EPICS Meeting, SLAC, October 2013

pvAccess Client APIs

Matej Sekoranja, presented by Marty Kraimer

Overview

COSYLAB

- pvAccess is network support for pvData
  - pvData supports structured data
  - Introspection and data API

- Implementations (APIs)
  - pvAccess Client API
  - pvAccess RPC API
  - EasyPVA
  - pvManager API
  - pvManager Service API
  - Python implementation

pvAccess Client API

COSYLAB

- Implemented in both Java and C++
  - This talk describes Java API, C++ API is similar
- Java and C++ both support complete pvAccess network protocol
- Both also support EPICS CA network protocol
  - pvAccess client API, CA over the wire
- Asynchronous Callback API

Channel Interface

COSYLAB

```
public interface Channel extends Requester {
        // ... some methods omitted

 void getField(GetFieldRequester requester, String subField);
 ChannelProcess createChannelProcess(ChannelProcessRequester cb, PVStructure pvReq);
 ChannelGet createChannelGet(ChannelGetRequester cb, PVStructure pvRequest);
 ChannelPut createChannelPut(ChannelPutRequester cb, PVStructure pvRequest);
 ChannelPutGet createChannelPutGet(ChannelPutGetRequester cb, PVStructure pvRequest);
 ChannelRPC createChannelRPC(ChannelRPCRequester cb, PVStructure pvRequest);
 Monitor createMonitor(MonitorRequester cb, PVStructure pvRequest);
 ChannelArray createChannelArray(ChannelArrayRequester cb, PVStructure pvRequest);
}
```

GetField – Get introspection info for channel.
ChannelProcess – Ask channel to process. No data is transferred.
ChannelGet – Get data from channel.
ChannelPut – Put data to channel.
ChannelPutGet – Put data to channel and get result. Thus like a Remote Procedure Call.
ChannelRPC – An RPC where different types of data sent and received for each request.
ChannelArray – put and get sub array.
Monitor – Monitor data changes.

Create channel

COSYLAB

```
// register pluggable channel providers
org.epics.pvaccess.ClientFactory.register();
org.epics.caV3.ClientFactory.register();


// get a pvAccess client provider
ChannelProvider channelProvider =
    ChannelAccessFactory.getChannelAccess()
       .getProvider("pva");


// create a channel
channelProvider.createChannel(
                    "ai001",
                    new ChannelRequesterImpl(),
                    ChannelProvider.PRIORITY_DEFAULT
                    );
```

Create Channel Callback

```
public interface ChannelRequester extends Requester
{
    void channelCreated(Status status, Channel channel);

    void channelStateChange(Channel channel,
                            ConnectionState connectionState);
}
```

Create ChannelGet and get

```
channel.createChannelGet(
        new ChannelGetRequesterImpl(),
        CreateRequestFactory.createRequest(
                "field(value,timeStamp)"
        )
);
```

```
// once you get channelGet in callback, invoke get
channelGet.get(false);
```

Create ChannelGet Callbacks

COSYLAB

```
public interface ChannelGetRequester extends Requester
{
    void channelGetConnect(
              Status status,
              ChannelGet channelGet,
              PVStructure pvStructure,
              BitSet bitSet);

    void getDone(Status status);
}


public interface ChannelGet extends ChannelRequest
{
    void get(boolean lastRequest);
}
```

EasyPVA

COSYLAB

- Layer on-top of pvAccess Client API
- Simplified synchronous API
- Java implementation not complete but usable
  - get, put, RPC implemented
- C++ version not implemented
- Can also be used from MatLab

```
// get the scalar value
double value = easyPVA.createChannel("ai001")
                      .createGet().getDouble();
```

EasyPVA

COSYLAB

```
// get the scalar value (with timestamp, alarm) multiple times
EasyGet easyGet = easyPVA.createChannel("ai001").createGet();

double value = easyGet.getDouble();
Alarm alarm = easyGet.getAlarm();
TimeStamp timeStamp = easyGet.getTimeStamp();

// ... later ...
value = easyGet.getDouble();
```

- There are plans for even more easier API, aka SuperEasyPVA

pvAccess RPC API

COSYLAB

- Layer on-top of pvAccess Client API that makes writing RPC client (and services) very easy
- Implemented in both Java and C++
  - This talk describes Java API, C++ API is similar
- Provides both - asynchronous and synchronous API

```
//
// sync example, timeout of 3.0s
//
RPCClientImpl client = new RPCClientImpl("myService");
PVStructure result = client.request(arguments, 3.0);
```

pvManager API

COSYLAB

- pvAccess plugin for pvManager implemented
- pvManager API can be used to talk pvAccess

```
// install pvAccess data source
PVManager.setDefaultDataSource(new PVADataSource());

// create a monitor with max 10Hz reate
PVReader<VInt> reader =
  PVManager.read(channel("testCounter", VInt.class, VInt.class)).
        readListener(new PVReaderListener<Object>() {

            @Override
            public void pvChanged(PVReaderEvent<VInt> event) {
                if (event.isValueChanged())
                   System.out.println(event.getPvReader().getValue());
                else
                   System.out.println(event.toString());
            }
        }).maxRate(TimeDuration.ofHertz(10));
```

pvManager Service API

COSYLAB

- pvAccess RPC plugin for pvManager Service API is being implemented
- pvManager Service API can be used to talk pvAccess RPC
- Services are defined using XML, no programming needed
- This allows you to connect services within CSS (GUIs, tables, etc.)

```
ServiceRegistry.getDefault().registerService(new PVAService());

// find method
ServiceMethod method = ServiceRegistry.getDefault()
                        .findServiceMethod("orbitService/getOrbit”);

// set arguments (if any)...
Map<String, Object> arguments = new HashMap<>();

// invoke RPC call
VTable table = (VTable) syncExecuteMethod(method, arguments).get("result");
```

Python implementation

COSYLAB

- Implementation started…
- Only fragmens of pvData implemented, nothing for pvAccess

THANK YOU!