EPICS
Channel Access Overview

Marty Kraimer
May 1996

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 channel access.

Search

```
ca_search_and_connect(name,pchid,connectionCallback,userarg)
ca_replace_access_rights_event(chid,accessRightsCallback)
```

- Above 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.

Puts

```
ca_array_put(type,count,chid,pvalues)
. . .
ca_flush_io()
```

- Calls are buffered until buffer full or ca_flush or ca_pend.

```
ca_put_callback(type,count,chid,pvalue,putCallback,userarg)
```

- putCallback called after all record processing resulting from put completes.

Gets

```
ca_array_get(type,count,chid,pvalues)
. . .
ca_pend_io(timeout)
```

OR

```
ca_array_get_callback(type,count,chid,getCallback,userarg)
. . .
ca_pend_event(timeout)
```

Monitors

```
ca_add_masked_array_event(type,count,chid,eventCallback,user
arg,pevid,mask)
```

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

```
ca_pend_event(timeout)
```

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 satisfied
- Also waits until ca_search with no callback are satisfied

`ca_pend_event(timeout)`

- Waits for timeout processing events as they happen

`timeout`

- 0 - Wait forever
- Short timeout, e.g. .0001 (seconds) is acceptable but DON'T hog CPU.

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 call

ezCa - Easy Channel Access

- Goals
    - Easy to use.
    - Provide non-callback synchronous model.
- Requirements
    - Group of about 6 people developed requirements.
    - Basic requirement was that brief description should be about one page
        - eaca.h is one page!!!
    - Support for synchronous groups and individual requests.
    - Pass channel names, auto search, easy error handling.

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);
```

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)
```

Other Functions

```
void ezcaAutoErrorMessageOff()
void ezcaAutoErrorMessageOn()
int ezcaDelay(seconds)
int ezcaGetRetryCount()
int ezcaSetRetryCount()
float ezcaGetTimeout()
int ezcaSetTimeout(seconds)
```

Escape to Channel Access

```
ezcaPvTochid(pvname,chid)
```

Example Program

```
/*example.c*/
/*From stdin read a list of pvnames. Get each as a double*/
#include <stddef.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include <cadef.h>
#include <tsDefs.h>
#include <ezca.h>
#define MAX_PV 1000
#define MAX_PV_NAME_LEN 40
```

```
main()
{
    int         npv=0;
    double      *pdata;
    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);
```

```
      npv++;
    }
    pdata = (double *)calloc(npv,sizeof(double));
    ezcaStartGroup();
    for(i=0; i<npv; i++) {
      status = ezcaGet(pname[i],ezcaDouble,1,(void *)
&pdata[i]);
      if(status) {
          ezcaPerror("ezcaGet");
          break;
      }
    }
    status = ezcaEndGroup();
    if(status) ezcaPerror("ezcaEndGroup");
    for(i=0; i<npv; i++) {
      printf("%s %f\n",pname[i],pdata[i]);
    }
    return(0);
}
```