're connected
if not (pv1.wait_for_connection(5.0) and
        pv2.wait_for_connection(5.0) and
        pv3.wait_for_connection(5.0)):
    raise Exception("No luck")

# Submit 3 put-callbacks
pv1.put(30, use_complete=True)
pv2.put(25, use_complete=True)
pv3.put(0, use_complete=True)

# Wait for all of them to complete
while not (pv1.put_complete and
           pv2.put_complete and
           pv3.put_complete):
    time.sleep(0.5)

# Close PVs
pv1.disconnect()
pv2.disconnect()
pv3.disconnect()
```

Other CA Client Options

- Python 'PVA' client for the new PVAccess Protocol
 - Supports "CA Transport", so new API for both CA and PVA
- Matlab/Octave/Scilab MCA resp. LabCA bindings
- SNL/Sequencer
- Display Builder scripts
- Plain C code calling CA.lib
- Plain Java code calling JCA.jar

Summary

- Higher level bindings like python are pretty easy:
 - caget, caput
- For dependable automation, remember get/put-callback