

*Supporting the next generation of scientific control systems*

# Introduction to EPICS V4

Design, Features and Status

---

Ralph Lange, Spring 2017 EPICS Collaboration Meeting

## *Preface: What V4 is not*

- V4 is not a replacement for V3
- V4 does not introduce a new IOC database
- V4 does not require you to rewrite all your drivers
- V4 does not break existing systems

V4 extends V3, and does not require upgrading

# *Outline: What's covered*

- What is V4 and what can I do with it?
- V4 key concepts
  - pvData and pvAccess
  - pvRequest
  - Normative Types
- V3/V4 interoperability
- pvTools
- How to get started with V4
  - Resources and modules
- Development and Status

What is V4 and what can I do with it?

---

# What is V4, in six words?

EPICS

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

**V4 adds structured data to EPICS**

# What is V4 good for?

- **Fix Channel Access problems:**  
Better array and string handling
- **Structured data:**  
Extending the scope of EPICS from I&C to 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 fixes 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):

Can construct pvData structures analogous to DBR types. For example the equivalent of a DBR\_TIME\_DOUBLE would be 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.

So 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 changed values (in **bold**) need be sent, plus a bitset indicating which fields have changed value.

# 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 analyse 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 provide remote access to, i.e. carry pvData structures
- 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

- Draft standard
- Request to limit operation to part of a structure
- Processing options (process, block)
- Monitoring options: queue size (deadband, server-side filtering in the future)
- RPC: TBD



# 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

|                  |            |      |
|------------------|------------|------|
| <i>scalar_t</i>  | value      |      |
| string           | descriptor | :opt |
| <i>alarm_t</i>   | alarm      | :opt |
| <i>time_t</i>    | timeStamp  | :opt |
| <i>display_t</i> | display    | :opt |
| <i>control_t</i> | control    | :opt |

*NTAggregate* :=

structure

|                |                |      |
|----------------|----------------|------|
| double         | value          |      |
| long           | N              |      |
| double         | dispersion     | :opt |
| double         | first          | :opt |
| <i>time_t</i>  | firstTimeStamp | :opt |
| double         | last           | :opt |
| <i>time_t</i>  | lastTimeStamp  | :opt |
| double         | max            | :opt |
| double         | min            | :opt |
| string         | descriptor     | :opt |
| <i>alarm_t</i> | alarm          | :opt |
| <i>time_t</i>  | timeStamp      | :opt |

- Specification of standard, named type
- Often choices (field types, field names)
- Required and optional fields
- Extra fields can be added
- *Italics* denote definitions (non-terminal terms)

# V3/V4 interoperability

---

# pvaSrv

- pvAccess server running on a regular V3 DB (aka IOC)
- **dbPv** function:
  - Any record.field PV is available as a standard NT structure
  - Zero configuration
- New implementation (QSRV) currently under development
- **dbGroup** function (*under development*):
  - Arbitrary groups of record.field PVs are accessible under a new name as a single structure
  - Configuration: mapping of record.field to pvData structure element  
*static persistent* (at IOC boot time) and/or *dynamic volatile* (on-the-fly as part of the operation)

# V3 / V4 bridging options

- **V4 pvAccess client library** handles pvAccess and Channel Access:  
New clients can connect to V4 and V3 servers  
→ *Does not need any change on V3 IOC*
- **pvaSrv/QSRV** puts a pvAccess server on top of the existing EPICS V3 DB:  
New clients can use pvAccess to connect to V3 IOCs  
→ *Needs pvaSrv and V4 libraries on V3 IOC*
- **Gateway** (*under development*) handles protocol conversion

# pvTools

---

# pvAccess commandline tools

Similar functionality as CA commandline tools:

- pvinfo: get server, connection state and introspection data of a channel
- pvget: get “value” element (if exists, else the complete structure)
- pvget -m: set up monitor subscription
- pvput: put data to “value” element
- pvlist: list available servers, or available channels on a specific server
  
- eget: extended get client with NTypes support

# How to get started with V4

---



# Resources

- Website on SourceForge: <http://epics-pvdata.sourceforge.net>  
(will move later this year with release of EPICS 7)
- Code on GitHub: <https://github.com/epics-base>

# Modules

**pvDataCPP/pvDataJava:** Implementation of pvData

**pvAccessCPP/pvAccessJava:** Implementation of pvAccess

**normativeTypesCPP/normativeTypesJava:** Implementation of Normative Types

**pvCommonCPP:** Boost shared pointers (for some OSs) and micro-benchmarking

**pvaSrv:** Adding pvAccess server to existing V3 IOC (C++ only)

**pvDatabaseCPP/pvDatabaseJava:** Record/Database library for creating V4 servers

**pvaClientCPP/pvaClientJava:** High-level simplified client library

**pvaPy:** EPICS V4 for Python library (client and server; wrapping C++ libraries)

# Development and status

---

# Software development

- V4 Development Group (since 2012)
  - ~4 developers actively involved (lost 2 in 2016)
  - Bi-weekly telecon, three face-to-face meetings per year
- Current 4.6.0 release
  - Usable (near production quality)
  - SNS are beating the bugs out  
(unit tests do not cover extreme conditions)
- Next step: **EPICS 7** – planned for October 2017
  - EPICS Base 3.16 plus EPICS V4 modules

$$3 + 4 = 7$$

# Who is using V4 at this time?

- **Diamond/NSLS-II:** transferring areaDetector images across the network / between processes using pvAccess
  - Using >90% of physical bandwidth on 10Gb ethernet (no compression)
- **NSLS-II:** middle-layer services using structured data
  - In production: MASAR service for saving/restoring setting snapshots
  - More services planned (channelFinder, archiver, elog interface, ...)
- **SNS Beamlines:** implementing next generation of controls and data acquisition
- **SLAC:** re-implementing all high-level physics database access using pvAccess and middle-layer services
- **FHI:** using archiver appliance with pvAccess and structured data

# Conclusions

- EPICS 7: V4 extends V3 without replacing the existing IOC
- pvData and pvAccess add flexible structures and an efficient network protocol
- Set of well-defined containers for generic clients to handle most applications
- Beyond beta status, first in-production users are happy
- Some important parts (Gateway, PV grouping) are still under development

*Thank you...*