PvaPy: Python API for EPICS PV Access

S. Veseli, Argonne National Laboratory, Argonne, IL 60439, USA

Overview

The PvaPy package provides a Python API for EPICS PV Access. It wraps the EPICS4 C++ libraries using the Boost.Python framework that enables interoperability between C++ and Python. Some of the PvaPy features include:

- Standard EPICS build, enhanced with automated configuration
- Support for all PV data types (scalars, structures, unions)
- Support for setting and retrieving channel values
- Monitoring support
- RPC Client/Service support
- Standard Python module documentation

The PvaPy source code is hosted on GitHub at https://github.com/epics-base/pvaPy and is bundled as part of the EPICS4 releases at http://sourceforge.net/projects/epics-pvdata/files

PvaPy Objects

PvObject class represents a generic PV structure. It is initialized with a dictionary of introspection data that describes the underlying structure in terms of field names (keys) and their types (values). All PV data types can be represented using standard Python types and data structures (dictionaries, lists, tuples).

Example 1: Initializing a PvObject from a structure array and a restricted union.

```
pv = PvObject({
  'sArray':[{'i':INT,'d':DOUBLE}],
  'u':({'f':FLOAT,'s':STRING},)
})
```

Actual field values for PvObject instances can be set using a dictionary keyed on the field names. The corresponding "get()" method returns a dictionary of all the the PvObject's field values.

Example 2: Setting a PvObject's value from a Python dictionary.

```
pv.set({
  'sArray':[
    {'i':1,'d':1.1},
    {'i':2,'d':2.2}
  ]
})
```

An alternative way of manipulating and accessing a PvObject's fields is to use setters and getters that correspond to different field types.

Example 3: Setting a specified structure array field.

```
pv.setStructureArray(
  'sArray',
  [
    {'i':1,'d':1.1},
    {'i':2,'d':2.2}
  ]
)
```

Figure 1: Documentation generated by Sphinx for the *pvaccess* PvaPy Python module.

Channel Class

The *Channel* class provides the Python interface for communicating with PV Access channels, as well as for their monitoring. In addition to PV Access, this class also supports Channel Access (the EPICS Version 3 protocol).

Channel's "get()" method returns a PvObject representing the current value for the given process variable. The "put()" method accepts either a PvObject or a standard Python data type as input for setting the process variable.

Example 4: Initializing the "doubleArray" Channel object and setting its PV value from a Python list.

```
c = Channel('doubleArray')
c.put([1.0,2.0,3.0])
```

The monitoring functionality allows users to subscribe to PV value changes and process them with a Python function that takes a PvObject as an argument and has no return value.

Example 5: Monitoring Channels.

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

RPC Server and Client

The *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 containing the processed result.

Example 6: A simple RPC service returning the sum of two numbers from the client's request.

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

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.

Example 7: An RPC client for the "sum" service.

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

Future Work

Some features planned for the future:

- 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