EPICS → ADO Bridge

Contents:

EPICS vs ADO

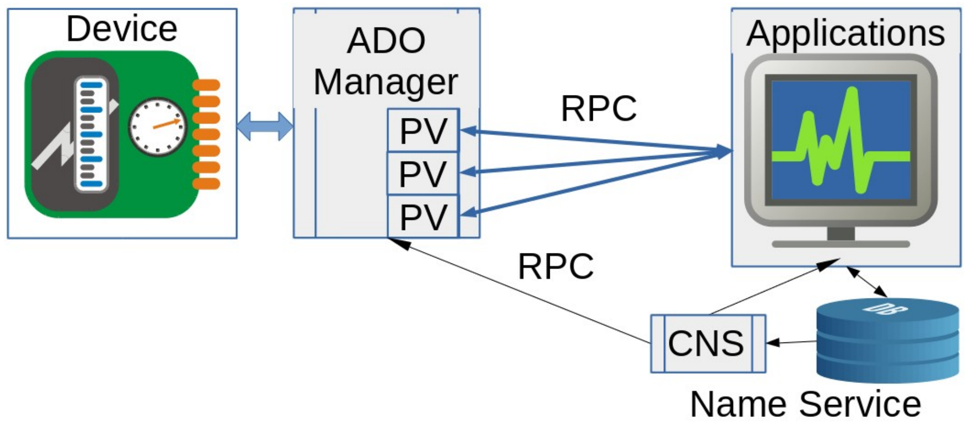


Fig. 1. RHIC Controls client-server concept

Naming conventions:
- Device:Parameter

!

ADO names are easy-translated to EPICS.

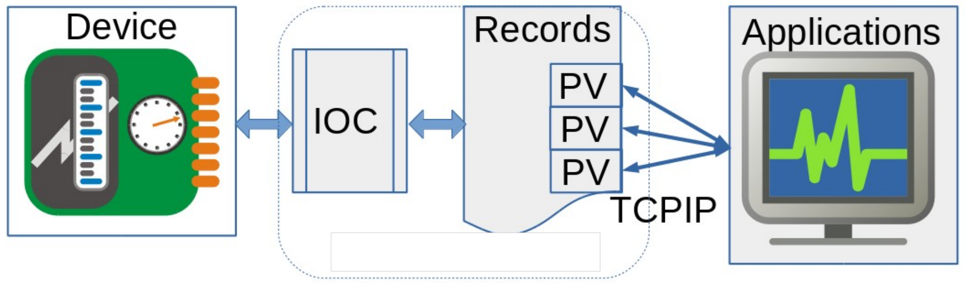


Fig. 2. EPICS client-server concept

Naming conventions:
- Flat namespace.
- No rules.

?

Time stamping supported:
A tuple of (seconds, nanoseconds)

The concepts of **records** and **state machine** in EPICS looks overcomplicated. They were introduced at times when CPUs were slow and memory was small.

The bridge is soft IOC

The bridge is ordinary soft IOC.
- One bridge can serve several ADOs.
- Implemented in python using **p4p**.

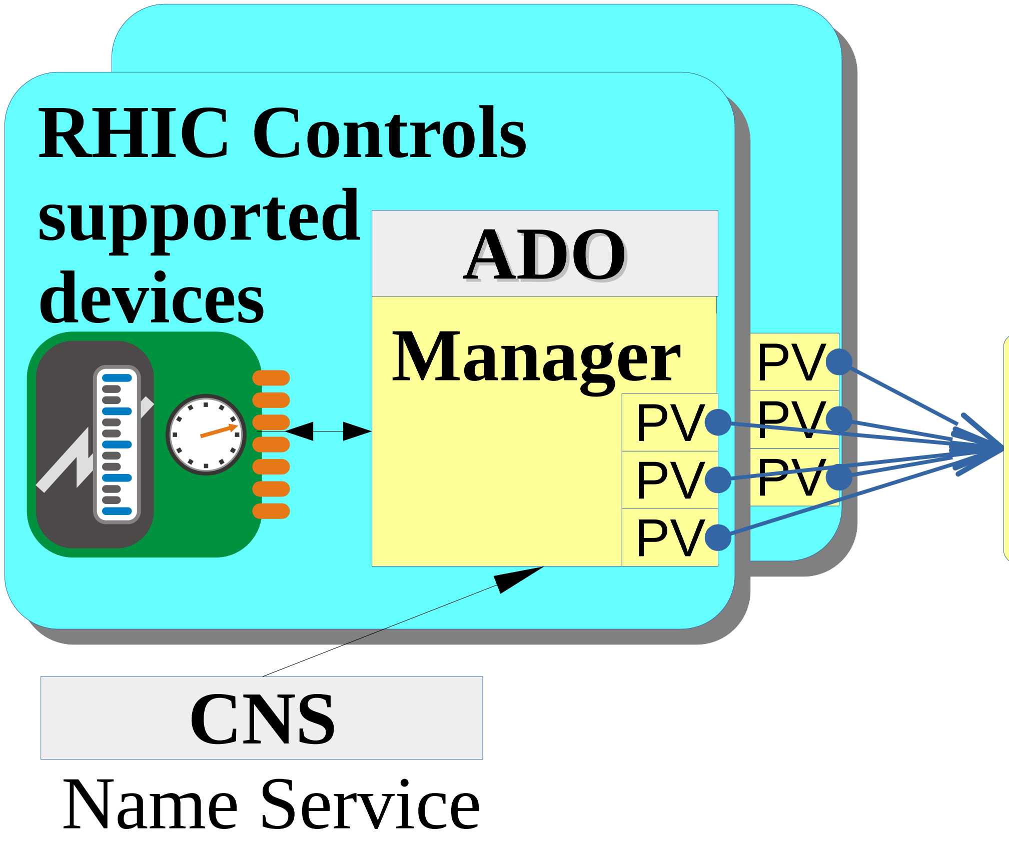
p4p: python binding of PVAccess protocol.
- Very well organized (developed by M. Davidsaver, 2023).
- **Easy installation using pip**.
- The PVs are treated very similar to ADO parameters:
  - PV is defined by its properties. There is **no records and state machine**.
  - 4 ways to access PV:
    - get() → get()
    - put() → set()
    - monitor() → subscribe()
    - post() → publish()
- EPICS Normative Types are tightly bound to NumPy arrays.
- Fast.

Disadvantages:
- Info() request is not supported.
- Documentation is scarce.

Bridge flowchart

<u>Create PVs</u>:
- Translate EPICS properties to ADO,
- Supply writable parameters with setters,
  - Handle legalValues,

<u>Subscription:</u>
```
for par in ADOpars:
   subscribe(callback,par)
```



```
def callback(args):
    for ADOPar,value in args:
        pv(ADOPar).post(value)
```

Configuration

The input for the bridge could be:

- A list of ADOs:
  `adoEpics -p'prefix1:' -a am_simple.1 simple.test`
- A list of python scripts, containing translation map:
  `adoEpics -p'prefix2:' -f a2e_am_simple.py a2e_simple.py`

Translation map:

Is a python file, which will be imported during startup.
It should contain **translationMap** variable:

```
ranslationMap = {
'ado1:parameter1': {},  # Automatic (native) translation.
'ado1:parameter2': {map_of_properties} # Specific translation.
…
```

$\mathtt{ado}_X\mathtt{:parameter}_Y\mathtt{':}\ \ (\mathtt{map\_of\_properties}_z)$

```
}
```

Map_of_properties defines the PV properties, which will be used instead of native ones.
Currently supported properties:
name, desc, units, opLow, opHigh, engLow, engHigh.

Example of translation map file

```
# Definitions
ro = {'readOnly':True}
units = 'units'
name = 'name'
desc = 'desc'# description

# Translation map
TranslationMap = {
'simple.test:timerIntervalS':{name:'simple:updatePeriod',
                     desc:'Update period', units:'s'},
'simple.test:alarmEnableS':{desc:'Enable alarm', **ro},
'simple.test:stringS':{desc:'String Setting'},
'simple.test:doubleS':{},
# 'simple.test:timerEnableS',
# 'simple.test:degM',
 'simple.test:sinM':{},
}
```

Summary

Fully functional bridge have been released as **adoEpics** program.
Gitlab project: https://gitlab.pbn.bnl.gov/python/cli-tools/adoEpics

Performance:
Bridge crossing time on the same host: ~1 ms.
It is time when PV value have appeared in monitor callback relative to time when it was modified in ADO.

Todo:
1) Translate alarm properties.
2) Implement **PPM** user as command line option.