NSRL
EPICS
Seminar

EPICS

Writing Channel Access Clients

Kazuro Furukawa, KEK, (2000-2004)

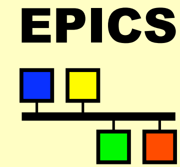
<kazuro.furukawa @ kek.jp>

(Marty Kraimer, APS, USPAS1999)

(Bob Dalesio, LANL, USPAS2003)

NSRL
EPICS
Seminar

References



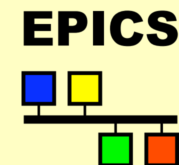
- EPICS R3.12 / R3.14 Channel Access Reference Manual are the Detailed Documents

- cadef.h caerr.h
  - Description of Basic Channel Access Interface

- db_access.h
  - Definition of data, difficult to understand but important to write robust software

- Tutorials on LANL Web Pages
  - Links from EPICS Web Pages

CA between IOC and OPI



WORKSTATION

user program

Channel Access Repeater

Channel Access client

Channel Access server

Channel Access client

C program

database access library

SNL sequence

DATABASE

database library

record support

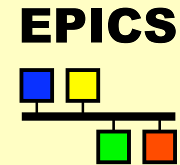
device support

device drivers

IOC

instrumentation and control hardware

Overview of Talk



- Introduction of Channel Access Client Software

  Demonstrate Simplified Usage of CA API/Macro

  Demonstrate Flavors of CA Callback

  (without details of db_access.h)

  SEVCHK

  ```
  SEVCHK(<function call>,"message")
  ```

  Macro to test the return code, it display message and abort, on error

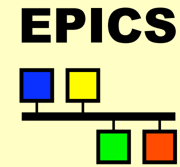
  Used in sample software, convenient in test software

  Should not be used in production applications

- CA Client Library slightly Changed between EPICS 3.13 and 3.14
  - In this talk, example codes are based on EPICS 3.13, And they can be used still on EPICS 3.14.5.
  - There exists newer API to support more functionalities, eg. ca_context_create() instead of ca_task_initialize(). Please read the documentation, or look in example codes.
  - On the other hand, it is assumed to build them on EPICS 3.14 in order to test them on Linux easily. On 3.13, you may need slightly different commands to build them.

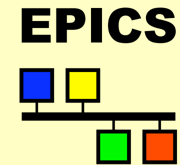
NSRL EPICS Seminar

Very Simple Example




```
/*caSimpleExample.c*/
#include <stddef.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "cadef.h"
main(int argc,char **argv)
{
    double      data;
    chid mychid;
    if(argc != 2) {
        fprintf(stderr,"usage: caExample pvname\n");
        exit(1);
    }
    SEVCHK(ca_task_initialize(),"ca_task_initialize");
    SEVCHK(ca_search(argv[1],&mychid),"ca_search failure");
    SEVCHK(ca_pend_io(5.0),"ca_pend_io failure");
    SEVCHK(ca_get(DBR_DOUBLE,mychid,(void *)&data),"ca_get failure");
    SEVCHK(ca_pend_io(5.0),"ca_pend_io failure");
    printf("%s %f\n",argv[1],data);
    return(0);
}
```

caExample

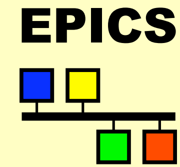



```
/*from stdin read list of PVs to monitor*/
#include <stddef.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <cadef.h>
#define MAX_PV 1000
#define MAX_PV_NAME_LEN 40
typedef struct{
    char          value[20];
    chid          mychid;
    evid          myevid;
} MYNODE;
```

- Declarations/Definitions for Channel Access
- A piece of software to accept a list of Process Variable (Record) names from stdin, then to process them

NSRL EPICS Seminar

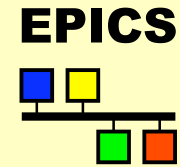
CA macros




```
static void printChidInfo(chid chid, char *message)
{
    printf("\n%s\n",message);
    printf("pv: %s  type(%d) nelements(%d) host(%s)",
        ca_name(chid),ca_field_type(chid),
        ca_element_count(chid),
        ca_host_name(chid));
    printf(" read(%d) write(%d) state(%d)\n",
        ca_read_access(chid),ca_write_access(chid),
        ca_state(chid));
}
```

- Various information available for given chid (Channel ID)
  - ca_name - name
  - ca_field_type - type as defined in db_access.h
  - ca_element_count - array size (1 for scalars)
  - ca_host_name - INET name of host
  - ca_read_access - Is read access allowed
  - ca_write_access - Is write access allowed
  - ca_state - connected, not connected, etc.

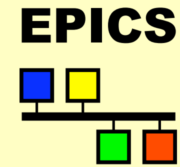
exception/connection callbacks




```
static void exceptionCallback(
   struct exception_handler_args args)
{
    chid chid = args.chid;
    MYNODE       *pnode = (MYNODE *)ca_puser(chid);
    long type = args.type;/*type of value returned*/
    long count = args.count;
    long stat = args.stat;
    printChidInfo(chid,"exceptionCallback");
    printf("type(%d) count(%d) stat(%d)\n",type,count,stat);
}
static void connectionCallback(struct connection_handler_args args)
{
    chid chid = args.chid;
    MYNODE       *pnode = (MYNODE *)ca_puser(chid);
    printChidInfo(chid,"connectionCallback");
}
```

- exceptionCallback
  - Called on events other than following callbacks
  - Errors detected at IOC, etc.
- connectionCallback
  - Called on each connect/disconnect

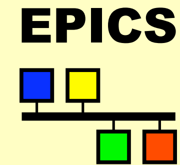
accessRightsCallback




```
static void accessRightsCallback(
   struct access_rights_handler_args args)
{
    chid      chid = args.chid;
    MYNODE    *pnode = (MYNODE *)ca_puser(chid);
    printChidInfo(chid,"accessRightsCallback");
}
```

- Called on connect
- Called whenever access rights change

eventCallback

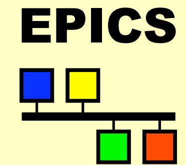



```
static void eventCallback(
  struct event_handler_args eha)
{
    chid    chid = eha.chid;
    MYNODE  *pnode = (MYNODE *)ca_puser(chid);
    long    type = eha.type;
    long    count = eha.count;
    if(eha.status!=ECA_NORMAL) {
      printChidInfo(chid,"eventCallback");
    } else {
      char *pdata = (char *)eha.dbr;
      printf("Event Callback: %s = %s\n",
        ca_name(eha.chid),pdata);
    }
}
```

- Called on Monitored event

NSRL EPICS Seminar

main - start



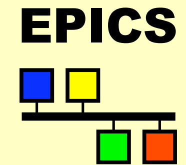

```
main()
{
    int         npv = 0;
    MYNODE      *pnode;
    MYNODE      *pmynode[MAX_PV];
    char        *pname[MAX_PV];
    int         i, status;
    char        tempStr[MAX_PV_NAME_LEN];
    char        *pstr;
    while(1) {
      if(npv >= MAX_PV ) break;
      pstr = fgets(tempStr,MAX_PV_NAME_LEN,stdin);
      if(!pstr) break;
      if(strlen(pstr) <=1) continue;
      pstr[strlen(pstr)-1] = '\0'; /*strip off newline*/
      pname[npv] = calloc(1,strlen(pstr) + 1);
      strcpy(pname[npv],pstr);
      pmynode[npv] = (MYNODE *)calloc(1,sizeof(MYNODE));
      npv++;
    }
```

- Read a list of Process Variable (Record) Names

  caExample < file

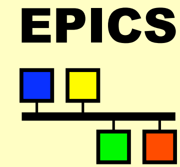
NSRL
EPICS
Seminar

Actual CA calls




```
    SEVCHK(ca_task_initialize(),
        "ca_task_initialize");
    SEVCHK(ca_add_exception_event(
        exceptionCallback,NULL),
        "ca_add_exception_event");
    for(i=0; i<npv; i++) {
       SEVCHK(ca_search_and_connect(
        pname[i],&pmynode[i]->mychid,
        connectionCallback,&pmynode[i]),
        "ca_search_and_connect");
       SEVCHK(ca_replace_access_rights_event(
        pmynode[i]->mychid,
        accessRightsCallback),
        "ca_replace_access_rights_event");
       SEVCHK(ca_add_event(DBR_STRING,
        pmynode[i]->mychid,eventCallback,
        pmynode[i],&pmynode[i]->myevid),
        "ca_add_event");
     }
     SEVCHK(ca_pend_event(0.0),"ca_pend_event");
     /* does not reach here */
     ca_task_exit();
}
```

Start and End



- **ca_task_initialize**

  initialize the interface to ca_repeater, etc
  (connection management)
  ca_context_create() is recommended in 3.14

- **ca_add_exception_event**

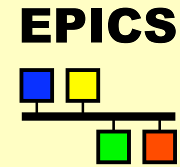
  Specify a Callback routine for an anomaly in CA

- {Body of the Code}

- **ca_task_exit**

  Release resources allocated to CA
  ca_context_destroy() is recommended in 3.14.

Search



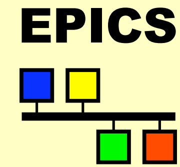
- `ca_search_and_connect(name,pchid, connectionCallback,userarg)`
  `ca_replace_access_rights_event(chid, accessRightsCallback)`
  - Requests are buffered till the buffer is full, or ca_pend or ca_flush are called
  - Search Process Variable via UDP broadcast
    - (if `EPICS_CA_AUTO_ADDR_LIST` is `YES` or not set)
    - An IOC in the subnet responds, if it contains the Process Variable
  - Establish a TCP connection with the IOC which contains the Process Variable
  - connectionCallback is called when the connection status changes
  - accessRightsCallback is called when the Access Right changes
  - Given a chid we can always retrieve userarg
  - **`ca_create_channel()`** is recommended in 3.14.

I/O

- Puts - Many flavors
  - `ca_array_put(type,count,chid,pvalues)`
    ```
    ...
    ca_flush_io()
    ```
    Calls are buffered until: buffer full, ca_flush, or ca_pend.
  - `ca_put_callback(type,count,chid, pvalue,putCallback,userarg)`
    putCallback called after all records processed because of put complete processing.

- Gets - Many flavors
  - `ca_array_get(type,count,chid,pvalues)`
    ```
    ...
    ca_pend_io(timeout)
    ```
  - `ca_array_get_callback(type,count,chid,getCallback,userarg)`
    ```
    ...
    ca_pend_event(timeout)
    ```

I/O continued



- Monitors - Many flavors
  - `ca_add_masked_array_event(type,count, chid,eventCallback,userarg, pevid,mask)`

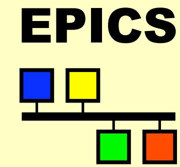
    Call this once for each channel to be monitored.
    Mask allows value changes, alarm changes, archival changes

    ...
  - `ca_pend_event(timeout)`

    Waits at least timeout seconds

NSRL
EPICS
Seminar

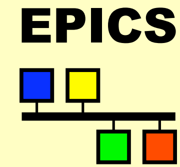
Waiting



- **`ca_flush_io()`**
  Normally called by ca_pend routines
  Sends any udp/tcp buffers that are not empty
- **`ca_pend_io(timeout)`**
  Calls ca_flush_io. Waits until timeout OR until all outstanding ca_gets complete.
  Also waits until ca_search with no callback are satisfied
- **`ca_pend_event(timeout)`**
  Processes incoming events for at least timeout seconds
- timeout
  - 0 means wait forever
  - Short time, e.g. .0001 means poll

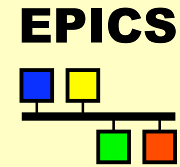
NSRL
EPICS
Seminar

CA with X



- Channel Access uses select() to wait.
- File Descriptor Manager can be used.
- Channel access provides ca_add_fd_registration
- X provides similar facility

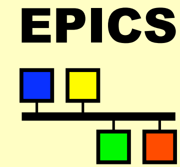
db_access.h



- Describes the data CA can transfer
- Hard to understand and use
- Provides access to
  - data types:
    string, char, short, long, float, double
  - status, severity, time stamp
  - arrays
  - enums (in ioc both menus and DBF_ENUM fields)
  - complete set of enum choices
  - control, display, alarm limits
  - Alarm Acknowledgment

NSRL EPICS Seminar

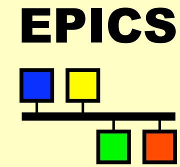
ezCa - Easy Channel Access



- Goals
  - Easy to use.
  - Provide non-callback synchronous model.
- Data Types
  - **ezcaByte, ezcaString, ezcaShort, ezcaLong, ezcaFloat, ezcaDouble**
- Basic Calls
  - **int ezcaGet(pvname, type, nelem, buff)**
  - **int ezcaPut(pvname, type, nelem, buff)**
  - **int ezcaGetWithStatus(pvname,type, nelem,buff,time,stat,sevr)**
- Synchronous Groups
  - **int ezcaStartGroup(void)**
  - **any combination of get and put**
  - **int ezcaEndGroup(void)**

NSRL
EPICS
Seminar

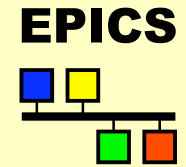
ezCa continued



- Error Handling
  - `ezcaPerror(message)`
  - `ezcaGetErrorString(message,errorstring)`
  - `ezcaFreeErrorString(errorstring)`
- Other Groupable Functions
  - `int ezcaGetControlLimits(pvname,type,low,high)`
  - `int ezcaGetGraphicLimits(pvname,type,low,high)`
  - `int ezcaGetNelem(pvname,nelem)`
  - `int ezcaGetPrecision(pvname,precision)`
  - `int ezcaGetStatus(pvname,time,stat,sevr)`
  - `int ezcaGetUnits(pvname,units)`
- Monitor Functions
  - `int ezcaSetMonitor(pvname,type)`
  - `int ezcaClearMonitor(pvname,type)`
  - `int ezcaNewMonitor(pvname,type)`
- Others

NSRL
EPICS
Seminar

Starting Your Practice

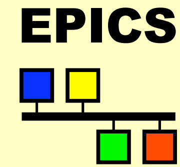


- **mkdir mytest**
- **cd mytest**
- **setenv EPICS_HOST_ARCH \ `$EPICS_BASE/startup/EpicsHostArch.pl`**
  - **(HOST_ARCH=`$EPICS_BASE/startup/EpicsHostArch.pl`; export HOST_ARCH)**
- **setenv PATH ${PATH}:$EPICS_BASE/bin/$EPICS_HOST_ARCH**
- **(USER=`whoami` ; export USER)**
- **makeBaseApp.pl -t caClient test1**
- **cd test1App**
- *create source code, or use examples there*
- *edit Makefile if necessary*
- **cd ..**
- **make (gmake)**

- gmake caExample
- caExample ffred

NSRL
EPICS
Seminar

Practice Explanation 1




- ```
  EPICS_HOST_ARCH=\
  `$EPICS_BASE/startup/EpicsHostArch.pl`
  export EPICS_HOST_ARCH
  ```

  assigning a platform name for EPICS software

  (backquotes around "$EPICS … HostArch.pl" mean "execute it and use the result")

- ```
  USER=`whoami` ; export USER
  ```

  (you don't need this, since normal Unix initializes this variable automatically)

  assigning a user name for EPICS software

- ```
  mkdir mytest ; cd mytest
  ```

  making directory for your test, and going into it

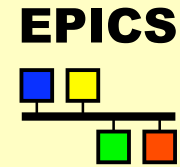
- ```
  makeBaseApp.pl -t caClient test1
  ```

  creating environment (directory and config files)
  for a new EPICS application

  see the manual "EPICS IOC Applications Developers Guide"

NSRL
EPICS
Seminar

Practice Explanation 2



- **`cd test1App`**

  Create a Simple sample code or use a sample code there

- **`make (gmake)`**

  build the application with gnu-make based on the Makefile made by makeBaseApp.pl

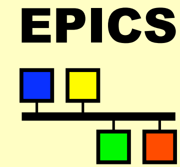
- *If you wrote a code and if you don't want to modify Makefile*
- **`cd O.linux ; make xxxx`**

  if you modifed Makefile, you don't need this step

- **`./xxxx`**

  Execute the application

Data Type Conversions in Channel Access



DBR

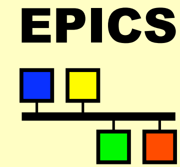
_STRING, _DOUBLE, _FLOAT, _LONG, _CHAR, _ENUM

Data type conversions are performed in the server

Endian and floating point conversions are done in the client

Polite clients requests data in native type and perform necessary conversion on the client side

Accessing Composite Data Structures

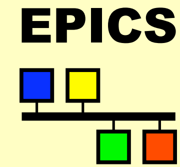


Many fields are fetched from the data store in one access:

```
struct dbr_ctrl_float       data;
struct dbr_ctrl_float       *pdata = &data;
ca_get(DBR_CTRL_FLOAT,mychid,(void *)pdata);
printf("%d %d\n",pdata->status, pdata->severity);
printf("%d %d\n",pdata->stamp.secPastEpoch, pdata->stamp.nsec);
printf("%f %f\n",pdata->high_display_limit,pdata->low_display_limit);
printf("%f %f\n",pdata->high_warning_limit,pdata->low_warning_limit);
printf("%f %f\n",pdata->high_alarm_limit,pdata->low_alarm_limit);
printf("%f %f\n",pdata->high_control_limit,pdata->low_control_limit);
printf("%f %s\n",pdata->value, pdata->units);
```

*Refer to db_access.h for structures...

Error Checking



- Error codes and error related macros are in caerr.h
- SEVCHK will exit on errors it deems irrecoverable
- ECA_NORMAL means the exchange was initiated successfully
- SEVCHK exit behavior can be replaced with your own exception handler

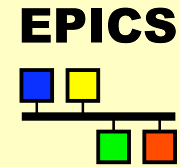
  ca_add_exception_event(…..)
- example:

  status = ca_array_put(data_type,channel_id,pvalue);

  SEVCHK(status,"additional info in error message");

NSRL
EPICS
Seminar

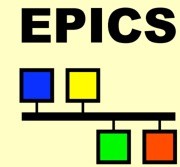
Caching vs. Queuing



- An event handler can either take its actions in the event handler -
  - queuing
  - all data is handled
  - degradation mode is longer delays
- Place data into an intermediate buffer and have an alternate thread handle the data
  - caching
  - data can be overwritten
  - degradation mode is intermediate data is discarded
- note that buffer in IOC will overwrite the last monitor on the queue when a buffer overflows in the IOC

NSRL EPICS Seminar

Channel Access Notes



- ca_repeater needs to be run once on each workstation
- in the database,
  - a deadband of 0 posts a monitor on any change
  - a deadband of -1 posts monitors on every scan
- R3.15 is hoped to provide an ability to limit the monitor event rates
- read cadef.h, caerr.h and db_access.h before writing a channel access client
- it is most efficient to use native data types and handle data conversions in the client program