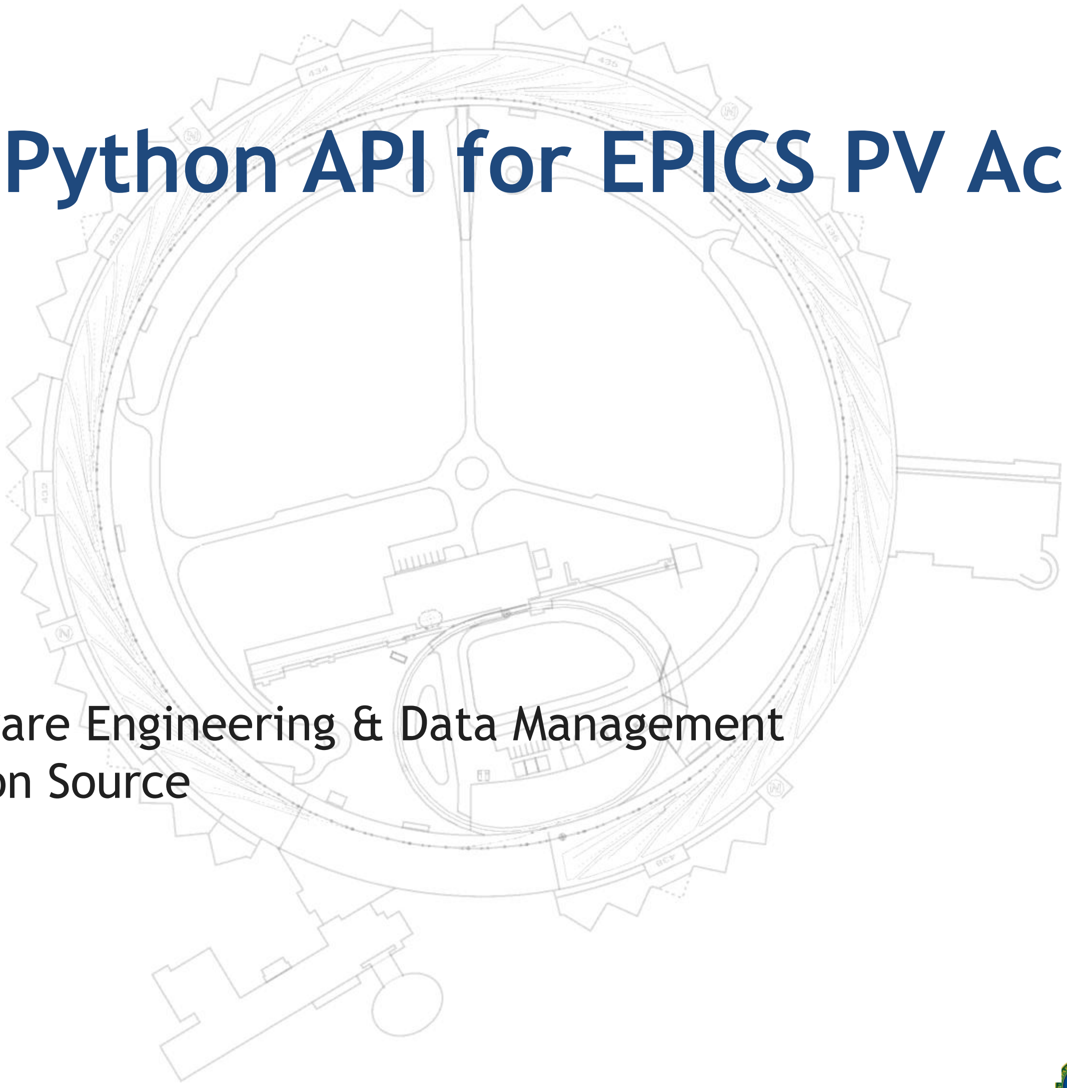
PvaPy: Python API for EPICS PV Access



Siniša Veseli
Scientific Software Engineering & Data Management
Advanced Photon Source

EPICS Meeting
October 2015

About PvaPy

- Python API for PV Access
- Hosted on GitHub: https://github.com/epics-base/pvaPy
- Part of the v4 release: http://sourceforge.net/projects/epics-pvdata/files
- Simple to build and use: one should be able to get started in minutes
- Uses Boost.Python framework to wrap PV Access C++ libraries:
  - Enables one to leverage existing functionality and reduce implementation effort
  - Simplifies maintenance: future improvements in C++ infrastructure should benefit python PVA API
- Python look and feel: easy conversion between python objects (dictionaries, lists, etc.) and PV structures

About PvaPy

- Features
  - Standard EPICS build, enhanced with automated configuration
  - Support for all PV data types (scalars, structures, unions)
  - Support for setting and retrieving channel values
  - Channel monitoring support
  - RPC Client/Service support
  - Initial NT object support
  - Standard Python module documentation
- Goal: provide full PV Access functionality, anything that can be done via C++ APIs should also be doable with PvaPy

Build

1) Configure build.

```
$ make configure EPICS_BASE=<epics_base>
   EPICS4_DIR=<epics4_dir>
```

Automated configuration generates `configure/RELEASE.local` and `configure/CONFIG_SITE.local` files. It also creates environment setup files.

2) Compile sources.

```
$ make
```

Build process creates and installs a loadable library named `pvaccess.so` under the `lib/python` directory which can be imported directly by Python.

Basic Usage

- Before using PvaPy, either source setup file, or modify $PYTHONPATH manually
- Setup file (bash): `source $PVAPY_DIR/bin/$EPICS_HOST_ARCH/setup.sh`
- Manual setup (bash): `export PYTHONPATH=$PVAPY_DIR/lib/python/$PYTHON_VERSION/$EPICS_HOST_ARCH:$PYTHONPATH`
- Python module is called “pvaccess”

```
 $ python -c "import pvaccess; print dir(pvaccess)”
```

PvObject Class

- Base class for all python PVA objects is *PvObject* (a generic PV structure)
- It is initialized with a dictionary of introspection data: key is the field name string, value is one of:
  - PVTYPE: a scalar type, any of BOOLEAN, BYTE, UBYTE, SHORT, USHORT, INT, UINT, LONG, ULONG, FLOAT, DOUBLE, or STRING
  - [PVTYPE]: a single element list, representing a scalar array
  - {key:value,…}: a dictionary, representing a structure
  - [{key:value,…}]: a single element list containing a dictionary, representing a structure array
  - (): an empty tuple, representing variant union
  - [()]: a single element list containing an empty tuple, representing variant union array
  - ({key:value,…},): a single element tuple holding a dictionary, representing a restricted union
  - [({key:value,…},)]: a single element list containing a single element tuple of a dictionary, representing a restricted union array

PvObject: Simple Structure Example

```
>>> pv = PvObject({'i' : INT, 's' : STRING})
>>> print pv
structure
    int i 0
    string s

>>> # Can set entire object with key/value dictionary
>>> pv.set({'i' : 12, 's' : 'abcd'})
>>> print pv
structure
    int i 12
    string s abcd

>>> # Can use getters/setters for each field
>>> pv.getString('s')
'abcd'
>>> pv.setString('s', 'xyz')
>>> pv.getString('s')
'xyz'
```

PvObject: Complex Structure Example

```
>>> pv = PvObject({'i': INT, 'slist' : [STRING], 'dict' : {'b' :
BOOLEAN, 'dict2' : {'d' : DOUBLE}, 'flist' : [FLOAT]}})
>>> print pv
structure
    int i 0
    string[] slist []
    structure dict
        boolean b 0
        float[] flist []
        structure dict2
            double d 0
>>> # Can use incomplete dictionaries to set fields
>>> pv.set({'i' : 15, 'dict' : {'flist' : [1.1, 2.2, 3.3]}})
>>> print pv
structure
    int i 15
    string[] slist []
    structure dict
        boolean b 0
        float[] flist [1.1,2.2,3.3]
        structure dict2
            double d 0
```

PvObject: Conversion to Dictionary

```
>>> # Conversion to dictionary: use either get() or toDict()
>>> pv.get()
{'i': 15, 'slist': [], 'dict': {'b': False, 'dict2': {'d':
0.0}, 'flist': [1.100000023841858, 2.200000047683716,
3.299999952316284]}}

>>> # Get structure field
>>> pv.getStructure('dict')
{'b': False, 'dict2': {'d': 0.0}, 'flist':
[1.100000023841858, 2.200000047683716, 3.299999952316284]}

>>> # Get introspection dictionary
>>> pv.getStructureDict()
{'i': pvaccess.PvType.INT, 'slist':
[pvaccess.PvType.STRING], 'dict': {'b':
pvaccess.PvType.BOOLEAN, 'dict2': {'d':
pvaccess.PvType.DOUBLE}, 'flist': [pvaccess.PvType.FLOAT]}}
```

PvObject: Union Support

```
>>> # Union support
>>> pv = PvObject({'v' : (), 'u' : ({'i': INT, 'd' :
DOUBLE},)})
>>> print pv
structure
    union u
        (none)
    any v
        (none)

>>> # Set variant union
>>> s = PvObject({'s' : STRING})
>>> s.setString('xyz')
>>> pv.setUnion('v', s)
>>> print pv
structure
    union u
        (none)
    any v
        string s xyz
```

PvObject: Union Support

```
>>> # Select restricted union field
>>> u = pv.selectUnionField('u', 'i')

>>> pv.getSelectedUnionFieldName('u')
'i'

>>> # Set restricted union field
>>> u.setInt(3)
>>> print u
structure
    int i 3
>>> print pv
structure
    union u
        int i 3
    any v
        string s xyz
```

Channel Class

- Provides interface for communicating with PV Access channels
- Support for channel monitoring
- Support for Channel Access (the EPICS Version 3 protocol).
- Channel’s “get()” method returns a PvObject representing the current value for the given process variable
- Channel’s “put()” method accepts either a PvObject, or a standard Python data type as input for setting the process variable

Channel Class Example

```
>>> # In addition to PvObjects, we allow standard
>>> # python types to be used for channel puts
>>> c = Channel('bigstring01’)
>>> c.put('My String')
>>> print c.get()
epics:nt/NTScalar:1.0
    string value My String

>>> c = Channel('intArray01’)
>>> c.put([1,2,3,4,5])
>>> print c.get()
structure
    int[] value [1,2,3,4,5]
```

Channel Monitor Example

- Define function to be called when PV value changes, subscribe to the channel, and start monitor

```
>>> def sumMonitor(pv):
...     s = 0
...     for i in pv.get()['value']:
...         s += i
...     print s
>>> c = Channel('intArray01')
>>> c.subscribe('sum', sumMonitor)
>>> c.startMonitor()
```

RPC Server

- *RpcServer* class is used for hosting one or more PVA Remote Procedure Call (RPC) services
- Users define an RPC processing function and register it with an RpcServer instance
- The RPC processing function takes a client’s request PvObject as input, and returns a PvObject that contains the processing result

```
>>> def sum(pvRequest):
>>>     a = pvRequest.getInt('a')
>>>     b = pvRequest.getInt('b')
>>>     return PvInt(a+b)
>>> srv = RpcServer()
>>> srv.registerService('sum',sum)
>>> srv.listen()
```

RPC Client

- *RpcClient* is a client class for PVA RPC services
- Users initialize an RpcClient object giving the service’s channel name, prepare a PV request object, and then invoke the service

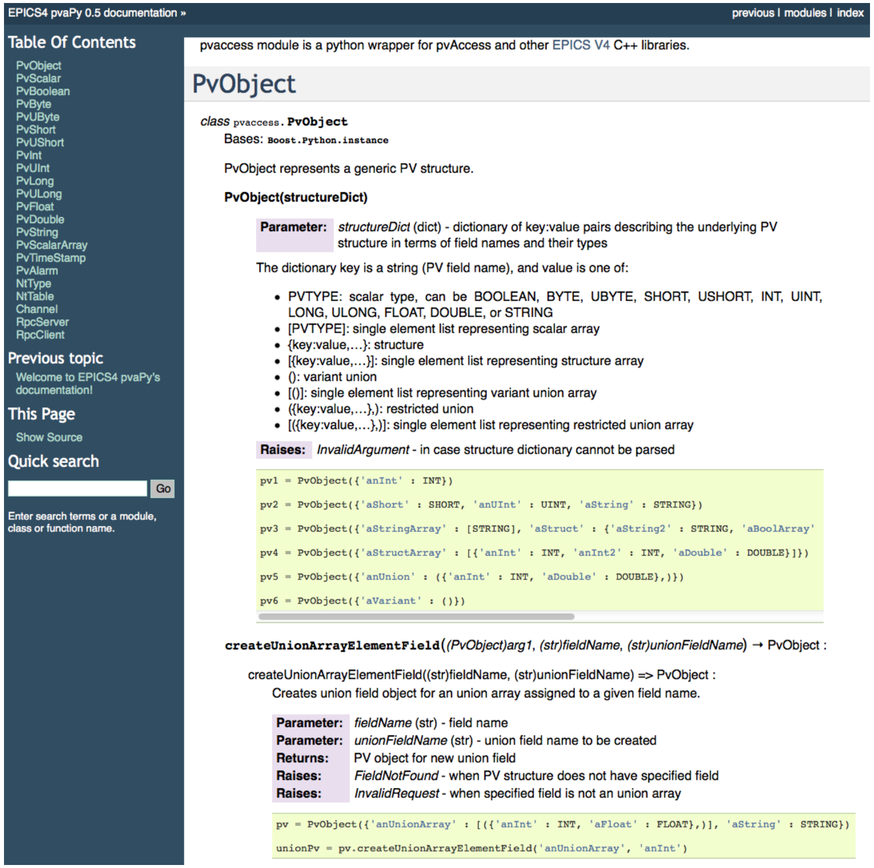
```
>>> c = RpcClient('sum')
>>> request = PvObject({'a':INT,'b':INT})
>>> request.set({'a':1,'b':2})
>>> sum = c.invoke(request)
```

Documentation

- Documentation generated during automated builds: http://epics-pvdata.sourceforge.net/docbuild/pvaPy/tip/pvaccess.html
- Generating HTML docs at build time:

  ```
  $ make doc
  ```

- PvaPy uses Sphinx framework



pvaccess module is a python wrapper for pvAccess and other EPICS V4 C++ libraries.

PvObject

class pvaccess.**PvObject**

Bases: **Boost.Python.instance**

PvObject represents a generic PV structure.

PvObject(structureDict)

Parameter: *structureDict* (dict) - dictionary of key:value pairs describing the underlying PV structure in terms of field names and their types

The dictionary key is a string (PV field name), and value is one of:

- PVTYPE: scalar type, can be BOOLEAN, BYTE, UBYTE, SHORT, USHORT, INT, UINT, LONG, ULONG, FLOAT, DOUBLE, or STRING
- [PVTYPE]: single element list representing scalar array
- {key:value,…}: structure
- [{key:value,…}]: single element list representing structure array
- (): variant union
- [()]: single element list representing variant union array
- ({key:value,…},): restricted union
- [({key:value,…},)]: single element list representing restricted union array

Raises: *InvalidArgument* - in case structure dictionary cannot be parsed

```
pv1 = PvObject({'anInt' : INT})

pv2 = PvObject({'aShort' : SHORT, 'anUInt' : UINT, 'aString' : STRING})

pv3 = PvObject({'aStringArray' : [STRING], 'aStruct' : {'aString2' : STRING, 'aBoolArray'

pv4 = PvObject({'aStructArray' : [{'anInt' : INT, 'anInt2' : INT, 'aDouble' : DOUBLE}]})

pv5 = PvObject({'anUnion' : ({'anInt' : INT, 'aDouble' : DOUBLE},)})

pv6 = PvObject({'aVariant' : ()})
```

createUnionArrayElementField(*(PvObject)arg1*, *(str)fieldName*, *(str)unionFieldName*) → PvObject :

createUnionArrayElementField((str)fieldName, (str)unionFieldName) => PvObject :

Creates union field object for an union array assigned to a given field name.

Parameter: *fieldName* (str) - field name
Parameter: *unionFieldName* (str) - union field name to be created
Returns: PV object for new union field
Raises: *FieldNotFound* - when PV structure does not have specified field
Raises: *InvalidRequest* - when specified field is not an union array

```
pv = PvObject({'anUnionArray' : [({'anInt' : INT, 'aFloat' : FLOAT},)], 'aString' : STRING})

unionPv = pv.createUnionArrayElementField('anUnionArray', 'anInt')
```

Future Work

- Complete support for all Normative Types
- Support for "putGet()" and "getPut()" operations
- Support for Python 3
- Support for NumPy arrays
- Channel monitor enhancements
- Test suite development
- PVA Server implementation

Summary

PvaPy is the EPICS4 Python API for PV Access.

Its interfaces have been designed with the end user in mind: to be as simple, flexible and intuitive as possible, while still retaining all capabilities and features provided by the PVA protocol.

Give it a try, all comments and suggestions are welcome!

ICALEPCS Poster Session: WEPGF116, 21 Oct 2015, 17:15-18:15

Acknowledgements

- A.N. Johnson worked on ensuring that PvaPy's build conforms to EPICS standards
- M. Kraimer and M. Davidsaver worked on prototyping support for PV unions
- M. Kraimer developed pvaClientCPP package
- K. Vodopivec provided early feedback and suggestions
- R. Lange and D. Hickin worked on automated builds and preparing software releases
- N.D. Arnold and the entire EPICS 4 working group provided support and encouragements for PvaPy development

Additional Slides

Derived Object Classes

- Each scalar type has its own class: `PvBoolean, PvByte, …, PvString`
- All scalar classes can be initialized using scalar value, and have setters/getters

```
>>> s = PvString('abc')
>>> print s
abc
>>> d = PvDouble(123.456)
>>> print d
123.456
>>> l = PvLong(123456789012345678L)
>>> print l
123456789012345678
>>> l.get()
123456789012345678L
>>> l.set(13L)
>>> l.get()
13L
```

Derived Object Classes

- Scalar array type class: `PvScalarArray`
- It is initialized using scalar type, has setter/getter

```
>>> array = PvScalarArray(INT)
>>> print array
structure
    int[] value []
>>> array.set([1,2,3,4,5])
>>> print array
structure
    int[] value [1,2,3,4,5]
```

NT Table Example

- Initialize table with number of columns and column type

```
>>> from pvaccess import *
>>> ntTable = NtTable(3, DOUBLE)
>>> ntTable.setLabels(['Col1', 'Col2', 'Col3'])
>>> ntTable.setColumn(0, [0.1, 1.1, 2.2])
>>> ntTable.setColumn(1, [1.1, 2.2, 3.3])
>>> ntTable.setColumn(2, [2.1, 3.3, 4.4])
```

- Initialize table with list of column types

```
>>> ntTable = NtTable([STRING, INT, DOUBLE])
>>> ntTable.setLabels(['String', 'Int', 'Double'])
>>> ntTable.setColumn(0, ['row0', 'row1', 'row2'])
>>> ntTable.setColumn(1, [1, 2, 3])
>>> ntTable.setColumn(2, [2.1, 3.3, 4.4])

>>> ntTable.setDescriptor("Nice Table, Bad Results")
>>> timeStamp = PvTimeStamp(12345678L, 12)
>>> ntTable.setTimeStamp(timeStamp)
>>> alarm = PvAlarm(11, 126, "Server SegFault")
>>> ntTable.setAlarm(alarm)
```