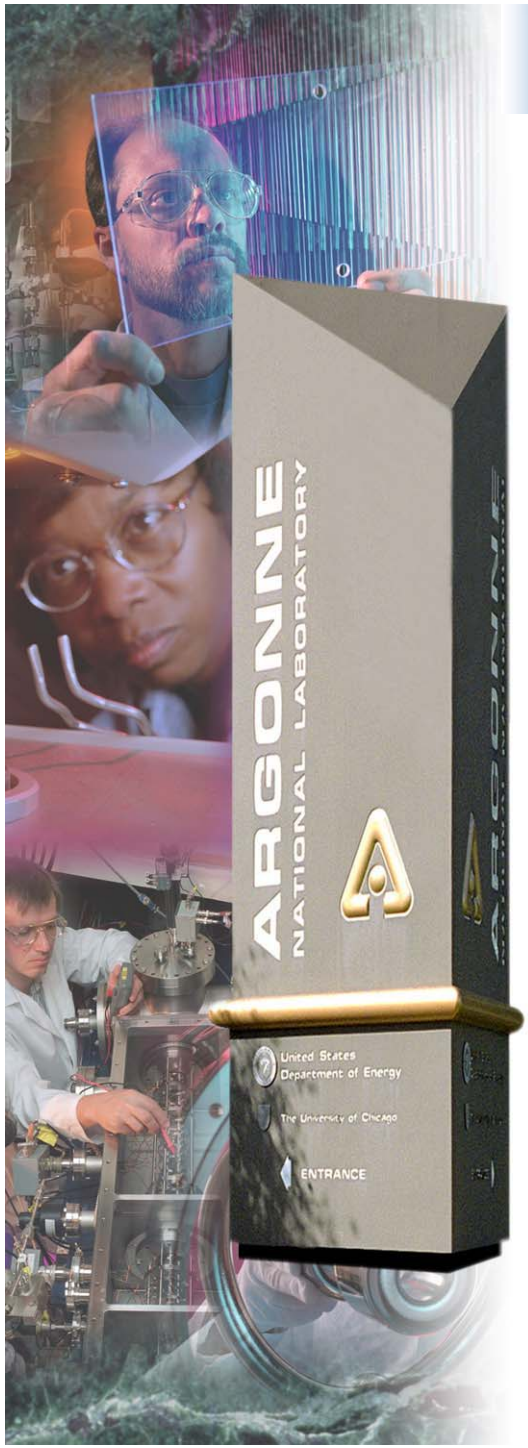

ARGONNE NATIONAL LABORATORY

ICALEPCS 2005: EPICS Workshop
EPICS V4 : Runtime Database

Marty Kraimer and Andrew Johnson

THE UNIVERSITY OF CHICAGO

Office of Science
U.S. DEPARTMENT OF ENERGY

Argonne National Laboratory is managed by
The University of Chicago for the U.S. Department of Energy

EPICS V4 Database – Summary Of New Features

- WARNING:
  - The design for the V4 Database is not complete
  - The ideas presented in this talk are subject to modification
- Database Definition Files and Record Instance Definitions
  - `struct` is a valid field type
  - Array fields are allowed
  - String is arbitrary length UFT-8
- Better Support for Data Acquisition
  - New Link Semantics - All can request
    - *Process*
    - *Wait until linked record completes*
    - *Link can block or allow other processing while waiting*
  - While active a record can be processed and post monitors
- `db_post_event` no longer exists – Database Access handles monitors
- Lock Sets replaced by per-record lock

EPICS V4 – Database Definition: Field Types

- Primitive types
  - boolean, octet, int16, int32, int64, float32, float64
    - *boolean is true,false but no conversion to/from integer*
    - *octet is 8-bit byte but no conversion to/from integer*
- string
  - Array of UTF-8 encoded characters
- Array field
  - 1-dimensional arrays of all field types are allowed
  - Multidimensional arrays may be restricted (only int or float?)
- enum – Like V3 but references a field that is an array of string
- menu – Like V3
- struct
  - struct is a dbd type and a field of a record can be a struct
- link

Example Record Definition: CalcRecord

- From dbCommon.dbd

```
struct(InputLinkData) {
      field(pvname,string)
      field(process,boolean)
      field(wait,boolean)
      field(block,boolean)
      field(inheritSeverity,boolean)
 }
struct(MonitorLinkData) {
    field(pvname,string)
    field(process,boolean)
    field(inheritSeverity,boolean)
}
struct(ProcessLinkData) {
    ...
}
struct(OutputLinkData) {
     ...
}
```

- Features
  - Options for processing and waiting
  - Option to block

- calcRecord.dbd

```
Include "dbCommon.dbd"
struct(CalcInpLink) {
    field(link, link(in,interface(LinkFloat64)))
    field(value,float64)
}
record(CalcRecord) extends RecordCommon {
    field(inp,array(struct(CalcInpLink)[]))
    field(units,string)
    field(displayLimit,struct(DisplayLimit))
    ...
    }
}
```

- *Feature*
  - *arbitrary number of input links*

Example Record Definition: MbbiRecord

- The following describes a state

```
struct(MbbiState) {
    field(name,string)
    field(value,array(octet[]))
    field (severity,menu(menuAlarmSevr))
}
```

- Part of the record definition

```
record(MbbiRecord) extends RecordCommon {
     field(state, array(struct(MbbiState)[]))
     ...
}
```

- Features
  - Octet[] allows a multiple of 8 bits. For example a 128-bit Digital I/O module
  - Arbitrary number of states are allowed

- MbbiState describes
  - name – the name for the state
  - value – bit pattern as array of octets
  - severity
- Record definition describes
  - state – an array of mbbiState
  - ...

Record Instance Examples

- Counter Record – count 0 to 10

```
CalcRecord counter = {
    scan = ".1 second";
    calc = "(a<10)?(a+1):0";
    inp = [1] {
        {link = monitorLink(LinkFloat64)
            {pvname=counter}
        }
    };
}
```

- Compute Max of recordA, recordB

```
CalcRecord computeMax = {
     calc = "(a>b)?a:b";
     inp = [2] {
        {link = monitorLink(LinkFloat64)
            {pvname=recordA;process=yes}
        },
        {link = monitorLink(LinkFloat64)
            {pvname=recordB;process=yes}
        }
     }
}
```

- mbbi Record 16 bit DAC, 3 states

```
MbbiRecord dac16 = {
    state = [3] {
        {name = "state1"; value = octet [2]{0x00,0x01}},
        {name = "state2"; octet[2]{0x00,0x02}},
        {name="state3";octet[2]{0x00,0x04}}
    }
    ...
}
```

V4 Links

- Monitor Link Data - Input
  - pvname
  - process – on new value
  - inheritSeverity
- Input Link Data - NOT monitored
  - pvname
  - process
  - Wait
  - block
  - inheritSeverity
- Output Link Data
  - pvname
  - process
  - Wait
  - block
  - inheritSeverity
- Process Link Data
  Replaces V3 fwd link
  - pvname
  - Wait
  - block

V4 Link Semantics

- pvname – Process Variable Name
- process
  - For MonitorLink it means to process record containing link
  - For others it means to request that linked record be processed
- wait – Wait until record completes processing before link completion
  - If process false then wait until next time record processes
  - For input wait BEFORE getting value
  - For output wait after putting value
  - For process wait until process completion
- block
  - If yes then do no other record processsing until this link completes
  - If no then allow other record processing

Data Acquisition Link Example

- Scan a sample
- Repeat the following until done
  - Move motor A to new position: record incA does this
  - Move motor B to new position: record incB does this
  - Wait until A and B are at new position
  - Take a sample; record getSample does this
  - NOTE: incA, incB, and getSample may all be a set of records
- Rules
  - Motors A and B can be moved together
  - Don't start a new move until Channel Access Client has received data
- Approach
  - The Channel Access Client will ask to go to new position
  - It will wait for record to complete, read sample, and then repeat

Some V4 Features

- `RecordCommon` replaces `dbCommon`
  - `RecordCommon` has associated record support
  - V3 `FLNK` field replaced by array of processLinks
- For our example we could use any record type
  - Just assume a record type void which has only RecordCommon

Record Type Void For Data Acquisition

- Assume that incA, incB, and getSample exist
- The following does the data acquisition:

```
void collectSample={
    processLink=[3] {
        {pvname = “incA”;          wait=true; block=false},
        {pvname= “incB”;           wait=true; block = true},
        {pvname= “getSample”;  wait=true; block=true}
    }
}
```

- The following occurs
  - The client issues a processWait request to record collectSample
  - collectSample requests that both incA and incB process; it waits
  - collectSample requests that getSample processes; it waits
  - collectSample completes
  - client receives completion notification, reads sample, and starts again

V4 Record Processing Semantics

- V3 record processing semantics did not work well for data acquisition
  - New V4 Link semantics help
  - SynApps (Xray beamline software) created new record types
    - *Scan, Motor, etc.*
    - *Set PACT false but do NOT call recGblFwdLink*
    - *Thus can be processed again even though they are not done*
    - *Can issue db_post_event so that current position can be monitored*
    - *When really done call recGblFwdLink*
    - *This behavior came as a big suprise to iocCore developers*
- V4 provides semantics that allow this behavior
  - Previous example showed new link behavior
  - Although not explained in this talk, the V4 processing semantics allow monitors to be posted
    - *While a record is in state processActive*
    - *At the completion of record processing*

Posting Monitors

- Database Access does all posting of monitors itself
  - It posts monitors when record support returns processActive
  - It posts monitors when RecordCommon returns processDone
  - Thus monitors are posted when state is processActive or when the record completes processing
- How can Database Access handle monitors itself?
  - Short answer is that database fields can ONLY be modified via an interface that is implemented by Database Access

Lock Sets Replaced By Per Record Lock

- V3 implemented lock sets
  - Allowed dbProcess to be called recursively
    - *Process passive database links implemented via recursion*
  - Prevented deadly embrace
  - For synchronous links fields can not be modified by Channel Access while records in lock set are being modified
    - *For asynchronous records this is no longer true*
- V4 queues requests rather than making recursive call to dbProcess in order to process linked records
  - Prevents stack overflow
  - Removes many complications of lock sets

V4 Record Processing: Performance Issues

- All access to database fields is via interface
  - Example:

```
interface DbfOctet extends Dbf {
    int16 get();
    void put(int16 val);
}
```

  - Storage Overhead
  - Runtime Overhead
- Queue Request instead of recursive call to dbProcess
  - Queue Request overhead
  - Context Switch overhead

Accessing Database Via Interface: Storage Overhead

- Storage is increased for fields that exist
  - Database Access keeps private info for every field
- BUT many fewer fields will exist
  - RecordCommon has fewer fields than the V3 dbCommon
    - *ProcessLink, which replaces the V3 FLNK is an array of links. It can be an empty array*
  - CalcRecord has array of struct(CalcInputLink).
    - *V3 always had storage for 16 links and associated info*
    - *V4 will only allocate storage for the number of links used*
  - Several other record types will have major savings, e.g. mbbi, mbbo
  - New link semantics will require fewer record instances

Accessing Database Via Interface: Runtime Overhead

- Overhead for primitive types is extra level of indirection
  - For gets it is just overhead
  - For puts posting monitors makes extra level of indirection insignificant
- Overhead for other types more severe
  - Arrays biggest problem
    - *Record or device support can provide storage and can share storage between record types for demanding applications*
- Will impose more discipline on record and device support
  - Don't use database to store private info
  - Set monitors so that it is not necessary to access field everytime record processes
  - Reward is not getting involved with monitors, etc.
- Data in IOC records is owned and controlled by database access, not by record and device support

Record Processing: Overhead of Queue Request

- Issuing a queue request instead of recursive call to dbProcess has more overhead
  - Queue request itself
  - Possible context switch
- Queue requests often not necessary
  - Using MonitorLink for input causes no queue request
  - Synchronous device support requires no queue request
  - InputLink and OutputLink with process and wait false require no queue request
- Many queue requests are because of new desired features of V4
- Optimization for context switch
  - When a queue request is issued, the scan thread will process this request as soon as the record issuing the request returns

WIKI Pages

- More Details are available in wiki pages
  - They are available via the main EPICS site http://www.aps.anl.gov/epics/
  - The core developer wiki pages are available via http://www.aps.anl.gov/epics/wiki/index.php/Core_Developer_Pages
- The current wikis of interest are those under the headings:
  - V4 Database Definition
  - V4 Database Runtime