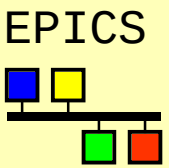
IHEP

EPICS

Seminar

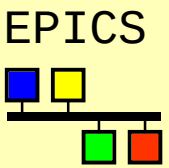


Writing Channel Access Clients

Kazuro Furukawa, KEK

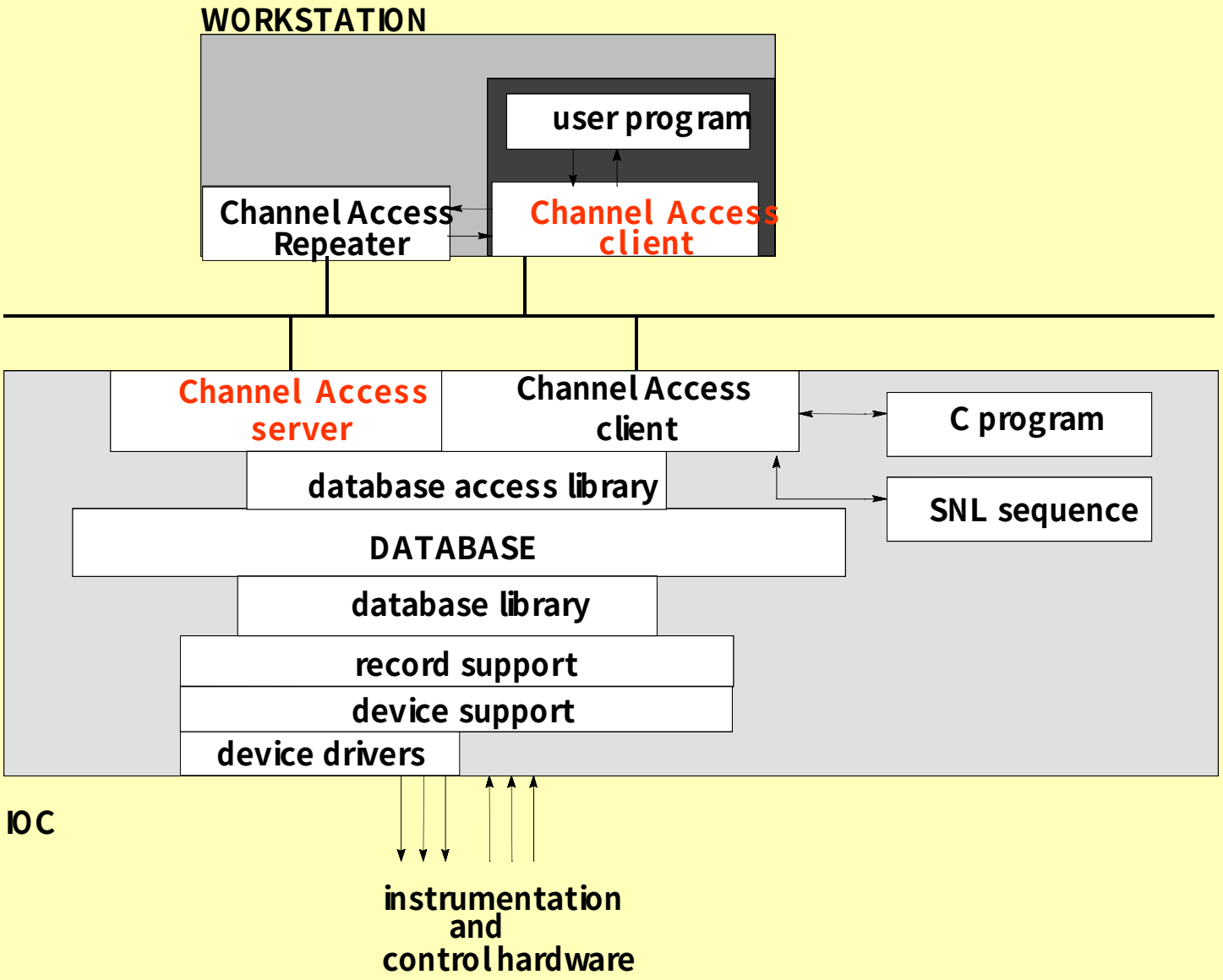
(Marty Kraimer, APS)

References

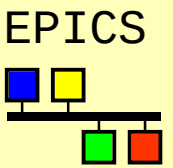


- EPICS R3.12 Channel Access Reference Manual
  Detailed reference manual
- cadef.h caerr.h - The CA interface
- db_access.h
  Horid mess?
  MUST understand before writing robust client
  Has macros to suppress “run on” code
- Tutorial Document at LANL

CA between IOC and OPI



Overview of Talk

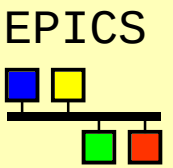


- Explain example client
  - Demonstrates CA macros
  - Demonstrates many flavors of callbacks
  - Does NOT explain details about db_access.h
- SEVCHK

  `SEVCHK(<function call>, message )`

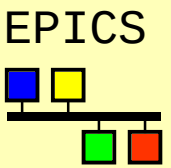
  - Macro that checks return codes
  - If error, displays message and aborts
  - Used in example
  - DONT use for robust client
  - Other macros for robust clients

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;
```

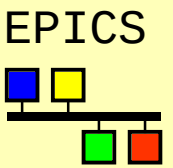
- example specific definitions
- Accepts list of PVNAMES from stdin

IHEP

EPICS

Seminar

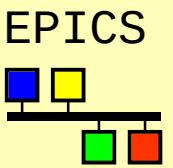
CA macros




```
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));
}
```

- Given a chid (Channel ID) the following available
  - 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.

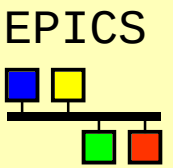
exception/connection callbacks




```
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
  - Events with no other callback
  - Errors detected in ioc
- connectionCallback
  - Each connect/disconnect

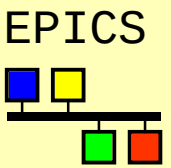
accessRightsCallback




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

- On connect
- Whenever access rights change

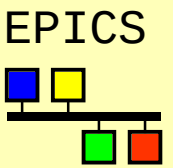
eventCallback




```
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);
    }
}
```

- monitored events

main - start

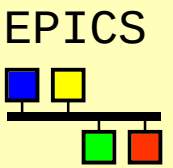



```
main()
{
    int          npv = 0;
    MYNODE       *pnode;
    MYNODE       *pmynode[MAX_PV];
    char *pname[MAX_PV];
    int          status;
    int          i;
    char tempStr[MAX_PV_NAME_LEN];
    char *pstr;
    while(1) {
       if(npv >= MAX_PV ) break;
       pstr = fgets(tempStr,MAX_PV_NAME_LEN,stdin);
       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++;
    }
```

- Reads list of PVs from stdin

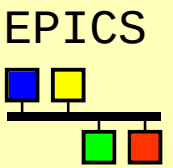
  caExample < file

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");
     }
     /*Should never return from following call*/
     SEVCHK(ca_pend_event(0.0),"ca_pend_event");
     ca_task_exit();
}
```

Start and End



- `ca_task_initialize`

  Interface with ca_repeater
  (connection management)

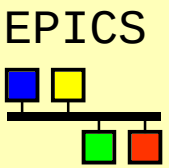
- `ca_add_exception_event`

  Specifies a callback routine that is called when ca detects problems

- {Other Code}
- `ca_task_exit`

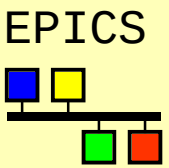
  Release all resources allocated by CA.

Search



- `ca_search_and_connect(name,pchid, connectionCallback,userarg)`
  `ca_replace_access_rights_event(chid, accessRightsCallback)`
  - Calls buffered until buffer full or ca_pend or ca_flush.
  - UDP broadcast
    - Each IOC on subnet receives each udp packet
  - TCP connection established to each IOC that has any channels
  - connectionCallback called whenever connection status changes.
  - accessRightsCallback called whenever access rights changes.
  - Given a chid you can always retrieve userarg.

I/O

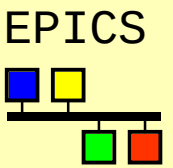


- Puts - Many flavors
  - `ca_array_put(type,count,chid,pvalues)`
    `...`
    `ca_flush_io()`

    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,user arg)`
    `...`
    `ca_pend_event(timeout)`

I/O continued

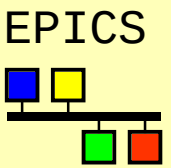


- Monitors - Many Flavors
  - `ca_add_masked_array_event(type,count, chid,eventCallback,userarg, pevid,mask)`

    Call this once for each channel to be monitored.
    Mask allows value changes, alarm changes, archival changes
  - ...
  - `ca_pend_event(timeout)`

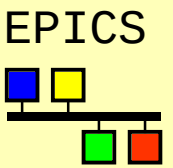
    Waits at least timeout seconds

Waiting



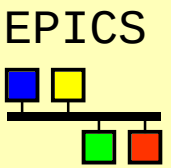
- `ca_flush_io()`
  Normally called by ca_pend routines
  Sends any udp/tcp buffers that are not empty
- `ca_pend_io(timeout)`
  Calls ca_flush_io. Waits until timeout OR until all outstanding ca_gets complete.
  Also waits until ca_search with no callback are satisfied.
- `ca_pend_event(timeout)`
 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

CA with X



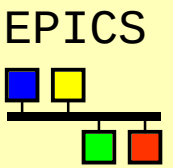
- Channel Access uses select() to wait.
- File Descriptor Manager can be used.
- Channel access provides ca_add_fd_registration
- X provides similar facility

db_access.h



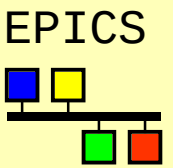
- Describes the data CA can transfer
- Hard to understand and use
- Provides access to
  - data types:
    string, char, short, long, float, double
  - status, severity, time stamp
  - arrays
  - enums (in ioc both menus and DBF_ENUM fields)
  - complete set of enum choices
  - control, display, alarm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

ezCa - Easy Channel Access



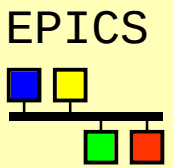
- Goals
  - Easy to use.
  - Provide non-callback synchronous model.
- Data Types
  - `ezcaByte, ezcaString, ezcaShort, ezcaLong, ezcaFloat, ezcaDouble`
- Basic Calls
  - `int ezcaGet(pvname, type, nelem, buff)`
  - `int ezcaPut(pvname, type, nelem, buff)`
  - `int ezcaGetWithStatus(pvname,type, nelem,buff,time,stat,sevr)`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

ezCa continued



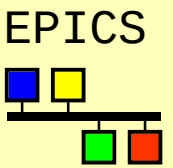
- Error Handling
  - `ezcaPerror(message)`
  - `ezcaGetErrorString(message,errorstring)`
  - `ezcaFreeErrorString(errorstring)`
- Other Groupable Functions
  - `int ezcaGetControlLimits(pvname,type,low,high)`
  - `int ezcaGetGraphicLimits(pvname,type,low,high)`
  - `int ezcaGetNelem(pvname,nelem)`
  - `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

Starting Your Practice



- mkdir test1
- cd test1
  - (setenv HOST_ARCH '$EPICS_BASE/../startup/HostArch')
- HOST_ARCH='$EPICS_BASE/../startup/HostArch'
  export HOST_ARCH
- USER='whoami' ; export USER
- makeBaseApp.pl –t simple test1
- cd test1App/src
- rcp epics@202.122.39.181:furukawa/* .
- gmake
- cd O.linux (cd O.solaris)
- gmake caTest
- caTest
  your_channel_1
  your_channel_2
  ^D
- gmake caExample
- caExample ffred

Practice Explanation 1



- `HOST_ARCH='$EPICS_BASE/../startup/HostArch'`
  `export HOST_ARCH`

  assigning a platform name for EPICS software

  (backquotes around "$EPICS … HostArch" mean

  "execute it and use the result")
- `USER='whoami' ; export USER`

  assigning a user name for EPICS software
- `mkdir test1 ; cd test1`

  making directory for our test

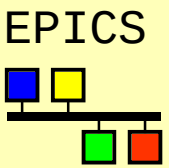
  and going into it
- `makeBaseApp.pl –t example test1`

  creating environment (directory and config files)

  for a new EPICS application

  see the manual "EPICS IOC Applications Building and Source Release Control"

Practice Explanation 2



- `cd test1App/src`
- `rcp  epics@202.122.39.181:/furukawa/ca/*  .`

  Copying example files over network from COSUN-02

  (single-quotes mean "" not expanded locally)*
- `gmake`

  build application based on the Makefile made by makeBaseApp.pl
- `cd O.solaris ; gmake caTest`

  if you modifed Makefile.Host, you don't need this step
- `caTest`

  `ffred`

  `ffreddy`

  `^D`

  executing the software, reading two pv names from stdin