OAK RIDGE National Laboratory

Device Support Interfaces in EPICS base

Kay Kasemir

Material copied from APS
"Getting Started with EPICS Lecture Series: Writing Device Support", Eric Norum, November 16, 2004

July 2026

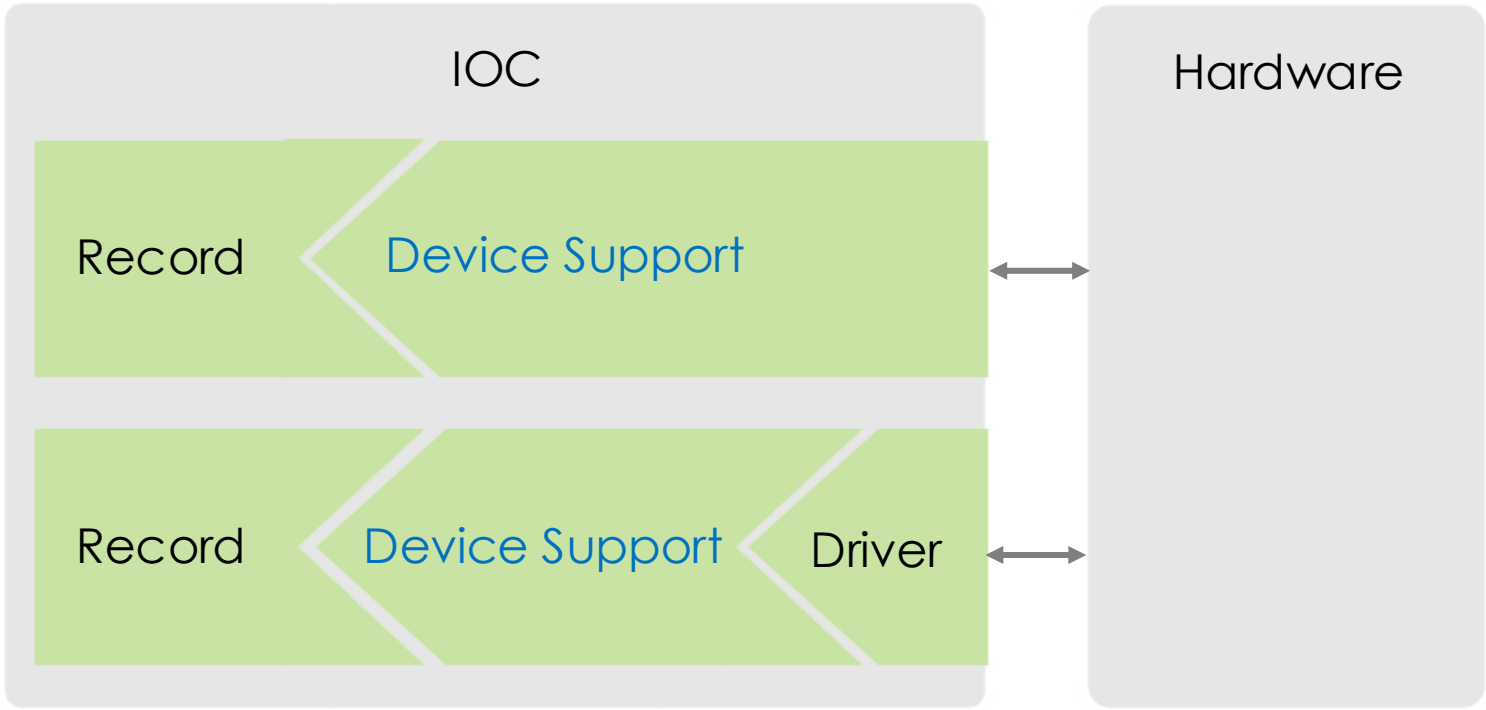
ORNL is managed by UT-Battelle, LLC for the US Department of Energy

U.S. DEPARTMENT OF ENERGY

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 Hardware, maybe via 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

Fundamentally Easy

1. Assume existing 'driver' with `XyzDriver_read()`
2. Implement 'device support' for AI:

```
// Called by AI record when processed
long ai_read_xyz(aiRecord *ai)
{
        // Call driver to get number
        // into record's raw value field
        ai->rval = XyzDriver_read();
        // Done, no error
        return 0;
}
```

3. Some boilerplate to inform EPICS that AI record can now use `DTYP="XYZ"`
   which should call `ai_read_xyz()`

Of course, there's more

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

```
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!

Example: Assume a simple Driver

```
/* drvRandom.h */
double drvRandom(double upper_limit);
```

```
/* drvRandom.cpp */
#include <stdlib.h>

double drvRandom(double upper_limit)
{
    return random() * upper_limit / RAND_MAX;
}
```

/ics/examples/20_devsup/devsupApp/src

Example Database

```
# my_device.db
# Example of AI record that uses DTYP "random"
record(ai, "$(user):aiRandom")
{
    field(DESC, "Random Test")
    field(DTYP, "random")
    field(INP,  "10.0")
    field(SCAN, "1 second")
}

record(ai, "$(user):aiRandom2")
{
    field(DESC, "Random Test")
    field(DTYP, "random")
    field(INP,  "100.0")
    field(SCAN, "1 second")
}
```

Idea:

Random number 0 .. 10

Random number 0 .. 100

/ics/examples/20_devsup/devsupApp/Db

Device Support for AI Record

```
/* devAiRnd.cpp - Example device support for Ai record */
#include <stddef.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "devSup.h"
#include "recGbl.h"
#include "dbAccessDefs.h"
#include "dbCommon.h"
#include "aiRecord.h"
#include "drvRandom.h"
#include "epicsExport.h"

/* Almost any device needs to maintain some data:
 * Address of hardware, state of communication with device, ...
 * In this case it's only the upper limit of the random  number generation.
 */
struct devRndData
{
    double upper_limit;
};

static long init_record(dbCommon *common)
{
    aiRecord *rec = (aiRecord *) common;
    devRndData *data;

    /* ai.inp must be a CONSTANT, defining the upper limit */
    if (rec->inp.type != CONSTANT)
    {
        recGblRecordError(S_db_badField, rec,
                          "devAiRnd (init_record) Illegal INP field");
        return S_db_badField;
    }
    data = new devRndData();
    recGblInitConstantLink(&rec->inp, DBF_DOUBLE, &data->upper_limit);
    /* device private (dpvt) is where we can park our device data */
    rec->dpvt = data;

    return 0;
}
```

```
static long read_ai(aiRecord *rec)
{
    devRndData *data = (devRndData *) rec->dpvt;
    if (data)
    {
        rec->val = drvRandom(data->upper_limit);
        rec->udf = FALSE;
    }
    return 2; /* 2 == don't convert rval to val */
}

/* Create the device support entry table */
aidset devAiRnd =
{
    {
        6,              // Number of methods
        NULL,           // report
        NULL,           // overall init
        init_record,    // init(dbCommon *)
        NULL            // get_ioint_info
    },
    read_ai,            // read(aiRecord *)
    NULL                // special_linconv
};
epicsExportAddress(dset, devAiRnd);
```

/ics/examples/20_devsup/devsupApp/src

DBD File

“my_device.dbd”:

```
device(ai, CONSTANT, devAiRnd, ”random")
```

/ics/examples/20_devsup/devsupApp/src

Makefile

```
TOP=../..

include $(TOP)/configure/CONFIG

# Use typed structures (since 3.16.2, see release notes)
USR_CPPFLAGS += -DUSE_TYPED_RSET -DUSE_TYPED_DSET

# Build the IOC application
PROD_IOC = devsup
DBD += devsup.dbd
# devsup.dbd will be made up from these files:
devsup_DBD += base.dbd
devsup_DBD += my_device.dbd

# Out driver & device support sources
devsup_SRCS += drvRandom.c
devsup_SRCS += devAiRnd.c

# devsup_registerRecordDeviceDriver.cpp derives from devsup.dbd
devsup_SRCS += devsup_registerRecordDeviceDriver.cpp

# Build the main IOC entry point on workstation OSs.
devsup_SRCS_DEFAULT += devsupMain.cpp
devsup_SRCS_vxWorks += -nil-

# Finally link to the EPICS Base libraries
devsup_LIBS += $(EPICS_BASE_IOC_LIBS)

include $(TOP)/configure/RULES
```

From

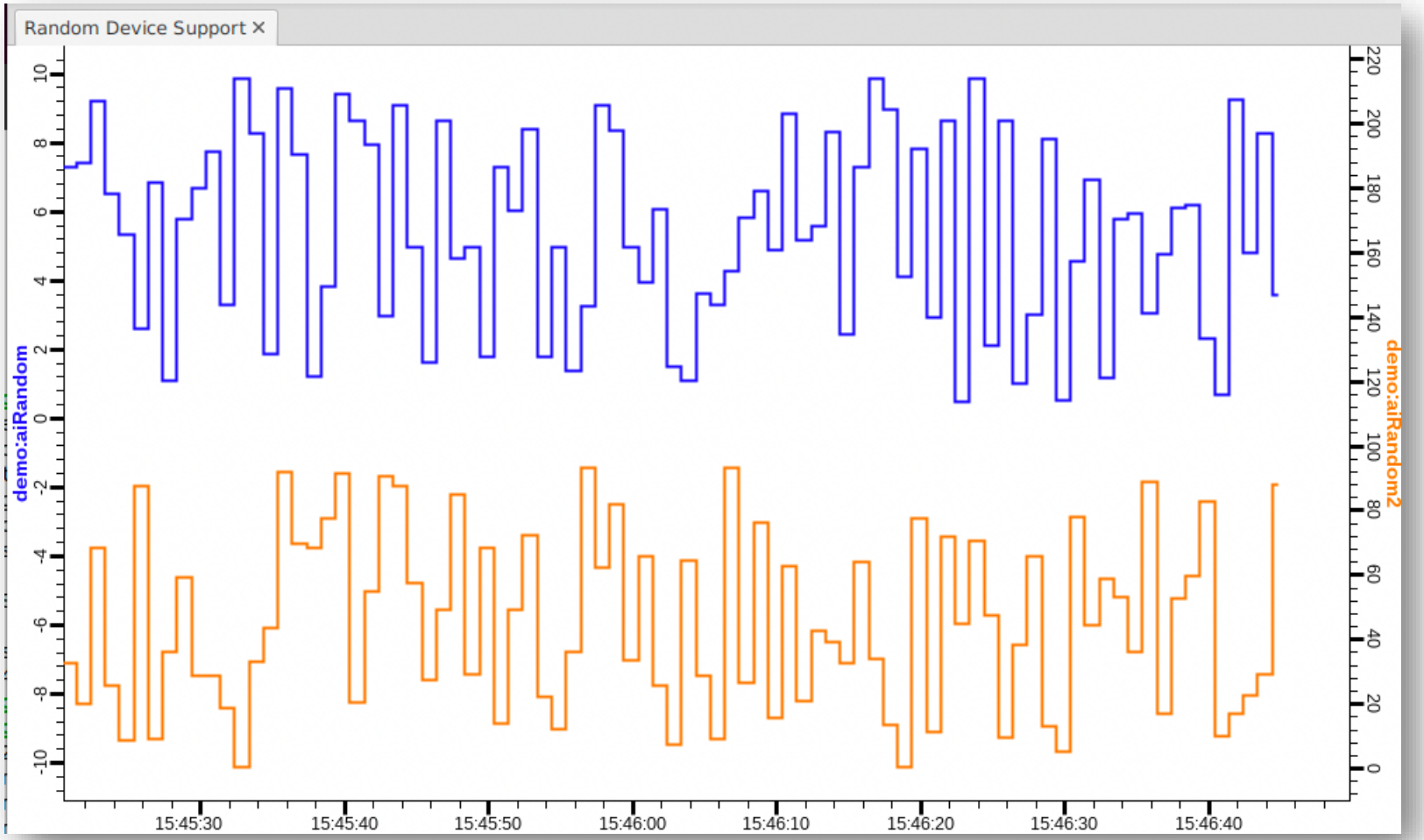
`makeBaseApp.pl -t ioc`

then adding our code

/ics/examples/20_devsup/devsupApp/src

IOC

```
cd /ics/examples/iocBoot/ioc_deviceSupport
./st.cmd
```



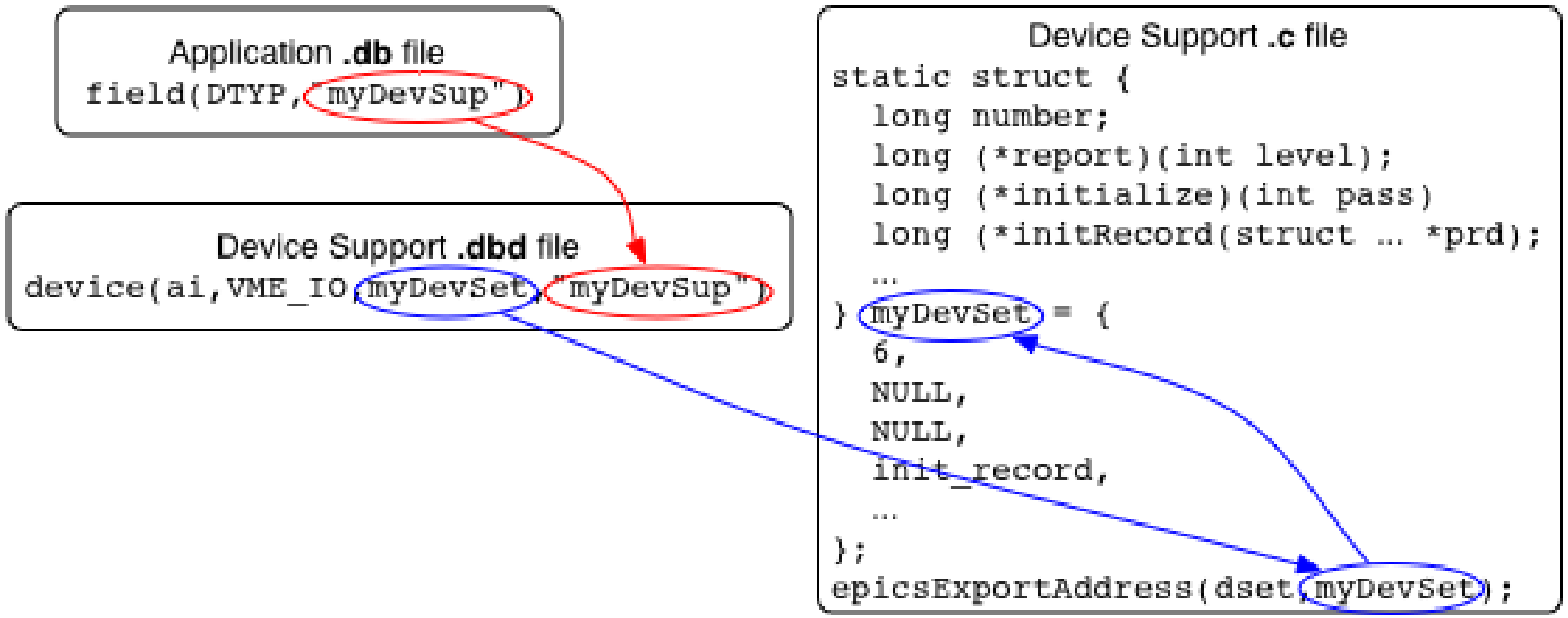
“Device Private”, DPVT

Used to store whatever you need to store

- Information fetched at initialization, needed for read/write
- Pointers to driver structures

Previous example: ‘devRndData’

Recapitulate: From DTYP to read()



The .dbd file entry

The IOC discovers device support from entries in .dbd files

```
device(recType, addrType, dsetName, "dtypeName")
```

recType is `ai ao …`

addrType is one of

```
CONSTANT    AB_IO        BITBUS_IO    CAMAC_IO    GPIB_IO
INST_IO     RF_IO        VME_IO       VXI_IO
```

```
CONSTANT: A number
INST_IO:  String
```

dsetName is the name of the 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()`

More Examples

20c Custom Device Support VME.txt

- Register-based hardware can be easy

20d Device Support_Problematic.pdf

- What is wrong?
- Compare Stream Device

```
out "*01X01";
in  "%f";
```

What's problematic in the 2nd example?

- ✓ Record with INP="10.1.2.3" will connect to that IP, send "*01X01" and read number into record's VAL
- Could split "device" and "driver"

❑ Can't have records with different INP
  - ✓ Use rec->dpvt instead of one global monitor_ip

❑ Connects/disconnects each time. Slow!!
  - ✓ Should keep connection open

❑ Blocks while reading
  - ✓ Driver must connect, read, … in separate thread

❑ Exit() on connection error
  - ✓ Set rec->sevr and stat to INVALID, READ

❑ Not checking format of received value

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
- Consider building on StreamDevice or Asyn