```
// Network driver example:
//
// Assume we have a networked device that listens on
// a TCP port '7654'. A record like the following should
// read from that device.


record (ai, "Netdev")
{
        field (DTYP, "Netdev")
        field (INP , "@localhost:7654")
        field (SCAN, "1 second")
}

______________________________________________________________________________
______________________________________________________________________________
______________________________________________________________________________


/* drvNetdev.h
 *
 * Example for a networked driver that's suitable for integration
 * into an EPICS IOC.
 *
 * This demo driver is meant to be used with e.g. "netcat":
 * Run netcat as a TCP server, simulating a networked
 * device that's awaiting requests.
 *
 * The driver will connect to the TCP server and send a request "Value?".
 * You then have about 2 seconds to type a number.
 * If you type a number, that's the value that the driver will read.
 * If you don't type anything useful in time, the driver will time out
 * and try again later.
 *
 * kasemir@lanl.gov
 */

/* EPICS Base */
#include <osiSock.h>
#include <epicsMutex.h>

typedef struct
{
    struct       sockaddr_in ip;
    epicsMutexId mutex;    /* lock before touching anything else in here! */
    int          is_valid; /* value is only valid if is_valid > 0 */
    int          value;    /* most recent value from the device */
} drvNetdev;

/* Launch a driver thread for the given address (Format: "IP:port").
 * The driver will continually try to connect to the address&port.
 * While connected, it'll send requests.
 * Whenever receiving a response, it will update the drvNetdev.value.
 * Any error will show up as drvNetdev.is_valid = 0.
 */
drvNetdev *drvNetdev_init(const char *address);

______________________________________________________________________________
______________________________________________________________________________
______________________________________________________________________________


/* System */
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
/* EPICS Base */
#include <epicsThread.h>
#include <iocsh.h>
#include <epicsExport.h>
/* This driver */
#include "drvNetdev.h"

static int verbosity = 10;
```

```
/* Driver task that tries to connect to the network device,
 * then sends requests and parses the response.
 *
 * This code is EPICS−specific because we use "epicsSocketCreate" etc.
 * This allows the code to run on any OS supported by EPICS base:
 * Linux, Solaris, Win32, vxWorks, RTEMS, ....
 *
 * Replacing epicsSocketCreate() with socket(),
 * epicsMutexCreate() with e.g. pthread_mutex_init() and so on
 * would turn this into code that knows nothing about EPICS
 * and is specific to Linux.
 */
static void driver_task(drvNetdev *drv)
{
    SOCKET s;
    char buf[100];
    fd_set fds;
    struct timeval timeout;
    int total_len, len;
    int waiting_for_response;

    while (1)
    {
        s = epicsSocketCreate(AF_INET, SOCK_STREAM, 0);
        sockAddrToDottedIP((const struct sockaddr *)&drv−>ip,
                           buf, sizeof(buf));
        if (verbosity > 1)
            printf("Connecting to %s\n", buf);
        if (connect(s, (const  struct sockaddr *)&drv−>ip, sizeof(drv−>ip))
            != 0)
        {
            fprintf(stderr, "driver_task cannot connect to %s\n", buf);
            epicsThreadSleep(5.0);
            continue; /* retry after 5 seconds */
        }
        if (verbosity > 1)
            printf("Connected\n");
        while (1)
        {
            if (verbosity > 1)
                printf("Sending request\n");
            if (write(s, "Value?\n", 7) != 7)
                break;
            /* Read response, ending in ’\n’ */
            waiting_for_response = 1;
            total_len = 0;
            while (waiting_for_response)
            {   /* try to read some characters */
                FD_ZERO(&fds);
                FD_SET(s, &fds);
                timeout.tv_sec = 5; /* 5 second read timeout */
                timeout.tv_usec = 0;
                if (select(s+1, &fds, 0, 0, &timeout) > 0  &&
                    (len = read(s, buf+total_len, sizeof(buf)−total_len)) > 0)
                {   /* Anything? Add to buffer. */
                    total_len += len;
                    buf[total_len] = ’\0’;
                    if (verbosity > 1)
                        printf("Got: ’%s’\n", buf);
                    if (strchr(buf, ’\n’))
                    {   /* Is it a full response ? */
                        epicsMutexLock(drv−>mutex);
                        drv−>value = atoi(buf);
                        drv−>is_valid = 1;
                        epicsMutexUnlock(drv−>mutex);
                        waiting_for_response = 0;
                    }
                }
                else
                {   /* Nothing. Time out, start over. */
                    if (verbosity > 1)
```

```
                        printf("Timeout / no data\n");
                    epicsMutexLock(drv->mutex);
                    drv->is_valid = 0;
                    epicsMutexUnlock(drv->mutex);
                    waiting_for_response = 0;
                }
            }
        }
        epicsMutexLock(drv->mutex);
        drv->is_valid = 0;
        epicsMutexUnlock(drv->mutex);
        if (verbosity > 1)
            printf("Disconnecting\n");
        epicsSocketDestroy(s);
    }
}

drvNetdev *drvNetdev_init(const char *address)
{
    if (verbosity > 1)
        printf("drvNetdev_init(%s)\n", address);
    drvNetdev *drv = calloc(1, sizeof(drvNetdev));
    if (drv)
    {
        aToIPAddr(address, 7543, &drv->ip);
        drv->mutex = epicsMutexCreate();
        epicsThreadCreate("drvNetdev",
                          epicsThreadPriorityLow,
                          epicsThreadStackMedium,
                          (EPICSTHREADFUNC)driver_task, drv);
    }
    return drv;
}

/* EPICS Driver support entry table: Don't need one. */

/* IOC Shell Registration Stuff.
 * None of this is required to use the driver
 * from an EPICS device support module.
 * Registration allows interactive stand-alone testing
 * of the driver from the EPICS IOC shell,
 * which can be useful when debugging the driver.
 * Yes, it's ugly but there really isn't much to it.
 */
static const iocshArg verbArg0 = {"value", iocshArgInt};
static const iocshArg *const verbArgs[1] = {&verbArg0};
static const iocshFuncDef verbosityDef = {"drvNetdev_verbosity", 1, verbArgs};
static void verbosityCall(const iocshArgBuf * args)
{       verbosity = args[0].ival; }

static const iocshArg initArg0 = {"address", iocshArgString};
static const iocshArg *const initArgs[1] = {&initArg0};
static const iocshFuncDef initDef = {"drvNetdev_init", 1, initArgs};
static void initCall(const iocshArgBuf * args)
{       drvNetdev_init(args[0].sval); }

static void drvNetdevRegistrar(void)
{
    static int firstTime = 1;
    if (firstTime)
    {
        firstTime = 0;
        iocshRegister(&verbosityDef, verbosityCall);
        iocshRegister(&initDef, initCall);
    }
};
/* Refer to this in DBD: registrar(drvNetdevRegistrar) */
epicsExportRegistrar(drvNetdevRegistrar);

------------------------------------------------------------------------------
------------------------------------------------------------------------------
------------------------------------------------------------------------------
```

```
/* devNetdev.c
 * Device support: drvNetdev <-> ai record
 *
 * Note that most of this is copied from EPICS base’s
 * src/dev/soft/devAiSoftRaw.c
 */
/* System */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
/* EPICS base */
#include "alarm.h"
#include "dbDefs.h"
#include "dbAccess.h"
#include "dbEvent.h"
#include "recGbl.h"
#include "recSup.h"
#include "devSup.h"
#include "link.h"
#include "aiRecord.h"
#include "epicsExport.h"
/* Driver */
#include "drvNetdev.h"

static long init_record(aiRecord *rec)
{
    if (rec->inp.type == INST_IO)
    {
        /* We expect INP="<IP>:port".
         * Init. driver, park the driver info pointer
         * in the "device private" member of the record.
         */
        rec->dpvt = drvNetdev_init(rec->inp.value.instio.string);
    }
    else
    {
        recGblRecordError(S_db_badField, rec,
                          "devAiNetdev (init_record) Illegal INP field");
        return S_db_badField;
    }
    return 0;
}

static long read_ai(aiRecord *rec)
{
    drvNetdev *drv = (drvNetdev *)rec->dpvt;
    if (drv)
    {
        epicsMutexLock(drv->mutex);
        if (drv->is_valid)
        {
            rec->rval = drv->value;
            rec->udf = 0;
        }
        else
        {
            recGblSetSevr(rec, READ_ALARM, INVALID_ALARM);
        }
        epicsMutexUnlock(drv->mutex);
    }

    return 0;
}

/* Create the dset for devAiSoftRaw */
struct
{
        long            number;
        DEVSUPFUN       report;
        DEVSUPFUN       init;
        DEVSUPFUN       init_record;
        DEVSUPFUN       get_ioint_info;
        DEVSUPFUN       read_ai;
        DEVSUPFUN       special_linconv;
}  devAiNetdev =
{
        6,
        NULL,
        NULL,
        init_record,
        NULL,
        read_ai,
        NULL
};
/* For DBD: device(ai,INST_IO,devAiNetdev,"Netdev") */

epicsExportAddress(dset,devAiNetdev);

------------------------------------------------------------------------------
------------------------------------------------------------------------------
------------------------------------------------------------------------------

# Example DBD file that loads the network device driver & device support
# together with EPICS base records

include "base.dbd"
registrar(drvNetdevRegistrar)
device(ai,INST_IO,devAiNetdev,"Netdev")
```