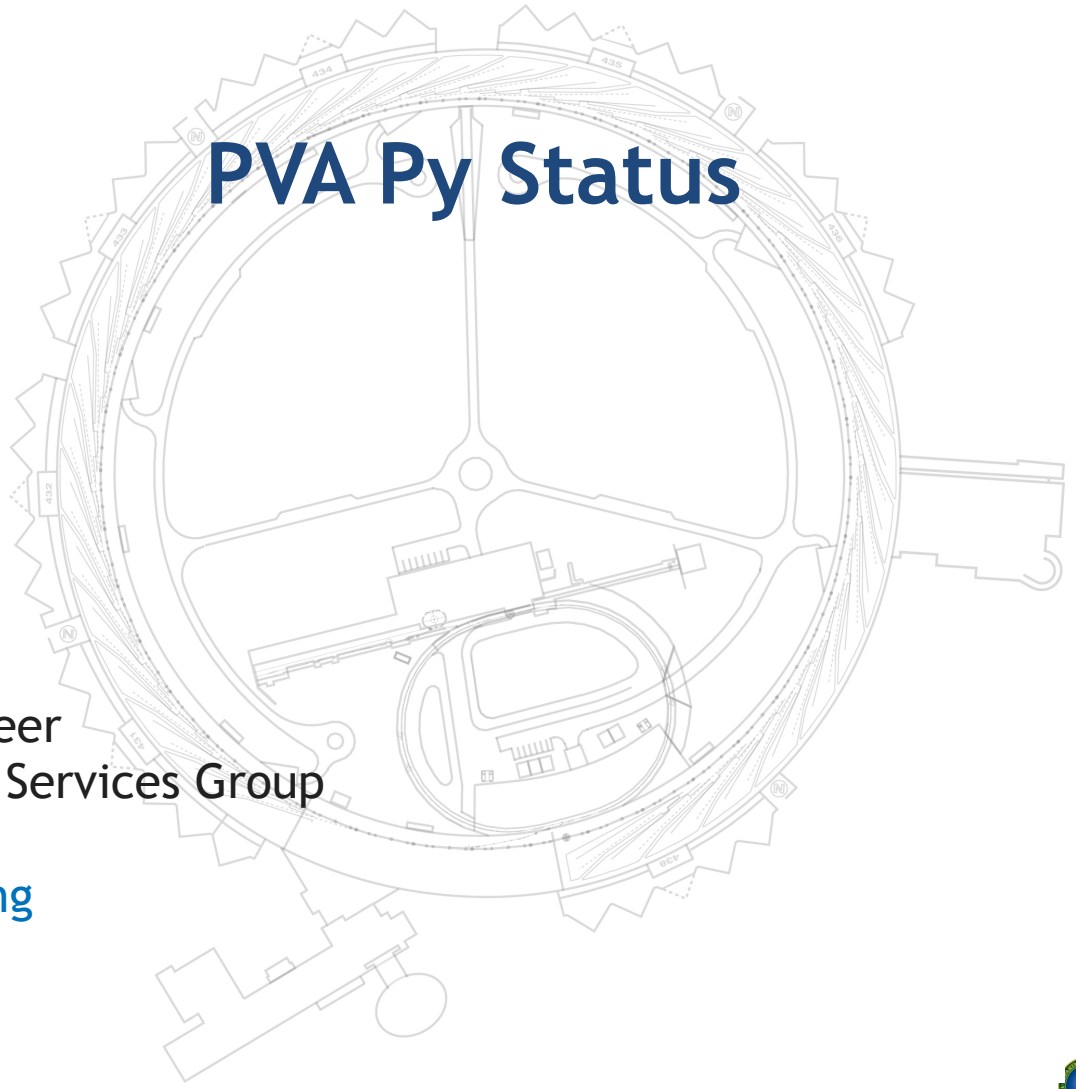
PVA Py Status



Siniša Veseli
Software Engineer
AES / Software Services Group

EPICS v4 Meeting
July 2014

About PVA Py

- Python API for PV Access:
  - Simple to build and use: one should be able to get started in minutes
  - Has potential to provide full PV Access functionality: anything that can be done via C++ APIs should be doable with PVA Py
  - Python look and feel: easy conversion to and from python objects (dictionaries, lists, etc.)
- Uses boost.python to wrap PV Access C++ libraries:
  - Enables one to leveraging existing functionality and reduce implementation effort
  - Simplifies maintenance: future improvements in C++ infrastructure should benefit python PVA API
  - Problem: No well-defined high level C++ API
- Current functionality: support for scalars and structures, pvput/pvget, monitor support, RPC server/client, initial NT object support

Recent Developments (6 months)

- Automated build configuration
  - Uses autoconf scripts to generate required build and user setup files
  - Can handle multiple versions of underlying PVA C++ libraries
- Added PV monitor functionality
- Added ability to specify arbitrary request descriptors for get/ put
- Framework for adding NT object support: currently only NtTable and related objects are implemented, but can easily add more as need arises
- Added interactive mode for RPC Server

PV Access in PVA Py

- Base for all python PVA objects is `PvObject`
  - Has setters/getters for all field types, e.g.
    - `setInt(key, value)`
    - `getInt(key)`
    - `setScalarArray(key, value)`
    - `getScalarArray(key)`
    - `setStructure(key, value)`
    - `getStructure(key)`
- PV structure is initialized via python dictionary of key:value pairs that describe underlying PV structure: key is a string and value one of PVTYPE, [PVTYPE], {key:value,…}, [{key:value,…}]
- PVTYPE: `BOOLEAN, BYTE, UBYTE, SHORT,…,STRING`
- Scalar Array: represented as a list with a single PVTYPE element describing element type, e.g. `[INT]`
- Structure Array: represented as list a with a single dictionary element describing element structure, e.g. `[{'x' : INT, 'y' : FLOAT}]`

Basic Usage

- Source setup file (or export PYTHONPATH=$PVAPY_DIR/lib/$EPICS_HOST_ARCH:$PYTHONPATH)
- Inspect package contents:

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

- Start v4 test IOC
- Use Channel class to get/set PVs:

```
$ python
>>> from pvaccess import *    # never do that in scripts
>>> c = Channel('int01')
>>> print c.get()
uri:ev4:nt/2012/pwd:NTScalar
  int value 0
>>> c.put(PvInt(7))
>>> print c.get()
uri:ev4:nt/2012/pwd:NTScalar
    int value 7
```

High-Level C++ API Issue

- Lots of work for PVA Py went into defining C++ API that would be easy to wrap using boost.python
- Fair amount PVA Py C++ code is duplicated from utilities like pvget and pvput
- Common code (e.g., various default requester impl classes, parsing utilities, etc.) could be extracted into high level PVA C++ API that would be easier to use and more attractive for an average user (RPC Service/Client C++ classes are an excellent example)
- All PVA command line/test utilities could be built on top such API (e.g., `src/pvaccess/testClient.cpp` retrieves PV from a given channel in about 20 lines of code)
- Promotes code reusability, easier maintenance, etc.

High-Level C++ API Issue

- For example, see src/pvaccess/testClient.cpp:

```
int main(int argc, char** argv)
{
    if(argc < 2) {
        std::cout << "Usage: " << argv[0]
                << " <channelName>" << std::endl;
        return 1;
    }
    try {
        Channel channel(argv[1]);
        std::cout << *(channel.get()) << std::endl;
    }
    catch (PvaException& ex) {
        std::cerr << ex.what() << std::endl;
    }
}
```

Work In Progress

- Code updates needed due to recent C++ library changes
- Work on user documentation and better usage examples
  - Will be using Sphinx to generate python docs at build time
- Anything else needed before the next v4 release?

Future Plans

- Usability Enhancements
  - Support for channel access
  - Additional object constructors
  - Python operators for scalar types
  - Support for more NT types
  - Support for unions
  - …
- Any requests/immediate needs?
- Sorry for the lack of demo…

Additional Slides

PvObject Examples

```
>>> 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 Examples

```
>>> 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 Examples

```
>>> # Conversion to dictionary
>>> pv.toDict()
{'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 original structure 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]}}
```

Derived Object Classes

- Each scalar type has its own class: `PvBoolean, PvByte, …, PvString`
  - Can be initialized using scalar value
  - Have setters/getters

```
>>> s = PvString('abc')
>>> print s
abc
>>> d = PvDouble(123.456)
>>> print d
123.456
>>> l = PvLong(12345678901234567L)
>>> print l
12345678901234567
>>> l.get()
12345678901234567L
>>> l.set(13L)
>>> l.get()
13L
```

Derived Object Classes

- Scalar array type class: `PvScalarArray`
  - Initialized using scalar type
  - 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]
```

Channel Class

- Provides get/put functionality

```
>>> c = Channel('bigstring01')
>>> c.put(PvString('Very Big String'))
>>> print c.get()
uri:ev4:nt/2012/pwd:NTScalar
    string value Very Big String
c = Channel('intArray01')
>>> print c.get()
structure
    int[] value []
>>> print array
structure
    int[] value [1,2,3,4,5]
>>> c.put(array)
>>> print c.get()
structure
    int[] value [1,2,3,4,5]
```

RPC Client Class

- Client for RPC service
- Start v4 test RPC service from $EPICS4_DIR/pvAccessCPP/bin/linux-x86_64/

```
$ ./rpcServiceExample # in terminal 2
```

- RPC test channel is “sum”:

```
>>> rpc = RpcClient('sum')
>>> request = PvObject({'a' : STRING, 'b' : STRING})
>>> request.set({'a' : '11', 'b' : '22' })
>>> print request
structure
    string a 11
    string b 22
>>> response = rpc.invoke(request)
>>> print response
structure
    double c 33
```

RPC Server Class

- Allows creating PVA services in python
- In terminal 1

```
$ python # in terminal 2
>>> from pvaccess import *
>>> srv = RpcServer()
>>> def echo(x):   # x is instance of PvObject
...     print 'Got object: ', x
...     return x   # service must return instance of PvObject
>>> srv.registerService('echo', echo)
>>> srv.listen()
```

- In terminal 1

```
>>> rpc = RpcClient('echo')
>>> response = rpc.invoke(request)
>>> print response
structure
    string a 11
    string b 22
```

RPC Client/Server Example

- In terminal 2

```
$ python
>>> from pvaccess import *
>>> srv = RpcServer()
>>> def sum(x):
...     a = x.getInt('a')
...     b = x.getInt('b')
...     return PvInt(a+b)
>>> srv.registerService('sum', sum)
>>> srv.listen()
```

- In terminal 1

```
>>> rpc = RpcClient('sum')
>>> request = PvObject({'a' : INT, 'b' : INT})
>>> request.set({'a' : 11, 'b' : 22})
>>> print request
structure
    int a 11
    int b 22
>>> response = rpc.invoke(request)
>>> print response
structure
    int value 33
```

RPC Client/Server Example

- In terminal 2

```
>>> from pvaccess import *
>>> srv = RpcServer()
>>> def hash(x):
...     import hashlib
...     md5 = hashlib.md5()
...     md5.update(str(x))
...     h = md5.hexdigest()
...     dict = x.getStructureDict()
...     dict['hash'] = STRING
...     response = PvObject(dict)
...     response.setString('hash', h)
...     return response
>>> srv.registerService('hash', hash)
>>> srv.listen()
```

- In terminal 1

```
>>> rpc = RpcClient('hash')
>>> request = PvString('abcd')
>>> print rpc.invoke(request)
structure
    string hash 0a380e7375d8c3f68d1bbe068141d6ce
    string value
```

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)
```

Channel Monitor Example

- Define function to be called when PV value changes and start monitor

```
>>> from pvaccess import *
>>> c = Channel('float03')
>>> def echo(x=125):
>>>     print 'Got value in python: ', x
>>> c.subscribe('echo', echo)
>>> c.startMonitor()
```

- Monitor NT Table

```
>>> def monitor(pvObject):
>>>     ntTable = NtTable(pvObject)
>>>     print "Full NT Table"
>>>     print ntTable
>>>     print "Column 0:"
>>>     print ntTable.getColumn(0)
>>> c = Channel('testTable')
>>> c.subscribe('m1', monitor)
>>> c.startMonitor('field()')
>>> time.sleep(10)
>>> c.unsubscribe('m1')
```