Using Python for Epics Channel Access: Library and Applications

Matthew Newville

Consortium for Advanced Radiation Sciences
University of Chicago

September 15, 2011

http://github.com/pyepics/

Why Python?

For General Programming:

Clean Syntax	Easy to learn, remember, and read
High Level	No pointers, dynamic memory, automatic memory
Cross Platform	code portable to Unix, Windows, Mac.
Object Oriented	full object model, name spaces.
Easily Extensible	with C, C++, Fortran, . . .
Many Libraries	GUIs, Databases, Web, Image Processing, . . .

For Scientific Applications:

numpy	Fast arrays.
matplotlib	Excellent Plotting library
scipy	Numerical Algorithms (FFT, lapack, fitting, . . .)
sage	Symbolic math (ala Maple, Mathematica)
. . .	... new scientific packages all the time . . . .
Free	Python and all these tools are Free (BSD, LGPL, GPL).

PyEpics: Epics Channel Access for Python

PyEpics contains 3 levels of access to CA:

Low level: ca and dbr modules. C-like API, complete mapping of CA library.

High level: PV object. Built on ca module.

Procedural: caget(), caput(), cainfo(), camonitor(), Built on PV.

Other objects (Alarm, Devices, GUI controls) are built on top of PV.

Procedural Interfaces: similar to command-line tools or EZCA library.

caget() / caput()

```
>>> from epics import caget, caput

>>> m1 = caget(’XXX:m1.VAL’)
>>> print m1
-1.2001

>>> caput(’XXX:m1.VAL’, 0)

>>> caput(’XXX:m1.VAL’, 2.30, wait=True)

>>> print caget(’XXX:m1.DIR’)
1

>>> print caget(’XXX:m1.DIR’, as_string=True)
’Pos’
```

caput(pvname, wait=True) waits until processing completes.

caget(pvname, as_string=True) returns String Representation of value (Enum State Name, formatted doubles)

Does 90% of what you need.
Probably too easy ...

cainfo() and camonitor()

cainfo() shows many informational fields for a PV:

cainfo()

```
>>> cainfo('XXX.m1.VAL')
== XXX:m1.VAL  (double) ==
   value      = 2.3
   char_value = 2.3000
   count      = 1
   units      = mm
   precision  = 4
   host       = xxx.aps.anl.gov:5064
   access     = read/write
   status     = 1
   severity   = 0
   timestamp  = 1265996455.417 (2010-Feb-12 11:40:55.417)
   upper_ctrl_limit    = 200.0
   lower_ctrl_limit    = -200.0
   upper_disp_limit    = 200.0
   lower_disp_limit    = -200.0
   upper_alarm_limit   = 0.0
   lower_alarm_limit   = 0.0
   upper_warning_limit = 0.0
   lower_warning       = 0.0
   PV is monitored internally
   no user callbacks defined.
=============================
```

camonitor() monitors a PV, writing out a message for every value change, until camonitor_clear() is called:

camonitor()

```
>>> camonitor('XXX:DMM1Ch2_calc.VAL')
XXX:DMM1Ch2_calc.VAL 2010-02-12 12:12:59.502945 -183.9741
XXX:DMM1Ch2_calc.VAL 2010-02-12 12:13:00.500758 -183.8320
XXX:DMM1Ch2_calc.VAL 2010-02-12 12:13:01.501570 -183.9309
XXX:DMM1Ch2_calc.VAL 2010-02-12 12:13:02.502382 -183.9285

>>> camonitor_clear('XXX:DMM1Ch2_calc.VAL')
```

You can supply your own callback to camonitor() to do something other than write out the new value.

The epics module maintains a global cache of PVs when using the ca***() functions: connections to underlying PVs are maintained for the session.

PV objects: Easy to use, full-featured.

PV objects are good way to interact with Channel Access Process Variables:

Using PV objects

```
>>> from epics import PV
>>> pv1 = PV(’XXX:m1.VAL’)
>>> print pv1.count, pv1.type
(1, ’double’)

>>> print pv1.get()
-2.3456700000000001

>>> pv1.put(3.0)

>>> pv1.value = 3.0  # = pv1.put(3.0)

>>> pv1.value   # = pv1.get()
3.0
>>> print pv.get(as_string=True)
’3.0000’

>>> # user defined callback
>>> def onChanges(pvname=None, value=None, **kws):
...     fmt = ’New Value for %s  value=%s\n’
...     print fmt % (pvname, str(value))

>>> # subscribe for changes
>>> pv1.add_callback(onChanges)
>>> while True:
...     time.sleep(0.001)
```

- Automatic connection management.
- Attributes for many properties (count, type, host,upper_crtl_limit, . . .)
- Can use get() / put() methods
- . . . or PV.value attribute.
- as_string uses ENUM labels or precision for doubles.
- put() can wait for completion or run user callback when complete.
- connection callbacks.
- can have multiple event callbacks.

User-Supplied Callbacks for PV Changes

Callback: User-defined function called when a PV changes.
The function must have a pre-defined call signature, using keyword arguments:

Simple Callback

```
import epics
import time
def onChanges(pvname=None, value=None,
              char_value=None, **kw):
    print ’PV Changed! ’, pvname, \
          char_value, time.ctime()

mypv = epics.PV(pvname)

# Add a callback
mypv.add_callback(onChanges)

print ’Now watch for changes for a minute’

t0 = time.time()
while time.time() - t0 < 60.0:
    time.sleep(1.e-3)

print ’Done.’
```

pvname	name of PV
value	new value
char_value	String representation of value
count	element count
ftype	field type (DBR integer)
type	python data type
status	ca status (1 == OK)
precision	PV precision
	. . .

Many CTRL values (limits, units, . . .) passed in.
Use `**kws` recommended!

Callbacks for the ca module have similar signatures (no CTRL parameters).

`put()` and connection callbacks have similar signatures.

PVs for Waveform / Array Data

Epics Waveform array data is very important for experimental data:

double waveform

```
>>> p1vals = numpy.linspace(3, 4, 101)

>>> scan_p1 = PV(’XXX:scan1.P1PA’)
>>> scan_p1.put(p1vals)

>>> print scan_p1.get()[:101]
[3.  , 3.01, 3.02, ..., 3.99, 3.99, 4.]
```

For recent versions of Epics base, sub-arrays are supported.

character waveform

```
>>> folder = PV(’XXX:directory’)
>>> print folder
<PV ’XXX:directory’, count=21/128,
                 type=char, access=read/write>

>>> folder.get()
array([ 84,  58,  92, 120,  97, 115,  95, 117, 115,
     101, 114,  92,  77,  97, 114,  99, 104,  50,  48,
      49,  48])

>>> folder.get(as_string=True)
’T:\xas_user\March2010’

>>> folder.put(’T:\xas_user\April2010’)
```

Character waveforms can be longer than 40 characters – useful for long strings.

Putting a Python string to a character waveform will convert to a list of bytes.

PyEpics History and Motivation

There have been several wrappings of Epics CA over the years.

Sept 2009: a tech-talk discussion asked if these could be combined.

My own was difficult to maintain (especially Windows), so I rewrote from scratch.

Goals for Python/Channel Access interface:

- complete(ish) access to low-level CA.
- high-level PV object built upon this foundation.
- support for multi-threading.
- preemptive callbacks: PV connection, event, put.
- documentation, unit-testing, maintenance.
- easy installation – including Windows.
- Python 2 and Python 3 support.

Key Design Decision: *Use Python's ctypes module*.

> PyEpics wraps the CA library, a C library that preemptively calls user-supplied Python code and accesses complex C data structures.
>
> *Zero lines of C*

Using ctypes

The ctypes library is a foreign function interface, giving access to C data types and functions in dynamic libraries at Python run-time.

ctypes for libca.so (low-level CA)

```
import ctypes

libca = ctypes.cdll.LoadLibrary(’libca.so’)
libca.ca_context_create(1)

chid  = ctypes.c_long()

libca.ca_create_channel(’MyPV’, 0,0,0, ctypes.byref(chid))
libca.ca_pend_event.argtypes = [ctypes.c_double]
libca.ca_pend_event(1.0e-3)

print ’Connected: ’, libca.ca_state(chid) == 2 # (CS_CONN)
print ’Host Name: ’, libca.ca_host_name(chid)
print ’Field Type: ’, libca.ca_field_type(chid)
print ’Element Count: ’, libca.ca_element_count(chid)
```

Ctypes gives a “just like C” interface to a dynamic library.

- Load library
- Create Channel ID
- Use Channel ID with library functions, being careful about data types for arguments.

Using ctypes makes several goals easy:

1. Complete CA interface easy to implement, debug.
2. Install on all systems: `python setup.py install`.
3. Best thread support possible, with Python Global Interpreter Lock.
4. Supports Python 2 **and** Python 3 with little code change.

ca module: low-level, but still Python

Wrapping CA with ctypes:

The ca interface

```
from epics import ca
chid  = ca.create_channel(’XXX:m1.VAL’)
count = ca.element_count(chid)
ftype = ca.field_type(chid)
value = ca.get()

print "Channel ", chid, value, count, ftype

# put value
ca.put(chid, 1.0)
ca.put(chid, 0.0, wait=True)

# user defined callback
def onChanges(pvname=None, value=None, **kw):
    fmt = ’New Value for %s  value=%s\n’
    print fmt % (pvname, str(value))

# subscribe for changes
eventID = ca.create_subscription(chid,
                                 userfcn=onChanges)
while True:
    time.sleep(0.001)
```

Enhancements for Python:

- Python namespaces, exceptions used.
  - **ca_fcn** → **ca.fcn**
  - **DBR_XXXX** → **dbr.XXXX**
  - `SEVCHK` → Python exceptions
- OK to forget many tedious chores:
  - initialize CA.
  - create a context (unless explicitly using Python threads).
  - wait for connections.
  - clean up at exit.
- No need to worry about data types.

Python decorators are used to lightly wrap CA functions so that:

- CA is initialized, finalized.
- Channel IDs are valid, and connected before being used.

CA interface design choices

Essentially all CA functions are defined to work "Just like C". A few details:

- Preemptive Callbacks are used by default. OK to forget `ca.pend_event()`. Can be turned off, but only before CA is initialized.
- DBR_CTRL and DBR_TIME data types supported, but not DBR_STS or DBR_GR.
- Array data will be converted to numpy arrays if possible.
- Some functions (`ca_set_puser()`, `ca_add_exception_event()`) are not needed.
- EPICS_CA_MAX_ARRAY_BYTES set to 16777216 (16Mb) unless already set.
- Connection and Event callbacks are (almost) always used internally. User-defined callback functions are called by the internal callback.
- Event Callbacks are used internally except for large arrays, as defined by ca.AUTOMONITOR_LENGTH (default = 16K).
- Event subscriptions use mask = (EVENT | LOG | ALARM) by default.

Devices, GUI components, and Applications

Devices: collections of PVs

A *PyEpics Device* is a collection of PVs, usually sharing a Prefix.
Similar to an Epics Record, but relying on PV names, not Record definition.

Epics Analog Input as Python epics.Device

```
import epics
class ai(epics.Device):
    "Simple analog input device"
    _fields = ('VAL', 'EGU', 'HOPR', 'LOPR', 'PREC',
    'NAME', 'DESC', 'DTYP', 'INP', 'LINR', 'RVAL',
    'ROFF', 'EGUF', 'EGUL', 'AOFF', 'ASLO', 'ESLO',
    'EOFF', 'SMOO', 'HIHI', 'LOLO', 'HIGH', 'LOW',
    'HHSV', 'LLSV', 'HSV', 'LSV', 'HYST')

    def __init__(self, prefix, delim='.'):
        epics.Device.__init__(self, prefix, delim=delim,
                              self._fields)
```

A `Device` maps a set of PV "fields" (name "suffixes") to object attributes, holding all the associated PVs.

Can save / restore full state.

Using an ai device

```
 >>> from epics.devices import ai
>>> Pump1 = ai('XXX:ip2:PRES')
>>> print "%s = %s %s" % (Pump1.DESC,
                          Pump1.get('VAL',as_string=True),
                          Pump1.EGU  )
Ion pump 1 Pressure = 4.1e-07 Torr
>>> print Pump1.get('DTYP', as_string=True)
asyn MPC
>>> Pump1.PV('VAL') # Get underlying PV
<PV 'XXX:ip1:PRES.VAL', count=1, type=double, access=read/write>
```

Can use `get()`/`put()` methods or attribute names on any of the defined fields.

Extending PyEpics Devices

And, of course, a *Device* can have methods added:

Scaler device

```
import epics
class Scaler(epics.Device):
    "SynApps Scaler Record"

    ...
    def OneShotMode(self):
        "set to one shot mode"
        self.CONT = 0

    def CountTime(self, ctime):
        "set count time"
        self.TP = ctime
    ...
```

Add Methods to a Device to turn it into a high-level Objects.

Can also include complex functionality – dynamically, and from client (beamline).

Long calculations, database lookups, etc.

Using a Scaler:

```
s1 = Scaler('XXX:scaler1')
s1.setCalc(2, '(B-2000*A/10000000.)')
s1.enableCalcs()
s1.OneShotMode()
s1.Count(t=5.0)
print 'Names:        ', s1.getNames()
print 'Raw  values: ', s1.Read(use_calcs=False)
print 'Calc values: ', s1.Read(use_calcs=True)
```

Simple Example: Read Ion Chamber current, amplifier settings, x-ray energy, compute photon flux, post to a PV.

Needs table of coefficients (∼16kBytes of data), but then ∼100 lines of Python.

Motor and other included Devices

A Motor Device has ~100 fields, and several methods to move motors in User, Dial, or Raw units, check limits, etc.

Using a Motor

```
>>> from epics import Motor
>>> m = Motor(’XXX:m1’)
>>> print ’Motor:  ’, m1.DESC , ’ Currently at ’, m1.RBV

>>> m1.tweak_val = 0.10 # or m1.TWV = 0.10

>>> m1.move(0.0, dial=True, wait=True)

>>> for i in range(10):
>>>     m1.tweak(dir=’forward’, wait=True)
>>>     print ’Motor: ’, m1.DESC , ’ at ’, m1.RBV

>>> print m.drive, m.description, m.slew_speed
1.030 Fine X 5.0

>>> print m.get(’device_type’, as_string=True)
’asynMotor’
```

Motor features:

- `get()` and `put()` methods for all attributes
- `check_limits()` method.
- `tweak()` and `move()` methods.
- Can use Field suffix (.VELO, .MRES) or English description (slew_speed, resolution).

Other devices included in the main distribution:
ao, ai, bi, bo,transform, scaler, struck (for multi-channel scaler), mca.

Alarms: react to PV values

An alarm defines user-supplied code to run when a PV's value changes to some condition. Examples might be:

- send email, or some other alert message
- turn off some system (non-safety-critical, please!)

Epics Alarm

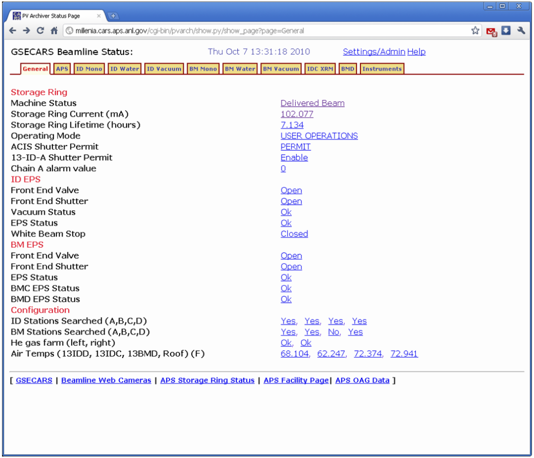
```
from epics import Alarm, poll

def alertMe(pvname=None, char_value=None, **kw):
    print "Soup is on!   %s = %s" % (pvname, char_value)

my_alarm = Alarm(pvname = 'WaterTemperature.VAL',
                 comparison = '>',
                 callback = alertMe,
                 trip_point = 100.0,
                 alert_delay = 600)
while True:
    poll()
```

When a PV's value matches the **comparison** with the **trip_point**, the supplied **callback** is run.
A delay is used to prevent multiple calls for values that "bounce around".

Epics Data Archiver – Epics+Python+MySQL+Apache



GSECARS Beamline Status: Thu Oct 7 13:31:18 2010 Settings/Admin Help

Storage Ring	
Machine Status	Delivered Beam
Storage Ring Current (mA)	102.077
Storage Ring Lifetime (hours)	7.134
Operating Mode	USER OPERATIONS
ACIS Shutter Permit	PERMIT
13-ID-A Shutter Permit	Enable
Chain A alarm value	0
ID EPS	
Front End Valve	Open
Front End Shutter	Open
Vacuum Status	Ok
EPS Status	Ok
White Beam Stop	Closed
BM EPS	
Front End Valve	Open
Front End Shutter	Open
EPS Status	Ok
BMC EPS Status	Ok
BMD EPS Status	Ok
Configuration	
ID Stations Searched (A,B,C,D)	Yes, Yes, Yes, Yes
BM Stations Searched (A,B,C,D)	Yes, Yes, No, Yes
He gas farm (left, right)	Ok, Ok
Air Temps (13IDD, 13IDC, 13BMD, Roof) (F)	68.104, 62.247, 72.374, 72.941

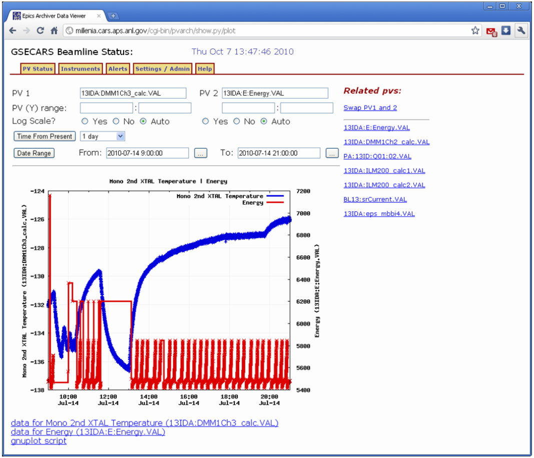
[GSECARS | Beamline Web Cameras | APS Storage Ring Status | APS Facility Page| APS OAG Data]

PV values displayed as html links to Plot of Data

Main features:

- Web interface to current PV values.
- $\gtrsim$ 5000 PVs monitored
- Data archived to MySQL tables.
- templates for status web pages
- plots of historical data
- web-definable email alerts

Epics Archiver: Plotting Historical Data



Epics Archiver Data Viewer

millenia.cars.aps.anl.gov/cgi-bin/pvarch/show.py/plot

GSECARS Beamline Status: Thu Oct 7 13:47:46 2010

PV Status | Instruments | Alerts | Settings / Admin | Help

PV 1: 13IDA:DMM1Ch3_calc.VAL — PV 2: 13IDA:E:Energy.VAL

PV (Y) range: : — :

Log Scale? ○ Yes ○ No ● Auto — ○ Yes ○ No ● Auto

Time From Present: 1 day

Date Range — From: 2010-07-14 9:00:00 ... To: 2010-07-14 21:00:00 ...

Related pvs:

- Swap PV1 and 2
- 13IDA:E:Energy.VAL
- 13IDA:DMM1Ch2_calc.VAL
- PA:13ID:Q01:02.VAL
- 13IDA:ILM200_calc1.VAL
- 13IDA:ILM200_calc2.VAL
- BL13:srCurrent.VAL
- 13IDA:eps_mbbi4.VAL

data for Mono 2nd XTAL Temperature (13IDA:DMM1Ch3_calc.VAL)
data for Energy (13IDA:E:Energy.VAL)
gnuplot script

Plots:

- default to past day
- using Gnuplot (currently)
- Plot "From now" or with "Date Range"
- Plot up to 2 PVs
- "Related PVs" list for common pair plots
- pop-up javascript Calendar for Date Range
- String labels for Enum PVs

GUI Controls with wxPython

Many PV types (Double, Float, String, Enum) have wxPython widgets, which automatically tie to the PV.

Sample wx widget Code

```
from epics import PV
from epics.wx import wxlib

txt_wid = wxlib.pvText(Parent, pv=PV('SomePV'),
                        size=(100,-1))

txtCtrl_wid = wxlib.pvTextCtrl(Parent, pv=PV('SomePV'))


dropdown_wid = pvEnumChoice(Parent, pv=PV('EnumPV.VAL'))

buttons_wid = pvEnumButtons(Parent, pv=PV('EnumPV.VAL'),
                            orientation=wx.HORIZONTAL)

flt_wid = wxlib.pvFloatCtrl(Parent, size=(100, -1),
                             precision=4)
flt_wid.set_pv(PV('XXX.VAL'))
```

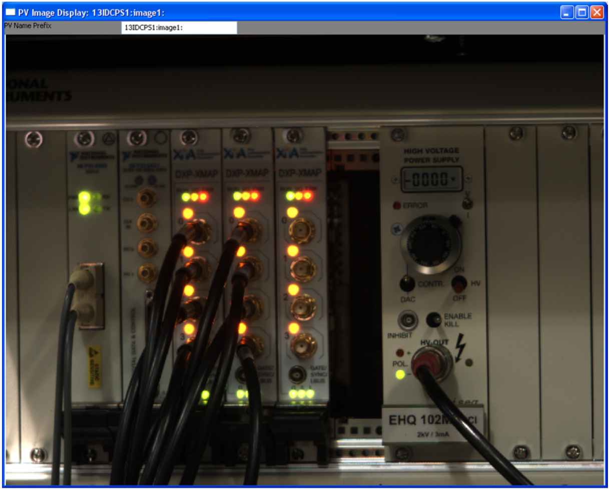
- pvText read-only text for Strings
- pvTextCtrl editable text for Strings
- pvEnumChoice Drop-Down list for ENUM states.
- pvEnumButtons Button sets for ENUM states.
- pvAlarm Pop-up message window.
- pvFloatCtrl editable text for Floats, only valid numbers that obey limits.
- Others: Bitmap, Checkboxes, Buttons, Shapes, etc

Mixin classes help extending other widgets (Many from Angus Gratton, ANU).

Function Decorators help write code that is safe against mixing GUI and CA threads.

Some Epics wxPython Apps:

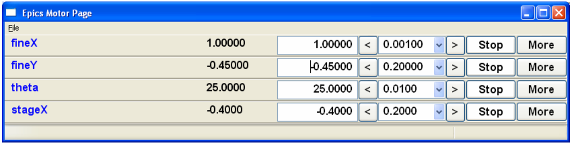
Simple Area Detector Display:



A 1360 × 1024 RGB image (4Mb) from Prosilica GigE camera.

Not super-fast: Can display at a few Hz. Can display a selected ROI at a much faster

wx Motor Controls



Epics Motor Page

File

fineX	1.00000	1.00000	<	0.00100	>	Stop	More
fineY	-0.45000	-0.45000	<	0.20000	>	Stop	More
theta	25.0000	25.0000	<	0.0100	>	Stop	More
stageX	-0.4000	-0.4000	<	0.2000	>	Stop	More

MEDM-like Motor Display, except

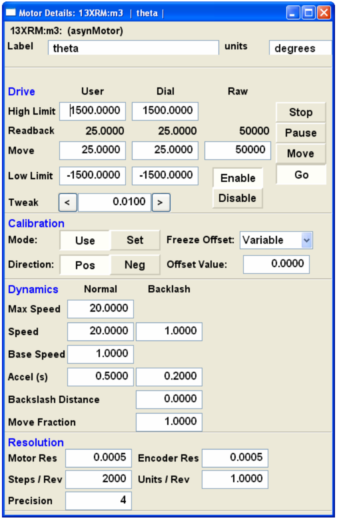
much easier to use.

Entry Values can only be valid number. Values outside soft limits are highlighted.

Tweak Values are generated from precision and range.

Cursor Focus is more modern than Motif.

More Button leads to Detail Panel . . .



Motor Details: 13XRM:m3 | theta |

13XRM:m3: (asynMotor)

Label: theta units: degrees

Drive	User	Dial	Raw	
High Limit	1500.0000	1500.0000		Stop
Readback	25.0000	25.0000	50000	Pause
Move	25.0000	25.0000	50000	Move
Low Limit	-1500.0000	-1500.0000	Enable	Go
Tweak	< 0.0100 >		Disable	

Calibration

Mode: Use Set Freeze Offset: Variable

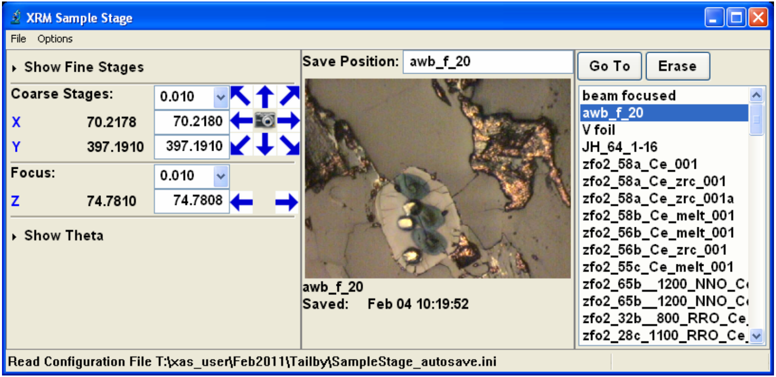
Direction: Pos Neg Offset Value: 0.0000

Dynamics	Normal	Backlash
Max Speed	20.0000	
Speed	20.0000	1.0000
Base Speed	1.0000	
Accel (s)	0.5000	0.2000
Backslash Distance		0.0000
Move Fraction		1.0000

Resolution

Motor Res	0.0005	Encoder Res	0.0005
Steps / Rev	2000	Units / Rev	1.0000
Precision	4		

Custom Application: Sample Stage



Save Position: awb_f_20

Coarse Stages: 0.010

X 70.2178 70.2180

Y 397.1910 397.1910

Focus: 0.010

Z 74.7810 74.7808

awb_f_20
Saved: Feb 04 10:19:52

beam focused
awb_f_20
V foil
JH_64_1-16
zfo2_58a_Ce_001
zfo2_58a_Ce_zrc_001
zfo2_58a_Ce_zrc_001a
zfo2_58b_Ce_melt_001
zfo2_56b_Ce_melt_001
zfo2_56b_Ce_zrc_001
zfo2_55c_Ce_melt_001
zfo2_65b__1200_NNO_C
zfo2_65b__1200_NNO_C
zfo2_32b__800_RRO_Ce
zfo2_28c_1100_RRO_Ce

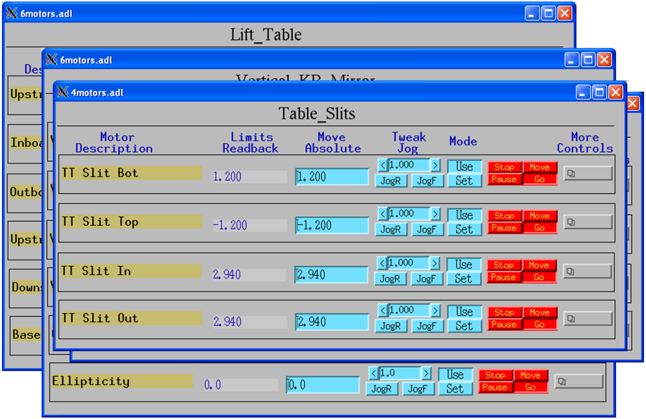
Read Configuration File T:\xas_user\Feb2011\Tailby\SampleStage_autosave.ini

A custom GUI for controlling a six-motor Sample Stage at GSECARS:

- Named Positions Positions can be saved by named and restored.
- Sample Image (JPEG) captured at each saved position.
- Simple Configuration with Windows-style .INI file.

Useful for my station, but definitely application specific.

Basic Epics GUIs are not good enough



Table_Slits

Motor Description	Limits Readback	Move Absolute	Tweak Jog	Mode		More Controls
TT Slit Bot	1.200	1.200	< 1.000 > JogR JogF	Use Set	Stop Move Pause Go	
TT Slit Top	-1.200	-1.200	< 1.000 > JogR JogF	Use Set	Stop Move Pause Go	
TT Slit In	2.940	2.940	< 1.000 > JogR JogF	Use Set	Stop Move Pause Go	
TT Slit Out	2.940	2.940	< 1.000 > JogR JogF	Use Set	Stop Move Pause Go	
Ellipticity	0.0	0.0	< 1.0 > JogR JogF	Use Set	Stop Move Pause Go	

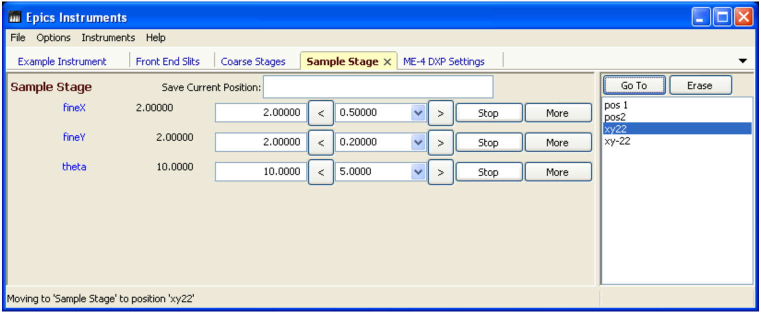
Besides being ugly and hard-to-use, MEDM screens can not save state information, and do not think about multiple PVs as a single item.

Epics Instruments: Saving Positions for Sets of PVs

Epics Instruments is a GUI application that lets any user:

- Organize PVs into *Instruments*: a named collection of PVs
- Manage Instruments with "modern" tab interface.
- Save Positions for any Instrument by name.
- Restore Positions for any Instrument by name.
- Remember Settings all definitions fit into a single file that can be loaded later.

Multiple Users can be using multiple instrument files at any one time.



Sample Stage

Save Current Position:

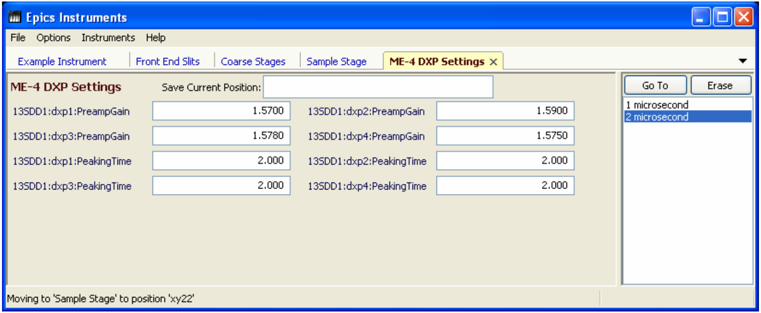
fineX	2.00000	2.00000	<	0.50000	>	Stop	More
fineY	2.00000	2.00000	<	0.20000	>	Stop	More
theta	10.0000	10.0000	<	5.0000	>	Stop	More

pos 1
pos2
xy22
xy-22

Moving to 'Sample Stage' to position 'xy22'

Magic ingredient: SQLite – relational database in a single file.

Epics Instruments: More details



ME-4 DXP Settings

Save Current Position:

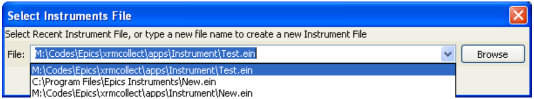
13SDD1:dxp1:PreampGain	1.5700	13SDD1:dxp2:PreampGain	1.5900
13SDD1:dxp3:PreampGain	1.5780	13SDD1:dxp4:PreampGain	1.5750
13SDD1:dxp1:PeakingTime	2.000	13SDD1:dxp2:PeakingTime	2.000
13SDD1:dxp3:PeakingTime	2.000	13SDD1:dxp4:PeakingTime	2.000

1 microsecond
2 microsecond

Moving to 'Sample Stage' to position 'xy22'

Save / restore settings can also include regular (non-motor) PVs.

Typing a name in the box will save the current position, and add it to the list of positions.



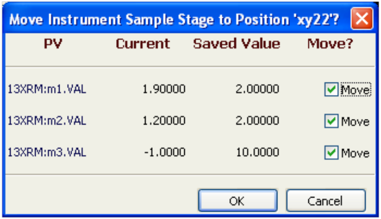
Select Recent Instrument File, or type a new file name to create a new Instrument File

File: M:\Codes\Epics\xrmcollect\apps\Instrument\Test.ein

M:\Codes\Epics\xrmcollect\apps\Instrument\Test.ein
C:\Program Files\Epics Instruments\New.ein
M:\Codes\Epics\xrmcollect\apps\Instrument\New.ein

At startup, any recently used database files can be selected.

Epics Instruments: A few more screenshots



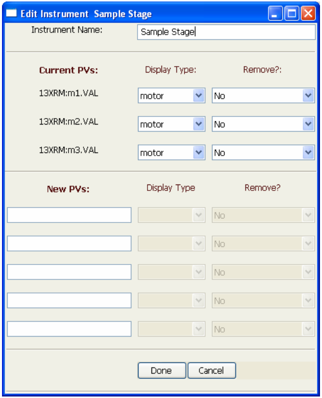
Move Instrument Sample Stage to Position 'xy22'?

PV	Current	Saved Value	Move?
13XRM:m1.VAL	1.90000	2.00000	☑ Move
13XRM:m2.VAL	1.20000	2.00000	☑ Move
13XRM:m3.VAL	-1.0000	10.0000	☑ Move

On “Go To”, settings can be compared with current values, and selectively restored.

Server Mode: An application can listen to a simple Epics Record.

This allows other processes (IDL, Spec, . . .) to restore instruments to named positions by writing to a couple PVs.



Edit Instrument Sample Stage

Instrument Name: Sample Stage

Current PVs:	Display Type:	Remove?:
13XRM:m1.VAL	motor	No
13XRM:m2.VAL	motor	No
13XRM:m3.VAL	motor	No

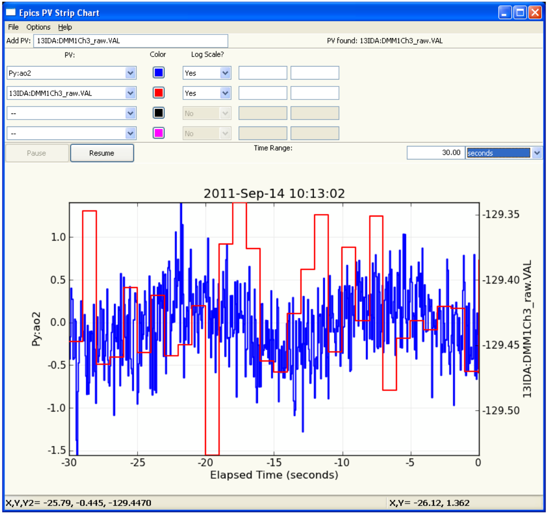
New PVs:	Display Type	Remove?
		No
		No
		No
		No
		No

Edit screen to set PVs that make up and Instrument.

Suggestions Welcome!

http://github.com/pyepics/epicsapps

PV StripChart Application



Add PV: 13IDA:DMM1Ch3_raw.VAL

PV found: 13IDA:DMM1Ch3_raw.VAL

PV:	Color	Log Scale?		
Py:ao2	🟦	Yes		
13IDA:DMM1Ch3_raw.VAL	🟥	Yes		
--	⬛	No		
--	🟪	No		

Time Range: 30.00 seconds

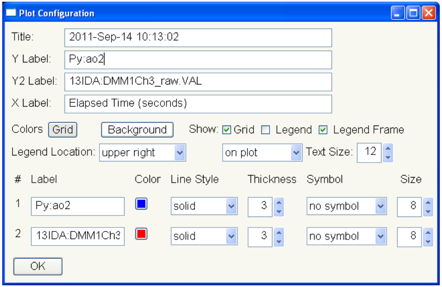
X,Y,Y2= -25.79, -0.445, -129.4470

X,Y= -26.12, 1.362

Live Plots of Time-Dependent PVs:

- Interactive Graphics, Zooming.
- Set Time Range, Y-range
- Set log plot
- Save Plot to PNG.
- Data can be saved to ASCII files.

More PV StripChart Views



Title: 2011-Sep-14 10:13:02

Y Label: Py:ao2

Y2 Label: 13IDA:DMM1Ch3_raw.VAL

X Label: Elapsed Time (seconds)

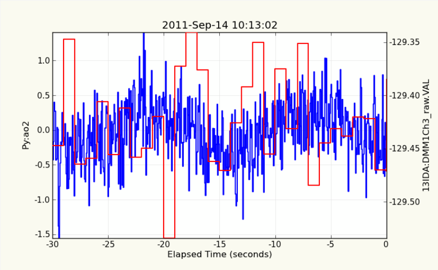
Colors: Grid Background Show: ☑ Grid ☐ Legend ☑ Legend Frame

Legend Location: upper right on plot Text Size: 12

#	Label	Color	Line Style	Thickness	Symbol	Size
1	Py:ao2	🟦	solid	3	no symbol	8
2	13IDA:DMM1Ch3	🟥	solid	3	no symbol	8

Simple plot configuration:

- trace colors, symbols, line widths.
- titles, axes labels (LaTeX for math/Greek characters!)
- chart legend



High Quality output PNG:

Ctrl-C for Copy-to-Clipboard

Ctrl-P to Print

Windows, Mac OS X, Linux.

http://github.com/pyepics/epicsapps

PyEpics: Epics Channel Access for Python

- near complete interface to CA, threads, preemptive callbacks.
- tested: linux-x86, linux-x86_64, darwin-x86, win32-x86 (base 3.14.12.1) with Python 2.5, 2.6, 2.7, 3.1.
- documented and some unit-testing (~70% coverage of core).
- easy installation and deployment.
- high-level PV class, Devices.
- GUI support (wxPython only so far).
- some general-purpose applications begun.
- `http://github.com/pyepics`

Acknowledgments: co-developer: Angus Gratton, ANU.

Suggestions, bug reports and fixes from Michael Abbott, Marco Cammarata, Craig Haskins, Pete Jemian, Andrew Johnson, Janko Kolar, Irina Kosheleva, Tim Mooney, Eric Norum, Mark Rivers, Friedrich Schotte, Mark Vigder, Steve Wasserman, and Glen Wright.