How to create EPICS device support for a simple serial device

W. Eric Norum
<norume@aps.anl.gov>

February 19, 2004

Contents

1 Introduction

This tutorial provides step-by-step instructions on how to create EPICS support for a simple serial device. The steps are presented in a way that should make it possible to apply them in cookbook fashion to create support for other serial devices. For comprehensive description of all the details of the I/O system used here, refer to the asynDriver and devGpib documentation.

This document isn't for the absolute newcomer though. You must have EPICS installed on a system somewhere and know how to build and run the example application. In particular you must have the following installed:

- EPICS R3.14.5 or higher
- An up-to-date version of `modules/asyn`

Serial devices can now be treated in much the same way as GPIB (IEEE-488) devices. Current versions of the EPICS asynDriver devGpib module contain a special 'GPIB' driver capable of communicating with devices connected to serial ports on the IOC or with devices connected through Ethernet/Serial converter boxes.

I based this tutorial on the device support I wrote for a CVI Laser Corporation AB300 filter wheel. You're almost certainly interested in controlling some other device so you won't be able to use the information directly. I chose the AB300 as the basis for this tutorial since the AB300 has a very limited command set, which keeps this document small, and yet has commands which raise many of the issues that you'll have to consider when writing support for other devices.

2 Determine the required I/O operations

The first order of business is to determine the set of operations the device will have to perform. A look at the AB300 documentation reveals that there are four commands that must be supported. Each command will be associated with an EPICS process variable (PV) whose type must be appropriate to the data transferred by the command. The AB300 commands and process variable record types I choose to associate with them are shown in table 1. There are lots of

Table 1: AB300 filter wheel commands

CVI Laser Corporation AB300 filter wheel	
Command	EPICS record type
Reset	longout
Go to new position	longout
Query position	longin
Query status	longin

other ways that the AB300 could be handled. It might be useful, for example, to treat the filter position as multi-bit binary records instead.

3 Create a new application

Now that the device operations and EPICS process variable types have been chosen it's time to create a new EPICS application to provide a place to perform subsequent software development. The easiest way to do this is with the makeBaseApp.pl script supplied with EPICS.

Here are the commands I ran. You'll have to change the `/home/EPICS/R3.14.5` to the path to where your EPICS is installed. If you're not running on Linux you'll also have to change all the `linux-x86` to reflect the architecture you're using (`solaris-sparc`, `darwin-ppc`, etc.). I built the application as a 'soft' IOC running on the host machine, but the serial 'GPIB' driver also works on RTEMS and vxWorks.

```
norume> mkdir AB300
norume> cd AB300
norume> /home/EPICS/R3.14.5/base/bin/linux-x86/makeBaseApp.pl -t ioc AB300
norume> /home/EPICS/R3.14.5/base/bin/linux-x86/makeBaseApp.pl -i -t ioc AB300
The following target architectures are available in base:
    RTEMS-pc386
    linux-x86
    solaris-sparc
    win32-x86-cygwin
    vxWorks-ppc603
What architecture do you want to use? linux-x86
```

4 Make some changes to the files in configure/

Edit the `configure/RELEASE` file which makeBaseApp.pl created and add an entry describing the path to where you installed the EPICS ASYN module:

```
ASYN=/home/EPICS/R3.14.5/modules/soft/asyn
```

Edit the `configure/CONFIG` file which makeBaseApp.pl created and specify the IOC architectures on which the application is to run. I wanted the application to run as a soft IOC, so I uncommented the `CROSS_COMPILER_TARGET_ARCHS` definition and set the definition to be empty:

```
CROSS_COMPILER_TARGET_ARCHS =
```

5 Create the device support file

The contents of the device support file provide all the details of the communication between the device and EPICS. The easiest way to create a device support file is to copy the skeleton device support file from the gpibCore module source directory to your application source directory:

```
norume> cd AB300App/src
norume> cp /usr/EPICS/R3.14.5/modules/soft/asyn/devGpib/devSkeletonGpib.c devAB300.c
```

Of course, device support for a device similar to the one you're working with provides an even easier starting point.

The remainder this section describes the changes that I made to the skeleton file in order to support the AB300 filter wheel. You'll have to modify the steps as appropriate for your device.

5.1 Declare the DSET tables provided by the device support

Since the AB300 provides only longin and longout records most of the `DSET_`*xxx* define statements can be removed. Because of the way that the device initialization is performed you must define an analog-in DSET even if the device provides no analog-in records (as is the case for the AB300).

```
#define DSET_AI    devAB300_ai
#define DSET_LI    devAB300_li
#define DSET_LO    devAB300_lo
```

5.2 Select timeout values

The default value of `TIMEWINDOW` (2 seconds) is reasonable for the AB300, but I increased the value of `TIMEOUT` to 5 seconds since the filter wheel can be slow in responding.

```
#define TIMEOUT     5.0
#define TIMEWINDOW  2.0
```

5.3 Clean up some unused values

The skeleton file provides a number of character string arrays. None are needed for the AB300 so I just removed them. Not much space would be wasted by just leaving them in place however.

5.4 Declare the command array

This is the hardest part of the job. Here's where you have to figure how to produce the command strings required to control the device and how to convert the device responses into EPICS process variable values.

Each command array entry describes the details of a single I/O operation type. The application database uses the index of the entry in the command array to provide the link between the process variable and the I/O operation to read or write that value.

The command array entries I created for the AB300 are shown below. The elements of each entry are described using the names from the GPIB documentation.

5.4.1 Command array index 0 – Device Reset

```
{&DSET_LO, GPIBWRITE, IB_Q_HIGH, NULL, "\377\377\033", 10, 10,
    NULL, 0, 0, NULL, NULL, "\033"},
```

dset This command is associated with an longout record.

type A WRITE operation is to be performed.

pri This operation should be placed on the high-priority queue of I/O requests.

cmd Because this is a GPIBWRITE operation this element is unused.

format The format string to generate the command to be sent to the device. The first two bytes are the RESET command, the third byte is the ECHO command. The AB300 sends no response to a reset command so I send the 'ECHO' to verify that the device is responding. The AB300 resets itself fast enough that it can see an echo command immediately following the reset command.

Note that the process variable value is not used (there's no printf `%` format character in the command string). The AB300 is reset whenever the EPICS record is processed.

rspLen The size of the readback buffer. Although only one readback byte is expected I allow for a few extra bytes just in case.

msgLen The size of the buffer into which the command string is placed. I allowed a little extra space in case a longer command is used some day.

convert No special conversion function is needed.

P1,P2,P3 There's no special conversion function so no arguments are needed.

pdevGpibNames There's no name table.

eos The end-of-string value used to mark the end of the readback operation.

5.4.2 Command array index 1 – Go to new filter position

```
{&DSET_LO, GPIBWRITE, IB_Q_LOW, NULL, "\017%c", 10, 10,
    NULL, 0, 0, NULL, NULL, "\030"},
```

dset This command is associated with an longout record.

type A WRITE operation is to be performed.

pri This operation should be placed on the high-priority queue of I/O requests.

cmd Because this is a GPIBWRITE operation this element is unused.

format The format string to generate the command to be sent to the device. The filter position (1-6) can be converted to the required command byte with the printf `%c` format.

rspLen The size of the readback buffer. Although only two readback bytes are expected I allow for a few extra bytes just in case.

msgLen The size of the buffer into which the command string is placed. I allowed a little extra space in case a longer command is used some day.

convert No special conversion function is needed.

P1,P2,P3 There's no special conversion function so no arguments are needed.

pdevGpibNames There's no name table.

eos The end-of-string value used to mark the end of the readback operation.

5.4.3 Command array index 2 – Query filter position

```
{&DSET_LI, GPIBREAD, IB_Q_LOW, "\035", NULL, 0, 10,
    convertPositionReply, 0, 0, NULL, NULL, "\030"},
```

dset This command is associated with an longin record.

type A READ operation is to be performed.

pri This operation should be placed on the high-priority queue of I/O requests.

cmd The command string to be sent to the device. The AB300 responds to this command by sending back three bytes: the current position, the controller status, and a terminating `'\030'`.

format Because this is a GPIBREAD operation this element is unused.

rspLen There is no command echo to be read.

msgLen The size of the buffer into which the reply string is placed. Although only three reply bytes are expected I allow for a few extra bytes just in case.

convert There's no sscanf format that can convert the reply from the AB300 so a special conversion function must be provided.

P1,P2,P3 The special conversion function requires no arguments.

pdevGpibNames There's no name table.

eos The end-of-string value used to mark the end of the read operation.

5.4.4 Command array index 3 – Query controller status

This command array entry is almost identical to the previous entry. The only change is that a different custom conversion function is used.

```
    {&DSET_LI, GPIBREAD, IB_Q_LOW, "\035", NULL, 0, 10,
        convertStatusReply, 0, 0, NULL, NULL, "\030"},
```

5.5 Write the special conversion functions

As mentioned above, special conversion functions are need to convert reply messages from the AB300 into EPICS PV values. The easiest place to put these functions is just before the `gpibCmds` table. The conversion functions are passed a pointer to the `gpibDpvt` structure and three values from the command table entry. The `gpibDpvt` structure contains a pointer to the EPICS record. The custom conversion function uses this pointer to set the record's value field.

Here are the custom conversion functions I wrote for the AB300.

```
/*
 * Custom conversion routines
 */
static int
convertPositionReply(struct gpibDpvt *pdpvt, int P1, int P2, char **P3)
{
    struct longinRecord *pli = ((struct longinRecord *)(pdpvt->precord));

    if (pdpvt->msgInputLen != 3) {
        epicsSnprintf(pdpvt->pasynUser->errorMessage,
                      pdpvt->pasynUser->errorMessageSize,
                      "Invalid reply");
        return -1;
    }
    pli->val = pdpvt->msg[0];
    return 0;
}
static int
convertStatusReply(struct gpibDpvt *pdpvt, int P1, int P2, char **P3)
{
    struct longinRecord *pli = ((struct longinRecord *)(pdpvt->precord));

    if (pdpvt->msgInputLen != 3) {
        epicsSnprintf(pdpvt->pasynUser->errorMessage,
                      pdpvt->pasynUser->errorMessageSize,
                      "Invalid reply");
        return -1;
    }
    pli->val = pdpvt->msg[1];
    return 0;
}
```

Some points of interest:

1. The custom conversion function can pass back an error message by printing it to the `errorMessage` buffer.
2. I put in a sanity check to ensure that the end-of-string character is where it should be.

5.6 Provide the device support initialization

Because of way code is stored in object libraries on different systems the device support parameter table must be initialized at run-time. The analog-in initializer is used to perform this operation. This is why all device support files must declare an analog-in DSET.

Here's the initialization for the AB300 device support. As you can see, most of the skeleton file values are left unchanged:

```
static long init_ai(int parm)
{
  if(parm==0) {
    devSupParms.name = "devAB300";
    devSupParms.gpibCmds = gpibCmds;
    devSupParms.numparams = NUMPARAMS;
    devSupParms.timeout = TIMEOUT;
    devSupParms.timeWindow = TIMEWINDOW;
    devSupParms.respond2Writes = 0;
  }
  return(0);
}
```

Three values have been changed:

1. The AB300 sends back values in response to commands, but needs no time delay, so the respond2Writes entry is set to 0.

2. The name entry is used for diagnostic purposes only.

6 Add the device support to the application

The makeBaseApp.pl script produces an application Makefile (`AB300App/src/Makefile`) with a commented-out set of application example source files. Remove the comment character and change the example names to the name of the device support file created in the previous section:

```
AB300_SRCS += devAB300.c
```

You must also link the GPIB support libraries with your application. Add the following line

```
AB300_LIBS += asyn
```

before the

```
AB300_LIBS += $(EPICS_BASE_IOC_LIBS)
```

line in the application Makefile.

7 Modify the application database defi nition fi le

Here's where you specify the link between the DSET names defined in the device support file and the DTYP fields in the application database. The `AB300App/src/AB300Include.dbd` file created by makeBaseApp.pl needs to be changed to include this information. I used `"AB300Gpib"` as the device type.

The driver support for serial line 'GPIB' devices must also be included in the application as shown.

```
include "base.dbd"

#
# Define the connection between the DTYP field name and the device DSET tables
#
device(longout, GPIB_IO, devAB300_lo, "AB300Gpib")
device(longin,  GPIB_IO, devAB300_li, "AB300Gpib")
device(ai,      GPIB_IO, devAB300_ai, "AB300Gpib")

#
# Pull in the driver support
#
include "drvGenericSerial.dbd"
```

8 Create the application database file

Now that the application includes the necessary device and driver support it's possible to create the database describing the actual EPICS process variables associated with the filter wheel.

I created the file `AB300App/Db/AB300.db` with the following contents:

```
record(longout, "$(user):FilterWheel:reset")
{
    field(DESC, "Reset AB300 Controller")
    field(SCAN, "Passive")
    field(DTYP, "AB300Gpib")
    field(OUT,  "#L0 A0 @0")
}
record(longout, "$(user):FilterWheel")
{
    field(DESC, "Set Filter Wheel Position")
    field(SCAN, "Passive")
    field(DTYP, "AB300Gpib")
    field(OUT,  "#L0 A0 @1")
    field(LOPR, 1)
    field(HOPR, 6)
}
record(longin, "$(user):FilterWheel:fbk")
{
    field(DESC, "Filter Wheel Position")
    field(SCAN, "Passive")
    field(DTYP, "AB300Gpib")
    field(INP,  "#L0 A0 @2")
    field(LOPR, 1)
    field(HOPR, 6)
}
record(longin, "$(user):FilterWheel:status")
{
    field(DESC, "Filter Wheel Status")
    field(SCAN, "Passive")
    field(DTYP, "AB300Gpib")
    field(INP,  "#L0 A0 @3")
```

```
}
```

Notes:

1. The numbers following the `L` in the INP and OUT fields are the number of the ‘link’ used to communicate with the filter wheel. This link is set up at run time by commands in the application startup script.

2. The numbers following the `A` in the INP and OUT fields are unused by the device support but must be a valid GPIB address (0-30) since the GPIB address conversion routines check the value and the diagnostic display routines require a matching value.

3. The numbers following the `@` in the INP and OUT fields are the indices into the GPIB command array.

4. The DTYP fields must match the names specified in the AB300Include.dbd database definition.

9 Add the database file to the application

The makeBaseApp.pl script put and exampl application database file into `AB300App/Db/Makefile` as a comment. Replace the example name with the name of the database file created in the previous step, leaving:

```
DB += AB300.db
```

10 Modify the application startup script

The `iocBoot/iocAB300/st.cmd` application startup script created by the makeBaseApp.pl script needs a few changes to get the application working properly.

1. Ensure that the application database records are loaded. Remove the # and give a reasonable value to the ’user’ macro:

   ```
   dbLoadRecords("../../db/AB300.db","user=AB300")
   ```

2. Set up the ’link’ between the IOC and the filter wheel.

   - If you’re using an Ethernet/RS-232 converter or a device which communicates over a telnet-style socket connection you need to specify the Internet host and port number like:

     ```
     drvGenericSerialConfigure("L0","164.54.9.91:4002",0,0)
     ```

   - If you’re using a serial line directly attached to the IOC you need something like:

     ```
     drvGenericSerialConfigure("L0","/dev/ttyS0",0,0)
     asynSetPortOption("L0", "baud", "9600")
     asynSetPortOption("L0", "bits", "8")
     asynSetPortOption("L0", "parity", "none")
     asynSetPortOption("L0", "stop", "1")
     asynSetPortOption("L0", "clocal", "Y")
     asynSetPortOption("L0", "crtscts", "N")
     ```

   - If you’re using a serial line directly attached to a vxWorks IOC you must first configure the serial port interface hardware. The following example shows the commands to configure a port on a GreenSprings UART Industry-Pack module.

```
ipacAddVIPC616_01("0x6000,B0000000")
tyGSOctalDrv(1)
tyGSOctalModuleInit("RS232", 0x80, 0, 0)
tyGSOctalDevCreate("/tyGS/0/0",0,0,1000,1000)
drvGenericSerialConfigure("L0","/tyGS/0/0",0,0)
asynSetPortOption("L0","baud","9600")
```

In all of the above examples the first argument of the drvGenericSerialConfigure and asynSetPortOption commands is the link identifier and must match the `L` value in the EPICS database record INP and OUT fields.

3. (Optional) Add lines to control the debugging level of the serial line 'GPIB' driver. The following turns on full tracing of control flow and every I/O operation which is useful during initial debugging.

```
asynSetTraceMask("L0",0,0xff)
asynSetTraceIOMask("L0",0,0x2)
```

A better way to control the amount and type of diagnostic output is through the use of the asynTrace facility.

11 Build the application

Change directories to the top-level directory of your application and:

norume> **make**

(**gmake** on solaris).
If all goes well you'll be left with an executable program in `bin/linux-x86/AB300`.

12 Run the application

Change directories to where makeBaseApp.pl put the application startup script and run the application:

```
norume> cd iocBoot/iocAB300
norume> ../../bin/linux-x86/AB300 st.cmd
dbLoadDatabase("../../dbd/AB300.dbd",0,0)
AB300_registerRecordDeviceDriver(pdbbase)
dbLoadRecords("../../db/AB300.db","user=AB300")
drvGenericSerialConfigure("L0","164.54.9.91:4002",0,0)
asynSetTraceMask("L0",0,0xff)
asynSetTraceIOMask("L0",0,0x2)
iocInit()
############################################################################
###  EPICS IOC CORE built on Jan 29 2004
###  EPICS R3.14.5 $R1-2$ $2004/02/13 20:04:36$
############################################################################
Starting iocInit
iocInit: All initialization complete
epics>
```

Check the process variable names:

```
epics> dbl
AB300:FilterWheel:fbk
AB300:FilterWheel:status
AB300:FilterWheel
AB300:FilterWheel:reset
```

Reset the filter wheel. The values sent between the IOC and the filter wheel are shown:

```
epics> dbpf AB300:FilterWheel:reset 0
DBR_LONG:           0         0x0
drvGenericSerial set eos 1 \033
drvGenericSerialWrite 3 \377\377\033
drvGenericSerialRead 1 \033
```

Read back the filter wheel position. The dbtr command prints the record before the I/O has a chance to occur:

```
epics> dbtr AB300:FilterWheel:fbk
ACKS: NO_ALARM      ACKT: YES           ADEL: 0             ALST: 0
ASG:                BKPT: 0x00          DESC: Filter Wheel Position
DISA: 0             DISP: 0             DISS: NO_ALARM      DISV: 1
DTYP: AB300Gpib     EGU:                EVNT: 0             FLNK:CONSTANT 0
HHSV: NO_ALARM      HIGH: 0             HIHI: 0             HOPR: 6
HSV: NO_ALARM       HYST: 0             INP:GPIB_IO #L0 A0 @2
LALM: 0             LCNT: 0             LLSV: NO_ALARM      LOLO: 0
LOPR: 1             LOW: 0              LSV: NO_ALARM       MDEL: 0
MLST: 0             NAME: AB300:FilterWheel:fbk             NSEV: NO_ALARM
NSTA: NO_ALARM      PACT: 1             PHAS: 0             PINI: NO
PRIO: LOW           PROC: 0             PUTF: 0             RPRO: 0
SCAN: Passive       SDIS:CONSTANT       SEVR: INVALID       SIML:CONSTANT
SIMM: NO            SIMS: NO_ALARM      SIOL:CONSTANT       STAT: UDF
SVAL: 0             TPRO: 0             TSE: 0              TSEL:CONSTANT
UDF: 1              VAL: 0
drvGenericSerial set eos 1 \030
drvGenericSerialWrite 1 \035
drvGenericSerialRead 3 \001\020\030
```

Now the process variable should have that value:

```
epics> dbpr AB300:FilterWheel:fbk
ASG:                DESC: Filter Wheel Position             DISA: 0
DISP: 0             DISV: 1             NAME: AB300:FilterWheel:fbk
SEVR: NO_ALARM      STAT: NO_ALARM      SVAL: 0             TPRO: 0
VAL: 1
```

Move the wheel to position 4:

```
epics> dbpf AB300:FilterWheel 4
DBR_LONG:           4         0x4
drvGenericSerialWrite 2 \017\004
drvGenericSerialRead 1 \020
drvGenericSerialRead 1 \030
```

Read back the position:

epics> **dbtr AB300:FilterWheel:fbk**

```
ACKS: NO_ALARM      ACKT: YES           ADEL: 0             ALST: 1
ASG:                BKPT: 0x00          DESC: Filter Wheel Position
DISA: 0             DISP: 0             DISS: NO_ALARM      DISV: 1
DTYP: AB300Gpib     EGU:                EVNT: 0             FLNK:CONSTANT 0
HHSV: NO_ALARM      HIGH: 0             HIHI: 0             HOPR: 6
HSV: NO_ALARM       HYST: 0             INP:GPIB_IO #L0 A0 @2
LALM: 1             LCNT: 0             LLSV: NO_ALARM      LOLO: 0
LOPR: 1             LOW: 0              LSV: NO_ALARM       MDEL: 0
MLST: 1             NAME: AB300:FilterWheel:fbk             NSEV: NO_ALARM
NSTA: NO_ALARM      PACT: 1             PHAS: 0             PINI: NO
PRIO: LOW           PROC: 0             PUTF: 0             RPRO: 0
SCAN: Passive       SDIS:CONSTANT       SEVR: NO_ALARM      SIML:CONSTANT
SIMM: NO            SIMS: NO_ALARM      SIOL:CONSTANT       STAT: NO_ALARM
SVAL: 0             TPRO: 0             TSE: 0              TSEL:CONSTANT
UDF: 0              VAL: 1
drvGenericSerialWrite 1 \035
drvGenericSerialRead 3 \004\020\030
```

And it really is 4:

epics> **dbpr AB300:FilterWheel:fbk**

```
ASG:                DESC: Filter Wheel Position             DISA: 0
DISP: 0             DISV: 1             NAME: AB300:FilterWheel:fbk
SEVR: NO_ALARM      STAT: NO_ALARM      SVAL: 0             TPRO: 0
VAL: 4
```

13 Device Support File

Here is the complete device support file for the AB300 filter wheel (`AB300App/src/devAB300.c`):

```
/*************************************************************************\
* Copyright (c) 2002 The University of Chicago, as Operator of Argonne
*     National Laboratory.
* Copyright (c) 2002 The Regents of the University of California, as
*     Operator of Los Alamos National Laboratory.
* EPICS BASE Versions 3.13.7
* and higher are distributed subject to a Software License Agreement found
* in file LICENSE that is included with this distribution.
\*************************************************************************/
/* devAB300.c,v 1.3 2004/01/30 17:26:32 norume Exp */
#include <epicsStdio.h>
#include <devCommonGpib.h>



/******************************************************************************
 *
 * The following define statements are used to declare the names to be used
 * for the dset tables.
 *
 * A DSET_AI entry must be declared here and referenced in an application
 * database description file even if the device provides no AI records.
 *
 ******************************************************************************/
#define DSET_AI     devAB300_ai
#define DSET_LI     devAB300_li
#define DSET_LO     devAB300_lo

#include <devGpib.h> /* must be included after DSET defines */



#define TIMEWINDOW 2.0       /* wait 2 seconds after device timeout */
#define TIMEOUT    5.0       /* I/O must complete within 5 seconds */



/*
 * Custom conversion routines
 */
static int
convertPositionReply(struct gpibDpvt *pdpvt, int P1, int P2, char **P3)
{
    struct longinRecord *pli = ((struct longinRecord *)(pdpvt->precord));

    if (pdpvt->msgInputLen != 3) {
        epicsSnprintf(pdpvt->pasynUser->errorMessage,
                      pdpvt->pasynUser->errorMessageSize,
```

```
                    "Invalid reply");
        return -1;
    }
    pli->val = pdpvt->msg[0];
    return 0;
}
static int
convertStatusReply(struct gpibDpvt *pdpvt, int P1, int P2, char **P3)
{
    struct longinRecord *pli = ((struct longinRecord *)(pdpvt->precord));

    if (pdpvt->msgInputLen != 3) {
        epicsSnprintf(pdpvt->pasynUser->errorMessage,
                      pdpvt->pasynUser->errorMessageSize,
                      "Invalid reply");
        return -1;
    }
    pli->val = pdpvt->msg[1];
    return 0;
}

/******************************************************************************
 *
 * Array of structures that define all GPIB messages
 * supported for this type of instrument.
 *
 ******************************************************************************/

static struct gpibCmd gpibCmds[] = {
    /* Param 0 -- Device Reset */
    {&DSET_LO, GPIBWRITE, IB_Q_HIGH, NULL, "\377\377\033", 10, 10,
        NULL, 0, 0, NULL, NULL, "\033"},

    /* Param 1 -- Go to new filter position */
    {&DSET_LO, GPIBWRITE, IB_Q_LOW, NULL, "\017%c", 10, 10,
        NULL, 0, 0, NULL, NULL, "\030"},

    /* Param 2 -- Query filter position */
    {&DSET_LI, GPIBREAD, IB_Q_LOW, "\035", NULL, 0, 10,
        convertPositionReply, 0, 0, NULL, NULL, "\030"},

    /* Param 3 -- Query controller status */
    {&DSET_LI, GPIBREAD, IB_Q_LOW, "\035", NULL, 0, 10,
        convertStatusReply, 0, 0, NULL, NULL, "\030"}
};

/* The following is the number of elements in the command array above.  */
#define NUMPARAMS        sizeof(gpibCmds)/sizeof(struct gpibCmd)


/******************************************************************************
```

```
 *
 * Initialize device support parameters
 *
 ******************************************************************************/
static long init_ai(int parm)
{
  if(parm==0)  {
    devSupParms.name = "devAB300";
    devSupParms.gpibCmds = gpibCmds;
    devSupParms.numparams = NUMPARAMS;
    devSupParms.timeout = TIMEOUT;
    devSupParms.timeWindow = TIMEWINDOW;
    devSupParms.respond2Writes = 0;
  }
  return(0);
}
```

14 asynTrace support

The asynTrace facility provides a convenient mechanism for controlling the diagnostic messages produced by asyn drivers. To add asynTrace to your application:

1. Add the line

   ```
   include "devAsynTrace.dbd"
   ```

   to the application *xxx*`Include.dbd` file.

2. Add the line

   ```
   DB_INSTALLS += $(ASYN)/db/asynTrace.db
   ```

   to an application `Makefile`.

3. For each port on which you wish to control diagnostic messages, add a line like

   ```
   dbLoadRecords("db/asynTrace.db","ioc=ab300Test,port=L0,addr=0")
   ```

   to the application startup (`st.cmd`) script. The `port` value must match the the value in the corresponding `drvGenericSerialConfigure` command. The `addr` value should be zero. The `ioc` value is arbitrary and should be chosen so that the ioc/port/addr combination is unique among all IOCs on your network.

To run the asynTrace screen, add *<asynTop>*`/medm` to your `EPICS_DISPLAY_PATH` environment variable and start medm with `ioc`, `port` and `addr` values matching those given in the `dbLoadRecords` command:

```
medm -x -macro "ioc=ab300Test,port=L0,addr=0" asynTrace.adl &
```