OAK RIDGE National Laboratory

Easy integration of Python into EPICS IOC

Klemen Vodopivec

October 19th, 2020

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

U.S. DEPARTMENT OF ENERGY

Introduction

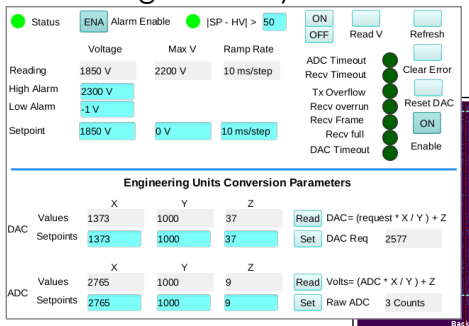
- PyDevice is EPICS device support for Python interpreter
  - But not tight integration with EPICS database (check pydevsup)
- Focus on easy integration of Python code into EPICS ecosystem
  - Quick prototyping
  - Expand record functionality with Python
  - Utilize extensive Python ecosystem through modules
  - Simple for Python developers to integrate code into IOC
- Used at SNS instruments as replacement for PCASpy
  - Common IOC setup, maintenance, expertise
  - Improved CA reliability
  - Scientific data processing

Motivation – EPICS for Python developers

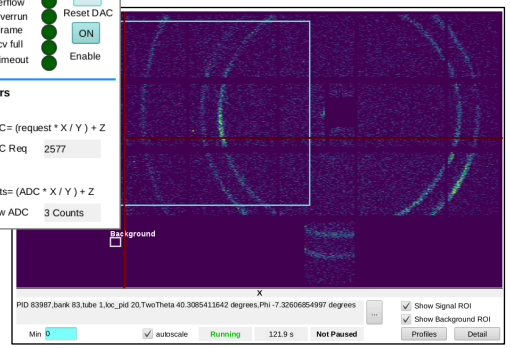
Instrument scientists using Python for data analysis would like to tap into control system and drive the experiment based on scientific analysis.


```
def get_expected_for_spectrum_n(data_workspace, selected_workspace_
    # Expected output.
    # 1. The background correction should produce [0., value., valu
    # 2. Conversion to wavelength changes the x axis to approximate
    #                                        [2.7, 5.5, 8.2, 10.9
    #                                        [1.0, 2.0, 3.0, 4.1,
    # 3. Rebinning from creates the x steps [2, 4, 6, 8]
    # This means for spectrum 1:
    #   Bin0:  0
    #   Bin1: (0 + abs(5.5-6.0)/abs(5.5 - 8.2))*value
    #   Bin2: (abs(6.0-8.0)/abs(5.5 - 8.2))*value
    # This means for spectrum 3:
    #   Bin0: (1. + abs(3.0-4.0)/abs(3.0 - 4.1))*value
    #   Bin1: (abs(4.0-4.1)/abs(3.0 - 4.1) + 1.)*value
    #   Bin2: 0

    # The second bin should have abs(4.0-4.7)/abs(3.2 - 4.7)*value
    # The third bin should have abs(6.0-6.3)/abs(4.7 - 6.3)*value +
    instrument = data_workspace.getInstrument()
    source = instrument.getSource()
    detector = data_workspace.getDetector(selected_workspace_index)
    distance_source_detector = detector.getDistance(source)
    h = 6.62606896e-34
    mass = 1.674927211e-27
    times = data_workspace.dataX(0)
    lambda_after_unit_conversion = [(time * 1e-6) * h / distance_so
    expected_lambda = [2., 4., 6., 8.]
    if selected_workspace_index == 0:
        expected_signal = [0.,
                           abs(lambda_after_unit_conversion[1] - ex
                           / abs(lambda_after_unit_conversion[1] -

                           (abs(expected_lambda[3] - expected_lambd
                            / abs(lambda_after_unit_conversion[1] -
```

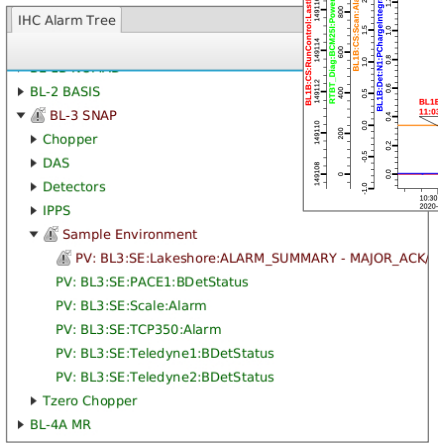
Interfacing control system



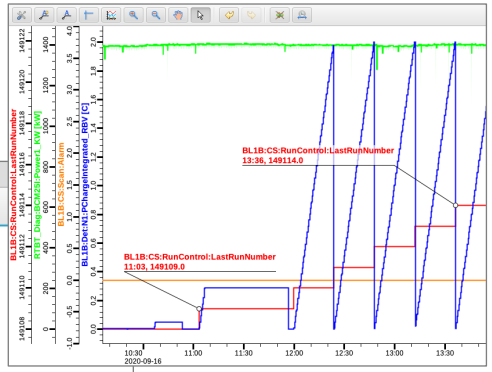
Plot scientific data



Alarming



Archived data



OAK RIDGE National Laboratory | SPALLATION NEUTRON SOURCE

Goal – Python for EPICS gurus

Control system engineer looking to simplify tasks by leveraging Python functionality.

```
44 record(mbbi, "$(P)PulseFlavor")
45 {
46     field(DESC, "Pulse flavor")
47     field(DTYP, "asynInt32")
48     field(INP,  "@asyn($(PORT))PulseFlavor")
49     field(SCAN, "I/O Intr")
50     field(ZRVL, "0")
51     field(ZRST, "No beam")
52     field(ONVL, "1")
53     field(ONST, "Target 1")
54     field(TWVL, "2")
55     field(TWST, "Target 2")
56     field(THVL, "3")
57     field(THST, "10us diagnostic")
58     field(FRVL, "4")
59     field(FRST, "50us diagnostic")
60     field(FVVL, "5")
61     field(FVST, "100us diagnostic")
62     field(SXVL, "6")
63     field(SXST, "Physics 1")
64     field(SVVL, "7")
65     field(SVST, "Physics 2")
66     #field(FLNK, "$(P)PrevPulseFlavor")
67 }
68 record(calc, "$(P)PrevPulseFlavor")
69 {
70     field(INPA, "$(P)PulseFlavor NPP")
71     field(CALC, "B; B:=A")
72 }
73 record(ai, "$(P)ProtonCharge")
74 {
75     field(DESC, "Pulse charge")
76     field(DTYP, "asynFloat64")
77     field(INP,  "@asyn($(PORT))ProtonCharge")
78     field(EGU,  "C")
79     field(PREC, "12")
80     field(SCAN, "I/O Intr")
81 }
```

Expand record functionality

Access to Python modules

python™

Modularization

Extended data processing

Control system experience not required

Teaser examples for EPICS audience

```
record(stringout, ”Example:Log") {
  field(DTYP, "pydev")
  field(OUT,  "@print(dt.today.isoformat()+’ %VAL%')")
}
```

```
record(ao, ”Example:Save") {
  field(DTYP, "pydev")
  field(OUT,  "@myarray.append(%VAL%)")
  field(FLNK, ”Example:Histogram")
}

record(waveform, ”Example:Histogram") {
  field(DTYP, "pydev")
  field(INP,  "@numpy.histogram(myarray, bins=100)[0]")
  field(NELM, "100")
  field(FTVL, ”FLOAT")
}
```

Check slide #10 for initialization of myarray and numpy

Assume
from socket import *

```
record(longout, ”Example:Tcp:Socket") {
  field(DTYP, "pydev")
  field(OUT,  "@dev=socket(AF_INET, SOCK_STREAM)")
  field(PINI, "YES")
  field(FLNK, "PyDev:Tcp:Connect")
}
record(longout, ”Example:Tcp:Connect") {
  field(DTYP, "pydev")
  field(OUT,  "@dev.connect((‘10.2.3.4', 4012)")
}
record(stringout, ”Example:Tcp:RequestIDN") {
  field(DTYP, "pydev")
  field(OUT,  "@dev.send('*IDN?')")
  field(FLNK, "PyDev:Tcp:IDN")
}
record(stringin, ”Example:Tcp:IDN") {
  field(DTYP, "pydev")
  field(INP,  "@dev.recv(39)")
}
```

Record support

- DTYP must be **pydev**
- INP or OUT field must be prefixed with **@** followed by Python expression or statement
- On-the-fly replacement of macros with live values
  - `field(OUT, "@print('%NAME%=%VAL%')")`
- Python exceptions translate into SEVR & STAT
- Value types are converted between record and Python type
  - ie. value from longout record produces *int* type in Python
  - Array from waveform record is converted to Python list, length based on number of actual elements

Supported records

- Input records: ai, longin, bi, mbbi, stringin
  - INP field must contain a Python expression which evaluates to a value
  - Expression examples: *2+2, "foo"+"bar", min(2, 22), 10 if x >10 else x*
- Output records: ao, longout, bo, mbbo, stringout, waveform
  - OUT field can contain a Python expression or a statement
  - Statement examples: *a=2, non_return_fnc(), import module*
  - But expressions also work

```
record(ao, "Example:AbsValue") {
  field(DTYP, "pydev")
  field(OUT,  "@abs(%VAL%)")
}
```

```
$ caput -c Example:AbsValue -3.3
Old : Example:AbsValue                0
New : Example:AbsValue                3.3
```

An expression evaluates to a value.
A statement does something.

Notes on custom Python code

- No major implications to Python coding
- Asynchronous callback model that supports put-callback
  - No main thread
  - Python functions are invoked when record processes
  - Their return value (if any) is assigned to record
- Requests are queued in FIFO, processed by single thread
  - CA thread will not be blocked while Python code is executing
  - If needed, worker threads can be started from Python code

- Functions can be called with at most one value from a record
  - But static values and macros are supported
- Functions must return a convertible type

Publish values from Python code

- Support for *I/O Intr* scanning
  - Subscribe in record: pydev.iointr(id)
  - Publish from Python: pydev.iointr(id, val)
- Available from global context
  - No pydev imports required
  - Unless testing Python code outside PyDevice environment
- Use cases
  - More than one return value
  - Asynchronous operation from Python thread

```
import math
def sqrt(value):
  ival = math.isqrt(value)
  pydev.iointr('sqrt_int', ival)
  pydev.iointr('sqrt_remain', value - ival*ival)
```

```
record(longout, "Example:Num") {
  field(DTYP, "pydev")
  field(OUT,  "@sqrt(%VAL%)")
}
record(longin, "Example:SqrtInteger") {
  field(DTYP, "pydev")
  field(INP,  "@pydev.iointr('sqrt_int')")
  field(SCAN, "I/O Intr")
}
record(ai, "Example:SqrtRemain") {
  field(DTYP, "pydev")
  field(OUT,  "@pydev.iointr('sqrt_remain')")
  field(SCAN, "I/O Intr")
}
```

Interfacing through IOC shell

- Set up Python search path for custom modules
  - `epicsEnvSet("PYTHONPATH", "$(TOP)/python/")`
- Initialize resources before iocInit()
  - `pydev("import numpy")`
  - `pydev("myarray = list()")`

Needed by Teaser Example #2

- Call custom Python code when debugging
  - `pydev("MyProject.setLoggingLevel(10)")`
  - Executes in IOC's main thread but serialized by GIL

Adding PyDevice support to existing IOC

Add to **configure/RELEASE**

```
PYDEVICE=/path/to/PyDevice
```

Add to **iocApp/src/Makefile**

```
ioc_DBD += pydev.dbd
ioc_LIB += pydev

SYS_PROD_LIBS += $(shell python-config --ldflags | sed 's/-[^l][^ ]*//g' | sed 's/-l//g')
```

Python code can now be executed from IOC shell or Database records.

IOC rebuild is only required for changes in Database, not for Python code updates.

Custom Python files must be in the search path, consider updating PYTHONPATH.

Dependencies

- Used in production
  - RHEL 7
  - Python 2.7
  - EPICS 3.14.12.6
- Builds with *(but help with testing is appreciated)*
  - Python 3
  - EPICS 7.x

Questions

- http://github.com/klemenv/PyDevice