Custom Device Support

Kay Kasemir

Jan. 2019

Material copied from APS
EPICS Nomenclature

Record: Database processing block
- AI record: ‘read’ a number,
  AO record: ‘write’ a number,
  STRINGOUT: ‘write’ a string, ...

Device Support: Links Record to Driver
- AI device support: `read(aiRecord *ai)`
- AO device support: `write(aoRecord *ao)`

Driver: Code that talks to hardware
- Ideally available as C(++) source code
- Could be in binary form, from hardware vendor
- May be totally unaware of EPICS
1. Assume given ‘driver’ with \texttt{XyzDriver\_read()}.

2. Implement ‘device support’ for AI:

   ```c
   // Called by AI record when processed
   int xyz\_ai\_read(aiRecord * const ai)
   {
     // Call driver to get number
     const int raw\_number = XyzDriver\_read();
     // Put into record’s raw value field
     ai->rval = raw\_number;
     // Done, no error
     return 0;
   }
   ```

3. Some boilerplate to inform EPICS that AI record now has a new \texttt{DTYP=“XYZ”} that should call \texttt{xyz\_ai\_read()}. 
Of course, there’s more

- Set AI record’s RVAL, let the record convert to EGU, or set the record’s VAL?
- How to decide what to read exactly?

```plaintext
record(ai, "MyXYZTest")
{
    field(DTYP, "XYZ")
    field(INP, "#C0 S2 @unipolar")
    ...
}
```

- Handle errors?
- What if instead of
  Record gets scanned → read from device
  ...
  I want
  Device changes → Process the record!
Assume a simple Driver

```c
// simple_driver.

// Read value from channel 0, 1, 2, ...
int simple_read(int channel);

// simple_driver.c
#include <stdlib.h>
#include "simple_driver.h"

int simple_read(int channel)
{
    return channel * 100 + random() / (RAND_MAX / 10) - 5;
}
```
Device Support for AI Record

// simple_device.c

// std
#include <stdlib.h>
#include <stdio.h>

// EPICS
#include <rec32bi.h>
#include <devSup.h>
#include <dev_db.h>
#include <rec32bi.h>
#include <aiRecord.h>
#include <epicsExport.h>

// Local
#include "simple_driver.h"

// Init routine, called at startup
static long simple_init_ai(aiRecord *ai)
{
    int channel = atoi(ai->imp.value.instio.string);
    printf("Record \%s': Init. w/ channel \%d\n", ai->name, channel);
    ai->dpvt = (void *)(long) channel;
    return 0;
}

// Read routine, called whenever record is processed
static long simple_read_ai(aiRecord *ai)
{
    int channel = (int)(long) ai->dpvt;
    ai->rval = simple_read(channel);
    if (ai->rpro)
        printf("Record \%s': channel \%d = \%d\n", ai->name, channel, ai->rval);
    return 0;
}

// Boilerplate
// Device Support Entry Table for AI
static struct
{
    long number;
    long (*propt) (int);
    long (*initialize) (int);
    long (*initRecord) (aiRecord *);
    long (*getIoInfo) () ;
    long (*read) (aiRecord *);
    long (*special_linconv) (aiRecord *, int);
} devAiSimple =
{
    6, NULL, NULL, simple_init_ai, NULL, simple_read_ai, NULL
};
// Magic for different OS to 'export' this structure
epicsExportAddress dezset, devAiSimple);
DBD File

“simple.dbd”:

device(ai, INST_IO, devAiSimple, "Simple")
Makefile

• Compile the sources:

```makefile```
example_SRCS += simple_device.c
example_SRCS += simple_driver.c
```

• Include the DBD:

```makefile```
example_DBD += simple.dbd
```
Example Database

record(ai, "simple1")
{
  field(DTYP, "Simple")
  field(INP, "@1")
  field(SCAN, "1 second")
}

$ camonitor simple1

simple1                        2013-02-06 13:15:00.003208 100
simple1                        2013-02-06 13:15:01.003284 103
simple1                        2013-02-06 13:15:02.003370 101
simple1                        2013-02-06 13:15:03.003435 97
“Device Private”, DPVT

Used to store whatever you need to store
- Information fetched
  at initialization,
  needed for read/write
- Pointers to driver structures

Previous example: Channel #
- Misusing the (void *)rec->dpvt as (int)
Proper use of DPVT with custom struct

typedef struct
{
    int channel;
    // There would be a ton more in a real example ...
} StuffINeedToKeep;

// Init routine, called at startup
static long simple_init_ai(aiRecord *ai)
{
    StuffINeedToKeep *sintk = malloc(sizeof(StuffINeedToKeep));
    sintk->channel = atoi(ai->inp.value.instio.string);
    printf("Record '%s': Init. w/ channel %d\n",
        ai->name, sintk->channel);
    ai->dpvt = sintk;
    return 0;
}

// Read routine, called whenever record is processed
static long simple_read_ai(aiRecord *ai)
{
    StuffINeedToKeep *sintk = (StuffINeedToKeep *) ai->dpvt;
    ai->rval = simple_read(sintk->channel);
    if (ai->tpro)
        printf("Record '%s': channel %d = %d\n",
            ai->name, sintk->channel, ai->rval);
    return 0;
}
Recapitulate: From DTYP to read()
The .dbd file entry

The IOC discovers device support from entries in .dbd files

device(recType, addrType, dsetName, ”dtypeName”)

addrType is one of

- AB_IO
- BITBUS_IO
- CAMAC_IO
- GPIB_IO
- INST_IO
- RF_IO
- VME_IO
- VXI_IO

dsetName is the name of the C Device Support Entry Table (DSET)

By convention name indicates record and hardware type:

device(ai, GPIB_IO, devAidg535, "dg535")
device(bi, VME_IO, devBiXy240, "XYCOM-240")
Read-worthy sections of EPICS App. Devel. Guide

- OS-independent routines for register access, threads, semaphore, interrupts, ...

- Support for SCAN="I/O Intr", DSET getIoIntInfo()

- Support for conversions, DSET specialLinconv()
A Problematic Example

• See Problematic.pdf
Asynchronous I/O – ASYN

• This should be your first consideration for new device support

• It provides a powerful, flexible framework for writing device support for
  – Message-based asynchronous devices
  – Register-based synchronous devices

• Subsequent lecture
Stream Device

Useful for a lot of ‘intelligent’ I/O

- RS232 serial line devices
- Text-based TCP protocol devices
Summary: Device Support is

• Glue between records and hardware ("driver")

• Fundamentally easy:
  – Maybe "init()"
  – "read()" or "write()"
  – Boilerplate to register: DSET, *.dbd "device(...)"

• A great opportunity to shoot yourself in the foot