Introduction to EPICS 7

Andrew Johnson
Controls Group, Accelerator Systems Division
Argonne National Laboratory

Preface: What is EPICS 7

EPICS 7 came from combining EPICS Base (V3) and EPICS V4

- V4 was not a replacement or rewrite of EPICS Base
- V4 did not introduce a new IOC database

- V4 used EPICS Base for its build system and OS-independence
- V4 provided functionality not available in EPICS Base

- I use ‘PVA modules’ to mean ‘the V4 code in EPICS 7’

- Upgrading existing IOCs to EPICS 7 makes many new features available to control system applications

Outline: What’s covered

- What was V4 and why was it developed?
- V4 key concepts
  - pvData and pvAccess
  - pvRequest
  - Normative Types
- IOCs and PVA
- pvTools and Language Bindings
  - C++, Python, Java
- Current Status

What was V4 and why was it developed?

Describe EPICS V4 in six words

EPICS is a set of tools, libraries and applications to create a distributed control system

V4 added structured data to EPICS

What was V4 designed for?

- Fix Channel Access problems
  - Better array and string handling
- Structured data
  - Extending the scope of EPICS to support data acquisition, image processing, and beyond
- Efficient network transfer
  - High performance archiving and image transfer
- RPC type services
  - Service oriented architecture: archiver, snapshot, database backends
- Complex control
  - Communicating with devices (groups of PVs on an IOC) in an always-consistent, transaction type way

V4 fixed a number of problems in V3

- Support for 64bit integers
  - 1/8/16/32/64bit integers, signed and unsigned
- Better support for arrays
  - No element_count upper limit (fixed and bounded arrays possible)
  - Clear distinction between arrays of size 1 and scalars
- Better support for strings
  - Arbitrary size
  - No fixed limit or need for long string workaround
- Much better support for arrays of strings
  - Handles arbitrary number of arbitrary length strings

Structured data

V4 can do everything V3 can do (but better)

- Construct pvData structures analogous to DBR types
- For example the equivalent of a DBR_TIME_DOUBLE is the structure

```
NTScalar
    double value
    alarm_t alarm
        int severity
        int status
        string message
    time_t timeStamp
        long secondsPastEpoch
        int nanoseconds
        int userTag
```

Efficient network transfer

pvAccess operations only send deltas on the wire.
If the value of the structure in the above example is modified to:

```
NTScalar
    double value                          8.1
    alarm_t alarm
        int severity                        2
        int status                          3
        string message             HIHI_ALARM
    time_t timeStamp
        long secondsPastEpoch      1460589145
        int nanoseconds             588698520
        int userTag                         0
```

only the changed values (shown in **bold**) need be sent, plus a bit-set indicating which fields have new values.

RPC type services

RPC type services can use structures that are different for every call and different for put (request) and get (response).

pvData can encode more complex data types, like a table:

```
NTTable
    string[] labels [value, seconds, nanoseconds, status, severity]
    structure value
        double[] value          [       1.1,        1.2,        2.0]
        long[] secondsPastEpoch [1460589140, 1460589141, 1460589142]
        int[] nanoseconds       [ 164235768,  164235245,  164235256]
        int[] severity          [         0,          0,          1]
        int[] status            [         0,          0,          3]
```

Complex control

- Possible to create complex structures representing, for example, a detector, camera driver, file writer or camera plugin
- Can operate on subset of fields for control or monitor whole structure
- With RPC can add “methods” and create distributed objects

V4 key concepts

pvData

- System of memory resident structured data types
  - Scalar fields
    - integer (1/8/16/32/64 bit, signed and unsigned)
    - float (32/64 bit)
    - string
    - enum
  - Variant (any) and regular (tagged) unions
  - Arrays
  - Structured fields (nested structures)
- Separate interfaces for introspection and data
  - Client can analyze structure before accessing data
  - Helper classes: factories for creating introspection and data structures

pvAccess

- V4 communication protocol, defined by pvAccess protocol specification
- Client/server architecture, multiple providers per server
- High performance network protocol
  - Codec based
  - Pluggable transports
  - Pluggable security
- Designed to transport pvData structures
  - Also uses pvData structures for channel configuration requests
- Successor to Channel Access

pvAccess communication flow

- Client connects to channel (top level pvData structure)
- Client creates a request object, specifying the specifics
  - Request types: Process, Put, Get, PutGet, Monitor
  - May use a subset of the structure
  - More options to control processing, blocking
- Both client and server create containers to hold data
- Client executes the request (multiple times)
  - pvAccess transmits only changed parts over the network

pvRequest and pvRequest string

- Options for a pvAccess channel, sent at connection time
- Requests supported depend on the server, e.g.
  - Limit operation to part of a structure
  - Processing options (process, block)
  - Monitoring options: queue size (deadband, server-side filtering)
- Cannot be introspected in the current implementations

Normative Types

- Well-defined standard types to aid interoperability
- Defines standard structures for alarm, timestamps, enumerations
- Generic simpler types for PVs
  - scalar, scalar array
  - enum
  - matrix
- Specific, more complex types for services and applications
  - table
  - array of PVs
  - areaDetector image
  - histogram
  - aggregate

Normative Types - Examples

```
NTScalar :=

structure
    scalar_t    value
    string      descriptor   :opt
    alarm_t     alarm        :opt
    time_t      timeStamp    :opt
    display_t   display      :opt
    control_t   control      :opt
```

- Specification of standard, named type
- Often choices (field types, field names)
- Required and optional fields
- Extra fields can be added
- Italics refer to other predefined types

```
NTAggregate :=

structure
    double    value
    long      N
    double    dispersion       :opt
    double    first            :opt
    time_t    firstTimeStamp   :opt
    double    last             :opt
    time_t    lastTimeStamp    :opt
    double    max              :opt
    double    min              :opt
    string    descriptor       :opt
    alarm_t   alarm            :opt
    time_t    timeStamp        :opt
```

IOCs and PVA

IOC Extensions for PVA

- EPICS 7 IOC now has pluggable server and link APIs
  - IOCs can be built without RSRV (CA Server) support
  - Additional server interfaces can be added to the IOC
  - Additional record link types (JSON) can also be added

- **QSRV** adds a pvAccess server to the IOC database
  - Client applications can use pvAccess to connect to IOC records.
  - IOC ‘pva’ record links can talk to other pvAccess servers.
  - IOC must be built and linked with PVA libraries

- The **pvAccess** client library can communicate over both pvAccess and Channel Access network protocols
  - Clients use the same API for both PVA and CA connections

    This functionality is not being included in reimplementations

QSRV

- pvAccess interfaces for use by IOCs
  - Builds an executable softIocPVA with PVA server and client (PV link) support
- **Single PV** server support with zero configuration
  - Any public record.field PV can be accessed as a standard NT structure
- **Group PV** server support
  - Arbitrary groups of record.fields accessible under a new name as a single structure
  - Configured in the IOC database (.db files) using info tags
  - Data access is atomic; all records are locked before any data is fetched/stored
  - Update event triggers are configurable
- **‘pva’** JSON link support
  - Record links can read/monitor data from other pvAccess servers (not just IOCs)
- **‘p2p’** PVA gateway
  - Provides fan-out & fan-in, connections between IP subnets
- Documentation for pva2pva module provides full details

PVA Tools and Language Bindings

pvAccess command-line tools

Provide similar functionality to the CA command-line tools

- **pvinfo**: Get server, connection state and introspection data of a channel
- **pvget**: Get “value” element (if one exists, else the complete structure)
- **pvmonitor**: Subscribe to monitor events from a channel
- **pvput**: Put data to “value” element of a channel
- **pvcall**: Make a remote procedure call (ChannelRPC)
- **pvlist**: Find and list available PVA servers, or list the channels provided by a specific server

- **eget**: Extended get/monitor client with better support for Normative Types (now unbundled)

Language Bindings: C++

- Original implementation: EPICS V4
  - pvData, pvAccess, normativeTypes, pvaClient, pvDatabase
  - Written by Marty Kraimer and Matej Sekonraja (Cosylab)
    - Neither were experienced C++ developers
  - Original code had many issues with object ownership, locking, complex API; reworked by Michael Davidsaver
    - Also added simpler ‘pva’ APIs for client and server code
  - Now stable, high performance

- New implementation: PVXS
  - Still in development, server already working
  - Similar to ‘pva’ APIs, code should convert easily
  - Written by Michael Davidsaver (Osprey DCS)
  - Requires C++11 compiler

Language Bindings: Python

- Original Python bindings: pvaPy
  - Client and server APIs, RPC
  - Written in C++ using Boost::Python
  - Available through PyPI and Conda Forge
    - Linux & MacOS, no Windows yet
  - Written and maintained by Siniša Veseli (APS, Argonne)

- Alternative Python bindings: p4p
  - Client and server APIs, RPC
  - Includes a PVA Gateway with access security
  - Written in C++ and Python
  - Available through PyPI (pip install)
    - Linux, MacOS and Windows
  - Written and maintained by Michael Davidsaver (Osprey DCS)

Language Bindings: Java

Two Java implementations

- Original version
  - Available from Maven Central
  - API somewhat hard to use, more like C++ than native Java
  - Written by Marty Kraimer and Matej Sekoranja (CosyLab), now maintained by CS-Studio Group (Kunal Shroff, NSLS-2)

- New implementation in development
  - Designed for use in Phoebus
  - APIs designed for native Java
  - Written by Kay Kasemir (SNS, ORNL) with Michael Davidsaver

Current Status

Who is using PVA at this time?

- **Many sites**: Using areaDetector to send images over pvAccess for display, further image processing, or to file storage
  - Using >90% of physical bandwidth on 10Gb ethernet (no compression)
- **NSLS-II**: Middle-layer services using structured data
  - MASAR service for saving/restoring setting snapshots
  - ChannelFinder, archiver, elog interface, ...
- **SNS Beamlines**: Implemented next generation of controls and data acquisition
- **SLAC**: Re-implementing high-level physics database access using pvAccess and middle-layer services
- **FHI**: Using the EPICS Archiver Appliance with pvAccess and structured data
- **APS-U**: New DAQ systems will transport high-speed machine data from front-ends to client applications

Conclusions

- EPICS 7: V4 extended V3 without replacing the IOC
- PVA modules add flexible structures and an efficient network protocol for transporting them
- Set of well-defined standard data container types to allow generic client applications to communicate with PVA servers
- Used in production and development at many sites

Thank you…