Argonne NATIONAL LABORATORY

Channel Access Client APIs

Andrew Johnson — Computer Scientist, AES-SSG

Includes material by Kay Kasemir, ORNL

U.S. DEPARTMENT OF ENERGY | Office of Science

Channel Access

- The main programming interface for writing Channel Access clients is the C library that comes with EPICS base
  - Internally uses C++, but API is pure C
- Almost all CA client interfaces in other languages call the C library
  - Main exception: Pure Java library 'CAJ' (may still have some issues)
- Documentation:
  - *EPICS R3.14 Channel Access Reference Manual* by Jeff Hill et al.
  - *CA - Perl 5 interface to EPICS Channel Access* by Andrew Johnson
  - In <base>/html, or from the EPICS web site
- This lecture covers
  - Fundamental API concepts (using Perl)
  - Template examples in C, discussion of Matlab and Java

Why teach the Perl API?

- Simpler, easier to learn than C
- Same principles, less code required

- Perl 5 API calls C interface internally
  - Builds on most Unix-like workstation platforms (not Windows)

Search and Connect to a PV

```
use lib '/path/to/base/lib/perl';
use CA;

my $chan = CA->new($ARGV[0]);
CA->pend_io(1);

printf "PV: %s\n", $chan->name;
printf "  State:          %s\n", $chan->state;
printf "  Host:           %s\n", $chan->host_name;
my @access = ('no ', '');
printf "  Access rights: %sread, %swrite\n",
    $access[$chan->read_access], $access[$chan->write_access];
printf "  Data type:      %s\n", $chan->field_type;
printf "  Element count: %d\n", $chan->element_count;
```

- This is the basic cainfo program in Perl (without error checking)

Get and Put a PV

```
use lib '/path/to/base/lib/perl';
use CA;

my $chan = CA->new($ARGV[0]);
CA->pend_io(1);

$chan->get;
CA->pend_io(1);
printf "Old Value: %s\n", $chan->value;

$chan->put($ARGV[1]);
CA->pend_io(1);

$chan->get;
CA->pend_io(1);
printf "New Value: %s\n", $chan->value;
```

- This is the basic caput program in Perl (without error checking)

Monitor a PV

```
use lib '/path/to/base/lib/perl';
use CA;

my $chan = CA->new($ARGV[0]);
CA->pend_io(1);

$chan->create_subscription('v', \&val_callback);
CA->pend_event(0);

sub val_callback {
   my ($chan, $status, $data) = @_;
   if (!$status) {
       printf "PV: %s\n", $chan->name;
       printf "  Value: %s\n", $data;
   }
}
```

- This is a basic camonitor program in Perl (without error checking)

Error Checking

- What happens if the PV search fails, e.g. the IOC isn't running, or it's busy and takes longer than 1 second to reply?
  - `CA->pend_io(1)` times out
  - CA library throws a Perl exception (die)
  - Program exits after printing:
    - ECA_TIMEOUT - User specified timeout on IO operation expired at test.pl line 5.
- We can trap the Perl exception using
  - ```
    eval {CA->pend_io(1)};
    if ($@ =~ m/^ECA_TIMEOUT/) { ... }
    ```
- How can we write code that can recover from failed searches and continue doing useful work?
    ```

Event-driven Programming

- First seen when setting up the CA monitor:
  - `$chan->create_subscription('v', \&callback);`
    `CA->pend_event(0);`
  - The CA library executes our callback subroutine whenever the server provides a new data value for this channel
  - The `CA->pend_event()` routine must be running for the library to execute callback routines
    - The Perl CA library is single threaded
    - Multi-threaded C programs can avoid this requirement
- Most CA functionality can be event-driven

Event-driven PV Search and Connect

```
use lib '/path/to/base/lib/perl';
use CA;

my @chans = map {CA->new($_, \&conn_callback)} @ARGV;
CA->pend_event(0);

sub conn_callback {
    my ($chan, $up) = @_;
    printf "PV: %s\n", $chan->name;
    printf "  State:          %s\n", $chan->state;
    printf "  Host:           %s\n", $chan->host_name;
    my @access = ('no ', '');
    printf "  Access rights: %sread, %swrite\n",
        $access[$chan->read_access], $access[$chan->write_access];
    printf "  Data type:      %s\n", $chan->field_type;
    printf "  Element count: %d\n", $chan->element_count;
}
```

- The cainfo program using callbacks

Event-driven PV Monitor

```
use lib '/path/to/base/lib/perl';
use CA;
my @chans = map {CA->new($_, \&conn_cb)} @ARGV;
CA->pend_event(0);
sub conn_cb {
    my ($ch, $up) = @_;
    if ($up && ! $monitor{$ch}) {
        $monitor{$ch} = $ch->create_subscription('v', \&val_cb);
    }
}
sub val_cb {
   my ($ch, $status, $data) = @_;
   if (!$status) {
       printf "PV: %s\n", $ch->name;
       printf "  Value: %s\n", $data;
   }
}
```

- The camonitor program using callbacks

Data Type Requests

- Most data I/O routines handle data type automatically
  - **`$chan->get`** fetches one element in the channel's native type
    - Value is returned by **`$chan->value`**
    - Arrays are not supported, no type request possible
  - **`$chan->get_callback(SUB)`** fetches all elements in the channel's native data type
    - Optional TYPE and COUNT arguments to override
  - **`$chan->create_subscription(MASK, SUB)`** requests all elements in the channel's native type
    - Optional TYPE and COUNT arguments to override
  - **`$chan->put(VALUE)`** puts values in the channel's native type
    - VALUE may be a scalar or an array
  - **`$chan->put_callback(SUB, VALUE)`** puts values in the channel's native data type
    - VALUE may be a scalar or an array

Specifying Data Types

- The TYPE argument is a string naming the desired DBR_xxx type
  - See the CA Reference Manual for a list
- The COUNT argument is the integer number of elements

- If you request an array, the callback subroutine's $data argument becomes an array reference
- If you request a composite type, the callback subroutine's $data argument becomes a hash reference
  - The hash elements are different according to the type you request
  - See the Perl Library documentation for details

Base caClient template

- EPICS Base Includes a makeBaseApp.pl template that builds two basic CA client programs written in C:
  - Run 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

CaClient's caExample.c

- Minimal CA client program
- Fixed timeout, waits until data arrives
- Requests everything as ‘DBR_DOUBLE’
  - … which results in values of type ‘double’
  - See db_access.h header file for all the DBR_… constants and the resulting C types and 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

```
/* 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_warning_limit;
        dbr_double_t    lower_alarm_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 */
};
```

caClient's caMonitor.c

- Better CA client program
  - Registers callbacks to get notified when connected or disconnected
  - Subscribes to value updates instead of waiting
  - … but still uses one data type (DBR_STRING) for everything

Ideal CA client?

- Register and use callbacks for everything
  - Event-driven programming; polling loops or fixed time outs
- On connection, check the channel's native type
  - Limit the data type conversion burden on the IOC
- Request the matching DBR_CTRL_*type* once
  - this gets the full channel detail (units, limits, …)
- Then subscribe to DBR_TIME_*type* for time+status+value updates
  - Now we always stay informed, yet limit the network traffic
  - Only subscribe once at first connection; the CA library automatically re-activates subscriptions after a disconnect/reconnect
- This is what MEDM, StripTool, etc do
  - Quirk: They don't learn about run-time changes of limits, units, etc.
    - Recent versions of CA support DBE_PROPERTY monitor event type
    - This solves that issue

Side Note: SNL just to get CAC help

- This piece of SNL handles all the connection management and data type handling:
  - ```
    double value;
    assign value to "fred";
    monitor value;
    ```
- Extend into a basic 'camonitor':
  - ```
    evflag changed;
    sync value changed;

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

Quick Hacks, Scripts

- In many cases, scripts written in bash/perl/python/php can just invoke the command-line ‘caget’ and ‘caput’ programs
- Especially useful if you only need to read/write one PV value and not subscribe to value updates
- CA Client library bindings are available for Perl, Python & PHP
  - Perl bindings are included in EPICS Base (not on MS Windows)
  - You have to find, build and update them for Python and PHP
    - Your script may be portable, but you still have to install the CAC-for-p* binding separately for Linux, Win32, MacOS…

Quick Perl Example

```
# caget: Get the current value of a PV
# Argument: PV name
# Result: PV value
sub caget {
    my ($pv) = @_;
    open(my $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

- 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 EPICS website)
  - Read the README, spend quality time with MEX
- Assume that's done by somebody else
  - ‘caget’ from EPICS base works
  - Matlab works (try "matlab -nojvm -nodesktop")
- Do this once:

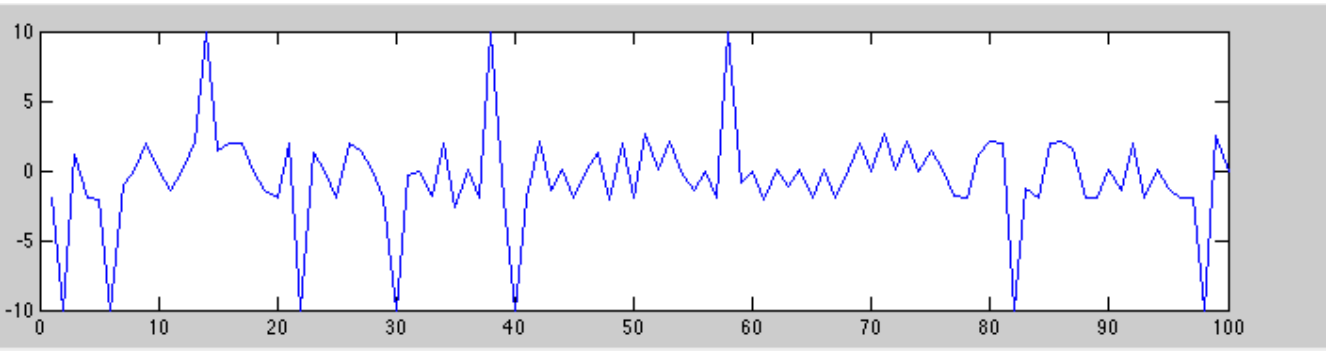
```
cd $EPICS_EXTENSIONS/src/mca
source setup.matlab
```

  - … 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 =

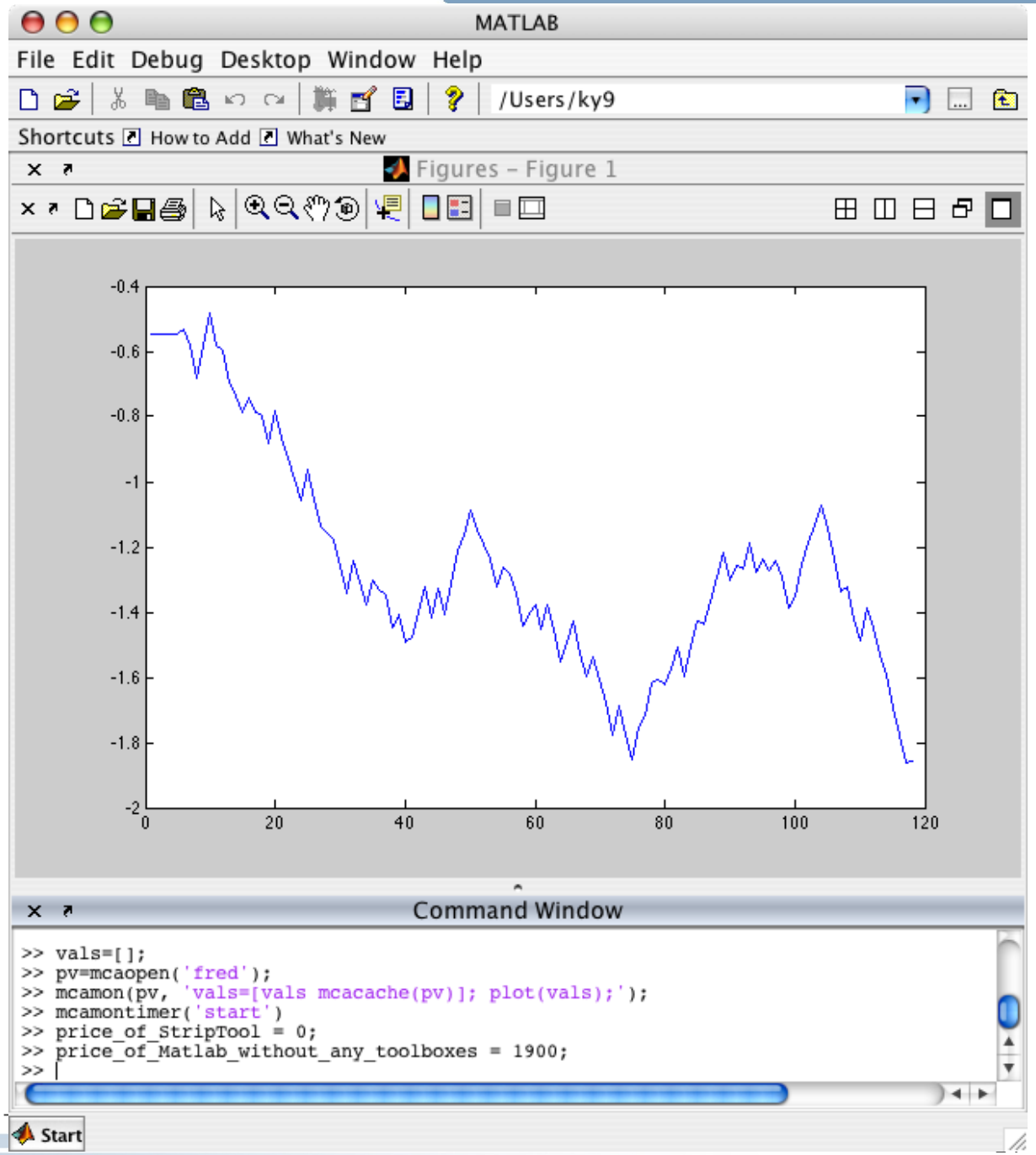
    0.3476

>> alan_pv = mcaopen('alan');
>> alan_value = mcaget(alan_pv);
>> plot(alan_value);
>> mcaclose(alan_pv);
>> mcaclose(jane_pv);
>> mcaclose(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);
```

MCA Value Subscription



Java

- There are now 2 CA bindings, using JNI or pure Java
  - Only difference is the initialization, they support the same API
  - Usage very much like C interface, “real programming” as opposed to Matlab, but in the more forgiving Java VM
- A Java CA example can be found at
  - http://ics-web.sns.ornl.gov/kasemir/train_2006/4_2_Java_CA.tgz

Acknowledgements

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