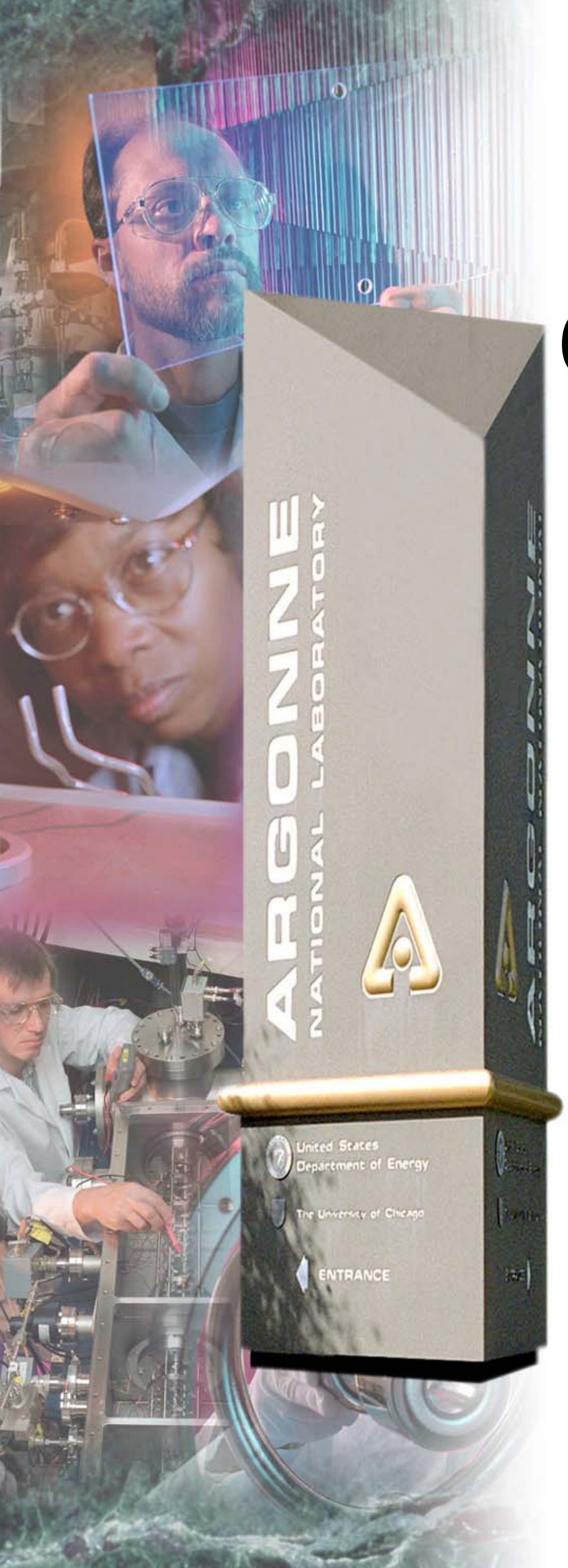

Getting Started With EPICS Lecture Series

Writing Record Support

Marty Kraimer, ASD/Controls
23 November 2004

Argonne National Laboratory

Advanced Photon Source

Office of Science
U.S. Department of Energy

A U.S. Department of Energy
Office of Science Laboratory
Operated by The University of Chicago

Overview

- **Record Support**
  - Implements a Record Type
  - Plugs into database common environment
- **Implementation steps**
  - Prepare a Database Definition (.dbd) file
  - Create a C source file which implements semantics
  - Start with existing support!!!
- **Documentation**
  - App Dev Guide 3.14.6 - Chapters 5, 6, and 11
  - makeBaseApp example - xxxRecord
  - base/src/rec
- **BUT FIRST: Ask the question Why?**
  - Reasons against a new record type
  - Reasons for a new record type

Reasons Against a New Record Type

- **Only one STAT, SEVR, VAL**
  - Example: Power Supply Record, i.e. multiple inputs, outputs. What do stat and sevr mean?
  - Is a template of standard record types a better choice?
- **No device support**
  - Standard record types have extensive associated device support
- **No documentation. You should also provide it.**
- **Can be tricky to implement proper semantics**
- **Some standard tools may also need extensions. Example capFast.**

Reasons For a New Record Type

- **Sets of standard record types will not satisfy need**
  - Are you really sure? Think hard!!!
- **State must be shared across multiple records**
  - Example asynRecord – Can dynamically change hardware addressing
    - *I implemented via device support and standard records. Complicated!!*
    - *Mark implemented via new record type. Much easier to understand.*
- **Many record instances: Example mbbiDirect record type.**
- **Because it is fun. NOT A GOOD REASON.**
- **Example Event Generator/ Event Receiver Records**
  - APS/SLS/Diamond/PLS – Special Record Types
  - SNS has different event system.
    - *Question: Modify EG/ER or write device support for standard records?*
    - *Answer: Not clear!!!*
      - Very few EG records
      - At most one ER per IOC
      - I think SNS decided to use device support for standard record types

Implementation: Create DBD.

- **Create a Database Definition (dbd) File.**
- **Chapter 6 of 3.14.6 Application Developer’s Guide.**
- **Syntax**

```
recordtype(<recordtype>) {
  include “dbCommon.dbd”
  field(<fieldname>,<fieldtype>) {
   <fieldattributes>
  }
  field(<fieldname>,<fieldtype>) {
   <fieldattributes>
  }
  ...
}
```

DBD Example

- **Example**

```
recordtype(example) {
  include “dbCommon.dbd”
  field(PREC,DBF_DOUBLE) {
   prompt(“Display Precision”)
   promptGroup(GUI_DISPLAY)
  }
  field(VAL,DBF_DOUBLE)
}
```

DBD continued

- `recordtype(<recordtype>)`
  Should be alphanumeric starting with letter
- `include “<filename>”`
  - `Must include dbCommon first.`
- `field(<fieldname>,<type>) {<attributes>}`
  - Fieldname should be alphanumeric starting with letter
  - Length is normally limited to four characters
    - *3.14 allows arbitrary length*
    - *May cause problems with some Channel Access clients*
  - In dbd file this should be upper case. It is converted to lower case when the <recordtype>.h file is generated. Bad decision!!

DBD – Field Attributes

Advanced Photon Source

- **<fieldattributes>**
  - asl : access security level
    - *ASL1 - Fields seldom modified - default*
    - *ASL0 - Fields meant to be modified*
  - initial : initial value
  - prompt : string for Database Configuration
  - promptgroup : Should DCT allow field to be configured?
  - special: Special handling on put
    - *Several values*
    - *SPC_MOD, SPC_DBADDR, SPC SPC_NOMOD*

DBD – Field Attributes continued

- **<fieldattributes> (continued)**
  - pp : process passive record on put?
  - interest : interest level used by dbpr (Database Print Record)
  - base : DECIMAL (default) or HEX (Only used by dbDumpRecord)
  - size : For DBF_STRING
  - extra : For DBF_NOACCESS
  - menu : For DBF_MENU.

DBD - Fieldtypes

- **`<fieldtype>`**

```
DBF_STRING
DBF_CHAR DBF_UCHAR
DBF_SHORT DBF_USHORT
DBF_LONG DBF_ULONG
DBF_FLOAT DBF_DOUBLE
DBF_ENUM
DBF_MENU
DBF_DEVICE
DBF_INLINK DBF_OUTLINK
DBF_FWDLINK
DBF_NOACCESS
```

DBD String Field

- **`DBF_STRING` fields**
  - Fixed length null terminated strings
  - Channel access support max of 40 characters
  - Arrays of character strings are possible
- **Example**

```
field(VAL,DBF_STRING) {
  size(40)
  prompt(“value”)
  promptgroup(GUI_DISPLAY)
}
```

DBD Numeric Fields

- **`DBF_CHAR, … , DBF_DOUBLE`**
  - Standard C datatypes
  - All attributes except size, extra, menu meaningful
- **Arrays**
  - dbd file must define `special(SPC_DBADDR)`
  - Record support must implement `cvt_dbaddr, get_array_info, put_array_info`
  - No support by Database Configuration Tools.

DBD: Enumerated Field

- `DBF_ENUM`
  - Record Support provides choices and index
  - See `biRecord`, `mbbiRecord`, etc
  - Actual field type is `unsigned short`
  - Record support MUST implement
    - *`get_enum_str`*
    - *`put_enum_str`*
    - *`get_enum_strs`*

DBD Menu Field

- **DBF_MENU**

```
menu(<menuname>) {
   choice(<choicename>,”<choicevalue>“)
   …
}
…
recordtype(<recordtype>) {
    field(<fieldname>,DBF_MENU) {
        menu(<menuname>)
    }
```

Menu Field Example

- **Example**

```
menu(menuYesNo) {
  choice(menuYesNo,”NO”)
  choice(menuYesNo,”YES”)
}
…
  field(PINI,DBF_MENU) {
   …
   menu(menuYesNo)
  }
```

DBD Device Field

- **`DBF_DEVICE`**
  - Database Common Field - `DTYP`
  - Related to device definitions
  - Has associated `INP` or `OUT` field
  - Device Support must be provided
- **Example**

```
field(DTYP,”Soft Channel”)
```

This requires a device definition like:

```
device(ai,CONSTANT,devAiSoft,”Soft Channel”)
```

DBD link Fields

- **DBF_INLINK, DBF_OUTLINK**
  - Not INP or OUT
    - *Constant link - Just initialization*
    - *PV_LINK becomes DB_LINK or CA_LINK*
  - INP or OUT and Device Support
    - *device definitions determine choices*
    - *Associated with choice is bus type*
    - *Gory details in link.h. Used by device support.*
- **DBF_FWDLINK**
  - references another record which is processed if it is passive.
  - FLNK
    - *In dbCommon*
    - *record support calls recGblFwdLink*
  - Other DBF_FWDLINKs can be defined
    - *Call dbScanFwdLink NOT recGblFwdLink*

Implementation: Generate include files

- **`Generate <menu>.h and <recordtype>.h include file`**
- **`dbToMenuH generates <menu>.h`**
  - Code should use generated files instead of hard coding choices
    - *Many standard records written before dbToMenuH existed.*
  - Example `menuYesNo.h`

```
typedef enum {
  menuYesNoNO,
  menuYesNoYES
} menuYesNo;
```

Generate Include Files continued

- **dbToRecordTypeH generates <recordtype>.h**
  - Code MUST use generated files
  - Include code to compute field sizes, offsets

```
typedef struct aiRecord {
  char name[29];
  …
}
#define aiRecordNAME 0
…
#ifdef GEN_SIZE_OFFSET
int aiRecordSizeOffset(dbRecordType *p)
{ … }
```

  - Only record support and associated device support is allowed to use the include files.

Implementation: Record Support Module

- **Provide Execution Semantics**
- **Defined via RSET - Record Support Entry Table**

```
/* Create RSET - Record Support Entry Table
#define report NULL
#define initialize NULL
static long init_record();
static long process();
…
struct rset <recordtype>RSET = {
  RSETNUMBER,
  report,
  initialize,
  init_record,
  process,
  …
};
```

Record Support Methods - Summary

- **Report, Initialization, Processing**
  - `report`
  - `initialize`
  - `init_record`
  - `process`

Record Support Methods - Summary

- **Support for Channel/ Database Access**
  - `special`
  - `cvt_dbaddr`
  - `get_array_info`
  - `put_array_info`
  - `get_units`
  - `get_precision`
  - `get_enum_str`
  - `put_enum_str`
  - `get_enum_strs`
  - `get_graphic_double`
  - `get_control_double`
  - `get_alarm_double`

Record Support Method: Syntax

- **RSET – Record Support Entry Table**
  - Except for RSETNUMBER just a structure of function pointers.
  - Similar to JAVA interface or pure virtual C++ class
  - BUT defined before standard C was available for EPICS
  - Does NOT use C function prototypes
  - Sorry!!!!
  - Perhaps this will change in EPICS version 4.

Methods: Report, Initialization, Processing

- **`report(void *precord)`**
  - Nothing calls this. Normally not implemented
- **`initialize(void)`**
  - Called once before first call to init_record .Not normally needed
- **`init_record(void *precord,int pass)`**
  - Normally implemented
  - Called twice
    - *pass 0 - Can do anything except access other records. For example storage for arrays should be allocated during pass 0.*
    - *pass 1 - Finish initialization and also call associated device support initialization*
- **`process`**
  - VERY IMPORTANT
  - Implements record semantics

init_record example

```
/* assume val is double *val */
typedef struct xxxdset {
    long number,
    …
    DEVSUPFUN init_record;
    …
    DEVSUPFUN read;
} xxxdset;
static long init_record(void *prec, int pass)
{
    xxxarrayRecord *paf = (xxxarrayRecord *)prec;
    xxxdest *pdset = (xxxdset *)paf->dset;

    if(pass==0) {
     if(paf->nelm<0) paf->nelm = 1;
     paf->val = (double *)calloc(paf->nelm,sizeof(double));
     return(0);
    }
    /* Must have dset defined */
    if(!pdset !! (pdset->number< 5) || !pdest->read ) {
     epicsPrintf(“%s no or invalid dset\n”,paf->name);
     return(0);
    }
    (*pdset->init_record)(paf);
    return(0);
}
```

process overview

- **`dbAccess:dbProcess calls process` if**
  - Decision was made to process the record
  - Record not active (`PACT FALSE`)
  - Record is enabled (`DISV !=DISA`)
- **`process` with device support must**
  - set record active (`PACT TRUE`)
  - Perform I/O
  - Check for record specific alarms
  - Raise database monitors
  - Request processing of forward links
  - Set `PACT FALSE` and return

process overview continued

- **For asynchronous devices.**
  - Asynchronous start
    - *Initiate I/O.*
    - *Determine a method for calling process when operation completes*
    - *Return leaving PACT TRUE.*
  - Asynchronous completion
    - *Call `process` as follows:*
      ```
      dbScanLock(precord);
      *(prset->process)(precord);
      dbScanUnlock(precord);
      ```
    - *`process` completes record processing*
      - Complete I/O
      - Check for record specific alarms
      - Raise database monitors
      - Request processing of forward links
      - Set PACT FALSE
      - return

process example

```
static long process(void *prec, int pass)
{
    xxxarrayRecord *paf = (xxxarrayRecord *)prec;
   xxxdest *pdset = (xxxdset *)paf->dset;
    unsigned char pact = paf->pact;
    unsigned short monitor_mask;

    /* Must have dset defined */
    if(!pdset !! (pdset->number< 5) || !pdest->read ) {
     /*set pact true so process will not be called*/
     paf->pact = TRUE;
     return(0);
    }
    (*pdset->read)(paf);
    /*return if beginning of asyn processing */
    if(!pact && paf->pact) return(0);
   paf->pact = TRUE;
    alarm(paf);
    /*sample monitor code */
    monitor_mask = recGblResetAlarms(paf);
    monitor_mask |= (DBE_LOG|DBE_VALUE)
    db_post_events(paf,paf->val,monitor_mask);
    recGblFwdLink(paf);
    paf->pact = FALSE
    return(0);
}
```

alarm and monitors

Advanced Photon Source

- **alarms**
  - `recGblSetSevr(void *prec, <alarm_status>,<alarm_severity>)`
  - Status and severity values defined in `alarm.h`
  - Must prevent alarm storms on numeric values
    - *value fluctuating around alarm limit*
    - *See Application Developer's guide for algorithm*
  - process must call `recGblResetAlarms` after all alarms have been raised and before calling `recGblFwdLink`. Also, if possible, before raising monitors.
- **monitors**
  - `db_post_event(void *prec,void *pfield, monitor_mask)`
  - Call for all fields that change as a result of record processing
  - Support value change hysteresis when possible.
  - Look at `aiRecord` for an example

Routines with a DBADDR argument

- **Called by database/channel access**
- **Call `dbGetFieldIndex` to find out which field.**

```
int fieldIndex;
fieldIndex = dbGetFieldIndex(pdbaddr);
if(fieldIndex == aiRecordVAL)
   ...
```

- **DBADDR**

```
typedef struct dbAddr{
 dbCommon *precord;
 void *pfield;
 void *pfldDes;
 void *asPvt;
 long no_elements;
 short field_type;
 short field_size;
 short special;
 short dbr_field_type;
}DBADDR;
```

Record Support Routines - Cont.

- **`special(DBADDR *pdbaddr,int after)`**
  - Called if `special` defined in dbd file
  - Called before and after field is modified.
  - `dbGetFieldIndex(pdbaddr)`
  - `pdbaddr->special`
    - *The special type as defined in .dbd file. USE SPC_MOD.*
    - *Must include file `special.h`*
- **`cvt_dbaddr(DBADDR *pdbaddr)`**
  - Called by database library when DB or CA connects to field.
  - Called if `special(SPC_DBADDR)` specified
  - Following fields of `DBADDR` can be changed

    `no_elements`
    `field_type`
    `field_size`
    `special`
    `dbr_type`

Record Support Routines - Cont.

Advanced Photon Source

- **Array Routines**
  - Called if `cvt_dbaddr` sets no_elements > 1
  - `get_array_info(DBADDR *pdbaddr, long *no_elements, long *offset)`
    - *Called when database access gets an array*
    - *Must set `no_elements`*
    - *Can set offset if field is circular buffer*
  - `put_array_info( DBADDR *pdbaddr,long nNew)`

    Called after database access has written array

Record Support Routines - Cont.

- **Units/Precision**
  - `get_units(DBADDR *pdbaddr, char *punits)`

    CA has `MAX_UNITS_SIZE 8`

  -
    ```
    get_precision(DBADDR *pdbaddr,
      long *precision)
    ```
    Problems with only precision.
- **Enumerated Values**
  - `get_enum_str(DBADDR *pdbaddr,char *p)`
  - `put_enum_str(DBADDR *pdbaddr,char *p)`
  - `get_enum_strs(DBADDR *pdbaddr,dbr_enumStrs *p)`

    CA has limit of 16 choices, 26 characters in each choice
- **Graphic/ Control/ Alarm Limits**
  - `get_graphic_double(DBADDR *pdbaddr,struct dbr_grDouble *p)`
  - `get_control_double(DBADDR *pdbaddr,struct dbr_ctrlDouble *p)`
  - `get_alarm_double(DBADDR *pdbaddr,struct dbr_alDouble *p)`

Global Record Support Routines

- **All start with `recGbl`**
- **Intended for record/device support but also called by other code**
- **Reporting Errors**
  Deprecated - Use `errlogPrintf` instead
- **Alarm Processing**
  - `recGblSetSevr(void *precord, short newstat, short newsevr)`
  - `recGblResetAlarms(void *precord);`
- **Limits**
  - `recGblGetGraphicDouble, recGblGetControlDouble`, etc.
  - Defaults for fields record support doesn’t know how to handle
  - Channel Access client often not happy
  - Graphics, Control, Alarm limits
- **`recGblInitConstantLink`**
  For use on link fields with a constant value. Only works on numeric values!!

Global Record Support Routines continued

- `recGblGetTimeStamp`
  record support should call this each time record completes processing.
- `recGblFwdLink`
  Must be called with pact true and all other processing complete

Database Access Routines for Record Support

- **When in context of process use dbGetLink, dbPutLink**
- **Link information**
  - `dbGetPdbAddrFromLink`
    gets address of DBADDR.
  - `dbGetRset`
    gets address of Record Support Entry table
- **`dbIsLinkConnected`**
  Is link connected?