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:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clean Syntax</td>
<td>Easy to learn, remember, and read</td>
</tr>
<tr>
<td>High Level</td>
<td>No pointers, dynamic memory, automatic memory</td>
</tr>
<tr>
<td>Cross Platform</td>
<td>Code portable to Unix, Windows, Mac.</td>
</tr>
<tr>
<td>Object Oriented</td>
<td>Full object model, name spaces.</td>
</tr>
<tr>
<td>Easily Extensible</td>
<td>With C, C++, Fortran, ...</td>
</tr>
<tr>
<td>Many Libraries</td>
<td>GUIs, Databases, Web, Image Processing, ...</td>
</tr>
</tbody>
</table>

### 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 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()**

```python
>>> 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:

```python
>>> 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:

```python
>>> 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**

```python
>>> 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
'...     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.'
```

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

put() and connection callbacks have similar signatures.
Epics Waveform array data is very important for experimental data:

### double waveform

```python
>>> 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.]
```

### character waveform

```python
>>> 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')
```

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

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.
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.

```python
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.
Wrapping CA with ctypes:

```python
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.
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.
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**

```python
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**

```python
>>> 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.
And, of course, a *Device* can have methods added:

```python
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.

**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

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

```python
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

Main features:

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

PV values displayed as html links to Plot of Data
Epics Archiver: Plotting Historical Data

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

Web-based Data Archiver
M Newville  CARS, Univ Chicago  15-Sep-2011
Many PV types (Double, Float, String, Enum) have wxPython widgets, which automatically tie to the PV.

Sample wx widget Code

```python
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.
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
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 . . .
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 GUls are not good enough

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

Epics Instruments: SQLite

M Newville CARS, Univ Chicago

15-Sep-2011
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.

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.

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

Suggestions Welcome!

http://github.com/pyepics/epicsapps
PV StripChart Application

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

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
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.