Created:February 4, 1997
Modified:January 13, 1998

Gemini
Controls
Group
Report

Writing EPICS Device

EPICS Input/Output Controller (IOC) Application Developer's Guide, Martin R. Kraimer, Argonne National Laboratory, November 1994.

2. *EPICS Input/Output Controller (IOC) Record Reference Manual*, Janet B. Anderson, Martin R. Kraimer, Argonne National Laboratory, December 1, 1994.
3. *CapFast Electronic Circuit Design Guide CAE User's Guide*, Phase Three Logic, Inc., 1991.

1.4 Abbreviations

EPICS	Experimental Physics and Industrial Control System.
IOC	Input/Output Controller.
PV	Process Variable.

1.5 Glossary

CapFast	
Device Support	
EPICS	A Real-time software package.
Record Support	
VxWorks	A real-time operating system.

1.6 Stylistic Conventions

1.6.1 References to other documents

References to other documents are given using a number in square brackets, for example: [1]. The reference can be found in Section 1.3 on page 1.

1.6.2 References to EPICS records, files, and PVs

References to EPICS records are given in italics, for example: *stringin*. References to files and other EPICS items are give in small type, for example: devSiHostId.c or INST_IO.

1.6.3 Typographic conventions

User input is written in bold courier, for example: **dbtr**. Output or code is written in regular courier font, for example: rcmd().

1.7 Revisions

1. Version 01, January 13, 1998, Document created.

networking in creating the example. However, we don't want to invest a major effort in creating the example, especially if we need to write network support on the host machine. Consequently, our example uses an existing network protocol, rsh, which is supported bon the host machines as a server daemon, and on the VxWorks machines as the client call rcmd().

The information we wish to transport across the network should be simple. We could read the files in a directory with **ls**, or the users with **who**, but both of these would require sophisticated (at least in EPICS) mechanisms for storing the return data. Instead, we will execute the command **hostid** and read the result back to the EPICS database. The result is small enough to fit into an EPICS *stringin* record or can be converted to a number and stored into an EPICS *longin* record. For this example we will write device support for the first record.

```
“stringin”    INST_IO      “devSiHostId”        “Host ID”
```

The first argument is the record which this device support affects. The second is the link type of the device support. See [1] for a link of link types. The third argument is the name of the device support structure, used in the source code. The final argument is the name of the device support, used by the record in the DTYP field.

3.2 src

There are two directories that source files can be located: src and sys. The only difference between them is the way they load the object modules. Files in src are installed as discrete object modules, while files in sys are loaded into one object module named sys.o and installed. For our example we will use the src directory.

3.2.1 src/devSiHostId.c

Our device support file will be named devSiHostId.c. Conventionally, we begin device support files with the “dev” prefix, followed by the record type (in this case stringin or “Si”), then the name of the device we are supporting. Since we are writing device support to read hostid information, we use the designator “HostId”.

There are three important parts of this device support code. They are

- device support structure,
- device support record initialization,
- device support read.

The device support structure defines the names of the routines to be executed by the record processing. Each record has a well-defined ordering of these routines which should be strictly adhered to. In the case of the *stringin* record the order is report, initialization, per-record initialization, I/O interrupt, and read functions. The name of the device support structure should be the same as the third argument in the file ascii/cat_ascii/devSup.ascii.

In our example we only support record initialization and read functions. For a more advanced description of the other functions see reference [1]. For a description of a record's device support options, see reference [2].

The device support record initialization should perform operations required only upon the initialization of the record. In our case we do the following:

- check the device support link type is INST_IO.
- create a private data storage area for this record.
- parse the input link value into the private area.
- test the validity of the input.

The device support read function is executed during record processing. In this example, the following actions occur:

- find the private record information (host name and user name),
- send the hostid command to the remote host,
- read the hostid into the VAL field.

Upon completing of record processing, the string returned by the Unix hostid command resides in the record's VAL field.

```
#include “vxWorks.h”
#include “remLib.h”
#include “ioLib.h”
#include “stdlib.h”

#include “dbDefs.h”
#include “dbAccess.h”
#include “recSup.h”
#include “devSup.h”
#include “stringinRecord.h”

#define  RSHD    514


static long   init_si (struct stringinRecord *);
static long   read_si (struct stringinRecord *);

struct
{
   long      number;
   DEVSUPFUN report;
   DEVSUPFUN init;
```

```
    DEVSUPFUN init_record;
    DEVSUPFUN get_ioint_info;
    DEVSUPFUN read_stringin;
} devSiHostId =
{
    5,
    NULL,
    NULL,
    init_si,
    NULL,
    read_si
};

typedef struct
{
    char host[40];
    char user[9];
} SIHOSTID_INFO;


static long init_si (
    struct stringinRecord *psi)
{
    struct link *plink = &psi->inp;
    SIHOSTID_INFO *pinfo;
    int n;

    if (plink->type != INST_IO)
    {
        recGblRecordError (S_db_badField,(void *) psi,
            “devSiHostId (init_record) Illegal INP field”);
        return S_db_badField;
    }

    /*
     * create some private data
     */
    pinfo = (SIHOSTID_INFO *) malloc (sizeof (SIHOSTID_INFO));
    n = sscanf (plink->value.instio.string, “%s %s”, pinfo->host, pinfo->user);
    if (n != 2)
    {
        recGblRecordError (S_db_badField,(void *) psi,
            “devSiHostId (init_record) Bad INP field value”);
        return ERROR;
    }
    psi->dpvt = (char *) pinfo;

    /*
     * we ought to test if the host and user can be accessed
     */

    psi->udf = FALSE;
    return OK;
}


static long read_si (
    struct stringinRecord *psi)
{
    SIHOSTID_INFO *pinfo;
    int fd;

    /*
     * find the private record information (host name and user name)
     */
    pinfo = (SIHOSTID_INFO *) psi->dpvt;

    /*
     * send the hostid command to the remote host
     */
    fd = rcmd (pinfo->host, RSHD, pinfo->user, pinfo->user,
       "/usr/bin/hostid", NULL);
    if (fd == ERROR)
    {
       /* need to set an error here */
       return ERROR;
    }

    /*
     * read the hostid into the VAL field.
     */
    if (read (fd, psi->val, 39) <= 0)
    {
       /* need to set an error here */
       close (fd);
       return ERROR;
    }
    psi->val[strlen (psi->val)-1] = NULL;

    close (fd);

    return OK;
}
```

3.3 capfast

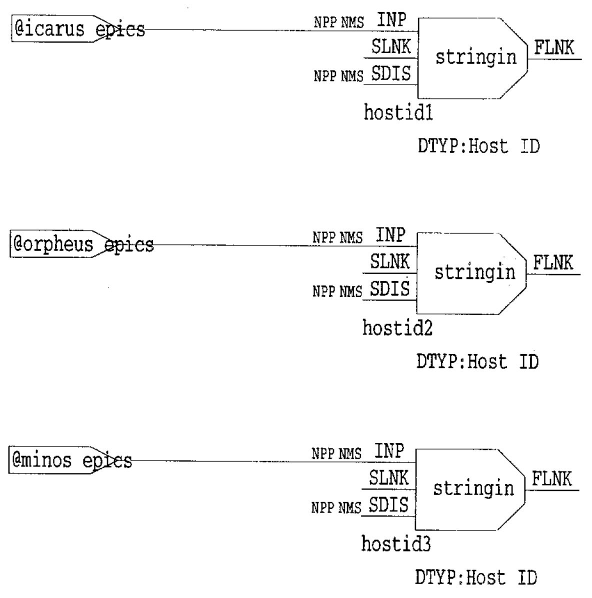
The capfast directory contains the graphical representation of the EPICS database for the application.

3.3.1 capfast/hostid.sch

In our example, we need to create some number of *stringin* records with hardware inputs attached to their INP fields. The stringin record should have the DTYP field changed to "Host ID". This type should match the fourth field of the ascii/cat_ascii/devSup.ascii file entry for the device type.

The hardware input should have a string with two words, the host name and the user name. The string should also begin with the '@' character to designate that this string is of link type INST_IO. This last step is perhaps the most obfuscated of the entire process of creating device support. Designating the type of link for the device support is not transparent, nor is it selectable through a menu. The description of the mechanisms used to select the link type won't be covered here; suffice it to say that INST_IO support begins with '@', CONSTANT support begins with a number (or sign), and any other string will be converted to type PV_LINK or DB_LINK. See reference [1] for more information.

The hostid.sch database contains three *stringin* records which read the hostid strings from three different computers. Network access to these computers will succeed only if they allow rsh commands from the processor running the EPICS database and have a user account named “epics”.



The startup directory contains the files used by VxWorks during boot time. In particular, the file startup.vws contains the VxWorks startup script identified in the boot EEPROM data.

3.4.1 startup/startup.vws

The UAE default version of this file is used as a template. There are three things modified in this file:

- a list of host names has been added,
- the device support object file is loaded,
- the hostid database is loaded.

The addition of the host names is required because we will be asking the VxWorks network task to connect to the hosts. The O/S requires the additions to its static routing table. Changes to the default template appear in *italics* in Figure 4.

[illegible]

```
cd “$(install)”
pwd

# The local file contains site-specific information
# such as the default router and the vxWorks password file.

< $(version)/bin/$(arch)/local

# For the hostid application, add some static routes
# hostAdd (“icarus”, “140.252.15.210”)
hostAdd (“orpheus”, “140.252.15.57”)
hostAdd (“minos”, “140.252.15.71”)

# Load EPICS core and record stuff
ld < $(epics)/base/bin/$(arch)/iocCore
ld < $(epics)/base/bin/$(arch)/drvSup
ld < $(epics)/base/bin/$(arch)/recSup
ld < $(epics)/base/bin/$(arch)/devSup

# If using state programs, load the sequencer
# ld < $(epics)/base/bin/$(arch)/seq

# Load any application records, device support or drivers here,
# or code for subroutine records
ld < $(version)/bin/$(arch)/devSiHostId.o

# Load sequences here
# ld < $(version)/bin/$(arch)/testseq.o

# Configure the Timestamp system, 1=Master IOC
TSconfigure 1, 0, 0, 0, 0, 0

# Load one or more ASCII databases here
dbLoad “$(version)/data/$(dctsdr).dctsdr”
dbLoadRecords “$(version)/data/hostid.db”

# This turns off logging
# iocLogDisable=1

iocInit “$(version)/data/resource.def”

# Start sequences
# seq &testseq
```

, or at the boot prompt with the ‘c’ command. The new startup script name is the absolute path name of the UAE application directory plus the string **bin/mv167/startup.**

[illegible]

```
‘.’ = clear field;  ‘-’ = go to previous field;  ^D = quit

boot device          : ei
processor number     : 0
host name            : icarus
file name            : /opt/local/vxworks/config/mv167/vxWorks
inet on ethernet (e) : 140.252.15.82:ffffff00
inet on backplane (b):
host inet (h)        : 140.252.15.210
gateway inet (g)     :
user (u)             : epics
ftp password (pw) (blank = use rsh):
flags (f)            : 0x8
target name (tn)     : sim1ioc
startup script (s)   : /scr3/goodrich/epics/tst3/bin/mv167/startup
other (o)            :
```

After this information has been entered, the VxWorks computer can be rebooted. If everything went well, no error message should appear.

5.1 Target command line tests

We can determine the names of the records we have loaded by using the command **dbl.** This returns the following:

dbpr “hostid:hostid1”,4

```
-> dbpr “hostid:hostid1”,4
ACKS: NO_ALARM      ACKT: YES          ASG :              ASP : 00000000
BKPT: 00           DESC: string input record             DISA: 0
DISP: 0            DISS: NO_ALARM      DISV: 1            DPVT: 00edea48
DSET: 00ef7544      DTYP: Host ID       EVNT: 0            FLNK: 0
INP : INSTIO  parm=icarus epics        LCNT: 0            LSET: 1
MLIS: 000000000000000000000000         MLOK: 00edeacc00000000
NAME: hostid:hostid1                   NSEV: NO_ALARM      NSTA:
OVAL:              PACT: 0            PHAS: 0            PINI: NO
PPN : 00000000      PPNN: 00000000      PRIO: LOW          PROC: 0
PUTF: 0            RPRO: 0            RSET: 00e26ce8      SCAN: Passive
SDIS: 0            SEVR: INVALID       SIML: 0            SIMM: NO
SIMS: NO_ALARM      SIOL: 0            SPVT: 00000000      STAT: UDF
SVAL:              TIME: 0000000000000000                 TPRO: 0
TSE : 0            TSEL: 0            UDF : 0            VAL : string
```

Notice the INP field is correctly set to INST_IO and has the parameters read from the Capfast hardware input symbol.

We also can test the execution of the record with the **dbtr** command:

dbtr “hostid:hostid1”

```
-> dbtr “hostid:hostid1”
ACKS: NO_ALARM      ACKT: YES          ASG :              BKPT: 00
DESC: string input record             DISA: 0            DISP: 0
DISS: NO_ALARM      DISV: 1            DTYP: Host ID       EVNT: 0
FLNK: 0            INP : INSTIO  parm=icarus epics        LCNT: 0
NAME: hostid:hostid1                   NSEV: NO_ALARM      NSTA:
OVAL: 727350b8      PACT: 0            PHAS: 0            PINI: NO
PRIO: LOW          PROC: 0            PUTF: 0            RPRO: 0
SCAN: Passive       SDIS: 0            SEVR: NO_ALARM      SIML: 0
SIMM: NO           SIMS: NO_ALARM      SIOL: 0            STAT:
SVAL:              TPRO: 0            TSE : 0            TSEL: 0
UDF : 0            VAL : 727350b8
```

Now the VAL field contains the host ID string from the machine *icarus*.

5.2 Host command line tests

5.3 Edd/dm