Introduction to the Channel Access Client Library

Kenneth Evans, Jr.
Kay Kasemir
Channel Access Reference Manual

• The place to go for more information
• Found in the EPICS web pages
  - Look under Documents
  - Also under Base, then a specific version of Base
EPICS Overview
Search and Connect Procedure

1. UDP Broadcast Sequence
   - Who has it?

2. UDP Reply
   - I have it!

3. TCP Connection
   - Let's talk!

MEDM MEDM Client Client Client MEDM

Check Check IOC Check

Meter Power Supply Camera
**Search Request**

- A search request consists of a sequence of UDP packets
  - Only goes to EPICS_CA_ADDR_LIST
  - Starts with a small interval (30 ms), that doubles each time
  - Until it gets larger than 5 s, then it stays at 5 s
  - Stops after 100 packets or when it gets a response
  - Never tries again until it sees a beacon anomaly or creates a new PV
  - Total time is about 8 minutes to do all 100

- Servers have to do an Exist Test for each packet
- Usually connects on the first packet or the first few
- Non-existent PVs cause a lot of traffic
  - Try to eliminate them
Beacons

• A Beacon is a UDP broadcast packet sent by a Server
• When it is healthy, each Server broadcasts a UDP beacon at regular intervals (like a heartbeat)
  – EPICS_CA_BEACON_PERIOD, 15 s by default
• When it is coming up, each Server broadcasts a startup sequence of UDP beacons
  – Starts with a small interval (25 ms, 75 ms for VxWorks)
  – Interval doubles each time
  – Until it gets larger than 15 s, then it stays at 15 s
    • Takes about 10 beacons and 40 s to get to steady state
• Clients monitor the beacons
  – Determine connection status, whether to reissue searches
Virtual Circuit Disconnect

• 3.13 and early 3.14
  – Hang-up message or no response from server for 30 sec.
  – If not a hang-up, then client sends “Are you there” query
  – If no response for 5 sec, TCP connection is closed
  – MEDM screens go white
  – Clients reissue search requests

• 3.14.5 and later
  – Hang-up message from server
  – TCP connection is closed
  – MEDM screens go white
  – Clients reissue search requests
Virtual Circuit Unresponsive

• 3.14.5 and later
  – No response from server for 30 sec.
  – Client then sends “Are you there” query
  – If no response for 5 sec, TCP connection is not closed
    • *For several hours, at least*
  – MEDM screens go white
  – Clients do not reissue search requests
    • *Helps with network storms*

  – Clients that do not call `ca_poll` frequently get a virtual circuit disconnect even though the server may be OK
    • *Clients written for 3.13 but using 3.14 may have a problem*
    • *May be changed in future versions*
Important Environment Variables

• EPICS_CA_ADDR_LIST
  – Determines where to search
  – Is a list (separated by spaces)
    • “123.45.1.255 123.45.2.14 123.45.2.108”
  – Default is broadcast addresses of all interfaces on the host
    • Works when servers are on same subnet as Clients
  – Broadcast address
    • Goes to all servers on a subnet
    • Example: 123.45.1.255
    • Use ifconfig –a on UNIX to find it (or ask an administrator)

• EPICS_CA_AUTO_ADDR_LIST
  – YES: Include default addresses above in searches
  – NO: Do not search on default addresses
  – If you set EPICS_CA_ADDR_LIST, usually set this to NO
EPICS_CA_ADDR_LIST

```
MEDM    MEDM    Client    Client    Client    MEDM

Broadcast  123.45.1.255
Subnet 1

Server

IOC

Meter

Specific  123.45.2.108
Subnet 2

IOC

Power Supply

Not Included

IOC

Camera

Not Included
```
Other Environment Variables

• CA Client
  EPICS_CA_ADDR_LIST
  EPICS_CA_AUTO_ADDR_LIST
  EPICS_CA_CONN_TMO
  EPICS_CA_BEACON_PERIOD
  EPICS_CA_REPEATER_PORT
  EPICS_CA_SERVER_PORT
  EPICS_CA_MAX_ARRAY_BYTES
  EPICS_TS_MIN_WEST

• CA Server
  EPICS_CAS_SERVER_PORT
  EPICS_CAS_AUTO_BEACON_ADDR_LIST
  EPICS_CAS_BEACON_ADDR_LIST
  EPICS_CAS_BEACON_PERIOD
  EPICS_CAS_BEACON_PORT
  EPICS_CAS_INTF_ADDR_LIST
  EPICS_CAS_IGNORE_ADDR_LIST

• See the Channel Access Reference Manual for more information
Channel Access

- The main CA client interface is the "C" library that comes with EPICS base
  - Internally uses C++, but API is pure C.
- Almost all other CA client interfaces use that C library
  - Exception: New pure Java JAC
Basic Procedure for a Channel Access Client

- Initialize Channel Access
  - `ca_task_initialize` or `ca_context_create`
- Search
  - `ca_search_and_connect` or `ca_create_channel`
- Do get or put
  - `ca_get` or `ca_put`
- Monitor
  - `ca_add_event` or `ca_create_subscription`
- Give Channel Access a chance to work
  - `ca_poll`, `ca_pend_io`, `ca_pend_event`
- Clear a channel
  - `ca_clear_channel`
- Close Channel Access
  - `ca_task_exit` or `ca_context_destroy`
**makeBaseApp.pl**

- Includes a template for basic CA client in C:
  - Start with this:
    ```
    makeBaseApp.pl -t caClient cacApp
    make
    ```
  - Result:
    ```
    bin/linux-x86/caExample <some PV>
    bin/linux-x86/caMonitor <file with PV list>
    ```
  - Then read the sources, compare with the reference manual, and edit/extend to suit your needs.
makeBaseApp's caExample.c

- Minimal CA client program.
  - Fixed timeout, waits until data arrives.
  - Requests everything as 'DBR_DOUBLE'.
    - ... which results in values of C-type 'double'.
    - See db_access.h header file for all the DBR_... constants and the resulting C types or structures.
    - In addition to the basic DBR_<type> requests, it is possible to request packaged attributes like DBR_CTRL_<type> to get { value, units, limits, ...} in one request.
Excerpt from db_access.h

```c
/* values returned for each field type */
/* DBR_DOUBLE returns a double precision floating point number */
/* DBR_CTRL_DOUBLE returns a control double structure (dbr_ctrl_double) */
*/

/* structure for a control double field */
struct dbr_ctrl_double{
    dbr_short_t     status;  /* status of value */
    dbr_short_t     severity; /* severity of alarm */
    dbr_short_t     precision; /* number of decimal places */
    dbr_short_t     RISC_pad0; /* RISC alignment */
    char            units[MAX_UNITS_SIZE]; /* units of value */
    dbr_double_t    upper_disp_limit; /* upper limit of graph */
    dbr_double_t    lower_disp_limit; /* lower limit of graph */
    dbr_double_t    upper_alarm_limit;
    dbr_double_t    upper_warning_limit;
    dbr_double_t    lower_alarm_limit;
    dbr_double_t    lower_warning_limit;
    dbr_double_t    upper_ctrl_limit; /* upper control limit */
    dbr_double_t    lower_ctrl_limit; /* lower control limit */
    dbr_double_t    value;      /* current value */
};
```
Better CA client program.
- Registers callbacks to get notified when connected or disconnected
- Subscribes to value updates instead of waiting.
- … but still uses the same data type (DBR_STRING) for everything.
Ideal CA client?

- Use callbacks for everything
  - no idle 'wait', no fixed time outs.
- Upon connection, check the channel's *native* type (int, double, string, ...)
  - to limit the type conversion burden on the IOC.
- ... request the matching DBR_CTRL_<type> *once*
  - to get the full channel detail (units, limits, ...).
- ... and then subscribe to DBR_TIME_<type> to get updates of only time/status/value
  - so now we always stay informed, yet limit the network traffic.
  - *Only subscribe once*, not with each connection, because CA client library will automatically re-activate subscriptions!
- This is what EDM, archiver, ... do.
  - Quirk: They don't learn about online changes of channel limits, units, ....
  - Doing that via a subscription means more network traffic, and CA doesn't send designated events for 'meta information changed'.
Side Note: SNL just to get CAC help

• This piece of SNL handles all the connection management and data type handling:

```c
double value;
assign value to "fred";
monitor value;
```

• Extend into a basic 'camonitor':

```c
evflag changed;
sync value changed;

ss monitor_pv
{
    state check
    {
        when (efTestAndClear(changed))
        {
            printf("Value is now %g\n", value);
        }
    }
}
```
**Quick Hacks, Scripts**

- In many cases, one can get by just fine by invoking the command-line 'caget' from within bash/perl/python/php.
- Especially if you only need to read/write one value of a PV, not a subscription!
- There are more elaborate CAC bindings available for perl/python/php
  - But that means you have to find, build and later maintain these!
  - A basic p* script is portable, but you'd have to install the CAC-for-p* binding separately for Linux, Win32, MacOS…
Perl Example

use English;

# Get the current value of a PV
# Argument: PV name
# Result: current value
sub caget($) {
    my ($pv) = @ARG;
    open(F, "caget -t $pv |") or die "Cannot run 'caget'\n";
    $result=<F>;
    close(F);
    chomp($result);
    return $result;
}

# Do stuff with PVs
$fred = caget("fred");
$jane = caget("jane");
$sum = $fred + $jane;
printf("Sum: %g\n", $sum);
Matlab 'MCA' Extension (Works with Octave as well)

• Same setup & maintenance issue as for p/p/p!
  – … but may be worth it, since Matlab adds tremendous number crunching and graphing.

• Initial setup
  – Get MCA sources (see links on APS EPICS web)
  – Read the README, spend quality time with MEX.

• Assume that's done by somebody else
  – You are in the SNS control room
  – 'caget' from EPICS base works
  – Matlab works (try "matlab -nojvm -nodesktop")

• Do this once:
  
  cd $EPICS_EXTENSIONS/src/mca
  source setup.matlab
  – … and from now on, Matlab should include MCA support
MCA Notes

• Basically, it's a chain of
  – pv = mcaopen('some_pv_name');
  – value = mcaget(pv);
  – mcaput(pv, new_value);
  – mcaclose(pv);
• Your pv is 'connected' from ..open to ..close
  – When getting more than one sample, staying connected is much more efficient than repeated calls to 'caget'.
• Try 'mca<tab>}' command-line completion to get a list of all the mca… commands
• Run 'help mcaopen' etc. to get help
Matlab/MCA Examples

Command Window

```
>> fred_pv = mcaopen('fred');
>> jane_pv = mcaopen('jane');
>> fred_value = mcaget(fred_pv);
>> jane_value = mcaget(jane_pv);
>> fred_value + jane_value
ans =
    0.3476

>> alan_pv = mcaopen('alan');
>> alan_value = mcaget(alan_pv);
>> plot(alan_value);
>> mcclose(alan_pv);
>> mcclose(jane_pv);
>> mcclose(fred_pv);
>>
>> help mcaopen
MCAOPEN open a Channel Access connection to an EPICS Process Variable

H = MCAOPEN(PVNAME);
    If successful H is a unique nonzero integer handle associated with this PV.
    Returned handle is 0 if a connection could not be established

[H1, ... ,Hn] = MCAOPEN(PVNAME1, ... ,PVNAMEn);
    Is equivalent to but more efficient than multiple single-argument calls
    H1 = MCAOPEN(PVNAME1);
    ...
    Hn = MCAOPEN(PVNAMEn);
```
Java

- There is actually a JNI and a pure Java binding.
  - Only difference in initialization, then same API.
  - Usage very much like C interface, "real programming" as opposed to Matlab, but in a more forgiving Java VM.
- See Docs/Java CA example.
Acknowledgements

• Channel Access on every level in detail:
  – Jeff Hill (LANL)
• makeBaseApp.pl
  – Ralph Lange (BESSY) and others
• MCA
  – Andrei Terebilo (SLAC) is the original author,
  – Carl Lionberger maintained it for a while (then SNS)
• Java CA
  – Eric Boucher is the original author (then APS),
  – Matej Sekoranja maintains it;
    he added the pure java version (Cosylab)