Part I: EPICS R3.12 Channel Access Reference Manual

Jeffrey O. Hill

JOHill@lanl.gov

Controls and Automation Group
AOT Division
Los Alamos National Laboratory
Los Alamos, NM 87545

CHAPTER 1 Configuration

1.1 Why Reconfigure Channel Access

Typically an EPICS site will choose to reconfigure Channel Access because:

- They wish for two or more independent projects to share the same network without fear of interaction between their control systems.
- They wish to setup a test system that does not interact with an operational system.
- They wish to setup a system that uses address lists instead of broadcasts for name resolution and server beacons.
- They wish to setup a system that uses multiple IP subnets.
- They wish to adjust client disconnect time outs or server beacon intervals.
- They wish to use EPICS time stamps from a client that is in a time zone other than the US Mountain Time Zone.

1.2 EPICS Environment Variables

All channel access configuration occurs through EPICS environment variables. Table 1 on page 4 contains a list of all of the EPICS environment variables related to channel access.

TABLE 1. Channel Access EPICS Environment Variables

Environment Variable Name	Range	Default	Section
EPICS_CA_ADDR_LIST	{N.N.N.NN.N.N.N...}	<none>	1.3
EPICS_CA_AUTO_ADDR_LIST	{YES, NO}	YES	1.3

Environment Variable Name	Range	Default	Section
EPICS_CA_CONN_TMO	$r > 0.1$ seconds	30.0	1.5
EPICS_CA_BEACON_PERIOD	$r > 0.1$ seconds	15.0	1.5
EPICS_CA_REPEATER_PORT	$i > 5000$	5065	1.4
EPICS_CA_SERVER_PORT	$i > 5000$	5064	1.4
EPICS_TS_MIN_WEST	$-720 \le i \le 720$ minutes	360	1.6

When executing under UNIX, EPICS environment variables can be modified by typing 'setenv <EPICS environment variable name> "<new value for the variable>"'. On IOCs, EPICS environment variables can be modified by adding a line of the form '<EPICS environment variable name> DBF_STRING <new value for the variable>' to the resource definition file (typically "resource.def" in the application directory that the IOC boots from). In the IOC start-up script the single argument to iocInit() should be the name (and potentially the file system path) of the resource definition file.

1.3 CA and Wide Area Networks

Normally in a local area network (LAN) environment CA discovers the address of the host for an EPICS channel by broadcasting frames containing a list of channel names (CA search messages) and waiting for responses from the servers that host the channels identified. Likewise CA clients efficiently discover that CA servers have recently joined the LAN or disconnected from the LAN by monitoring periodically broadcasted “beacons” sent out by the servers. Since hardware broadcasting requires special hardware capabilities, we are required to provide additional configuration information when EPICS is extended to operate over a wide area network (WAN).

1.3.1 IP Network Administration Background Information

EPICS Channel Access is implemented over the internet protocols (IP). IP addresses are divided into host and network portions. The boundary between each portion is determined by the IP “netmask”. Portions of the IP address corresponding to zeros in the “netmask” specify the hosts address within an IP “subnet”. Portions of the IP address corresponding to binary ones in the “netmask” specify the address of a host’s IP “subnet”. It is easy to examine these parameters on most UNIX hosts by typing "netstat -i". More detailed information can be obtained by typing "ifconfig <interface name>" where <interface name> was obtained from the output of “netstat”. Similarly this information can be obtained from the vxWorks operating system by typing “ifShow”.

Normally the scope of a broadcasted frame will be limited to an IP subnet however it is possible for a host on one IP subnet to issue a broadcast frame on a different IP subnet. Addresses with the host address portion set to all zeros or all ones are special. Modern IP implementations reserve destination addresses with the host portion set to all ones for the purpose of sending broadcasts to a particular subnet. Therefore in theory we can issue a broadcast frame on any broadcast capable LAN within the interconnected internet by specifying the proper subnet address combined with a host portion set to all ones. In practice this ability to send "directed broadcasts" may be limited by configuration options in routers which determine if these directed broadcasts are discarded.

The proper directed broadcast address required to reach a particular vxWorks host can be obtained by logging into that host and typing “ifShow”. Ignore the loop back interface and use the broadcast address associated with the interface along the path to your router (usually there is only one choice here - ei). Similarly the proper broadcast address required to reach a particular UNIX host can be found by typing "netstat -i "followed by "ifconfig <interface name>" (usually "ifconfig leo").

IP ports are positive integers. Within the IP protocols the IP address, port number, and protocol type uniquely identify the source and destination of a particular frame on particular host computers. In practice the port number also uniquely identifies a protocol layered over one of the IP protocols. Servers are usually addressed by a “well known” port number. Clients are usually assigned a unique port number during initialization. IP ports below 1024 are reserved for servers that provide standardized facilities such as mail or file transfer. Port number between 1024 and 5000 are normally reserved for clients and other dynamic port number assignments.

1.3.2 Configuring CA for Multiple Subnets

During initialization CA builds a list of the destination addresses used when sending CA client name resolution (search) requests and CA server beacons. This table is initialized by traversing the IP interfaces attached to the host. For each interface found that is attached to a broadcast capable IP subnet the broadcast address of that subnet is added to the list. For each point to point interface found the destination address of that link is added to the list. This available network interface driven initialization is skipped only if the EPICS environment variable “EPICS_CA_AUTO_ADDR_LIST” exists and its value is one of either “no” or “NO”.

```
UNIX:
% setenv EPICS_CA_AUTO_ADDR_LIST YES

vxWorks:
Add the following line to the file "resource.def".
EPICS_CA_AUTO_ADDR_LIST DBF_STRING YES
% iocInit "resource.def"
```

Next the list of destination addresses is appended to by any IP addresses specified in the EPICS environment variable “EPICS_CA_ADDR_LIST”. An identical procedure is followed by the server when it initializes the list of addresses that the server sends beacons to. Proper configuration of CA for multiple subnets requires that CA name resolution (search requests) frames pass from CA clients to the targeted CA servers. Likewise proper operation also requires that beacons issued by the targeted CA servers reach the hosts of all potential clients. For this reason the “EPICS_CA_ADDR_LIST” environment variable parameter will often be identical within a group of clients that communicate between themselves. The contents of EPICS_CA_ADDR_LIST is currently constrained to be a list of dot format IP addresses seperated by white space. Host names may not be substituted for IP addresses because some of the operating system environments in which CA runs do not support host name to IP address translation.

```
UNIX:
```

```
% setenv EPICS_CA_ADDR_LIST "1.2.3.255 8.9.10.255"

vxWorks:
Add the following line to the file "resource.def".
EPICS_CA_ADDR_LIST DBF_STRING 1.2.3.255 8.9.10.255
% iocInit "resource.def"
```

Note that there is no requirement that the addresses specified in the EPICS_CA_ADDR_LIST be a broadcast addresses. However this will often be the most convenient choice.

1.4 Configuring the IP port numbers utilized by CA

CA uses two TCP/IP port numbers and two UDP/IP port numbers as identified in Table 2 on page 7.

TABLE 2. Channel Access Port Numbers

Protocol	Default	Purpose
TCP/IP	5064	Server
TCP/IP	5065	Reserved for future expansion
UDP/IP	5064	Process variable name resolution
UDP/IP	5065	Repeater (broadcast fan out within a host)

A particular site may choose to reconfigure the IP port numbers used by Channel Access. This will normally occur because a site wants to setup two or more completely independent control systems that will share all or part of the same network. For instance a site may wish to setup an operational control system and a test control system on the same network. The operational system might be assigned special port numbers and the test control system might be assigned the default CA port numbers. This configuration would allow the test system and the operational system to use identical PV names without fear of collision. A site may also need to configure the CA port numbers because some other facility at the site has already reserved the two default port number that are normally used by CA.

```
UNIX:
% setenv EPICS_CA_REPEATER_PORT 20000
% setenv EPICS_CA_SERVER_PORT 20001

vxWorks:
Add the following line to the file "resource.def".
EPICS_CA_REPEATER_PORT DBF_STRING 20000
EPICS_CA_SERVER_PORT DBF_STRING 20001
% iocInit "resource.def"
```

1.5 Configuring Disconnect Time Out / Server Beacon Period

If CA has not seen a beacon from a server that it is connected to for EPICS_CA_-CONN_TMO seconds then an echo message is sent to the server over TCP/IP. If this echo message isn't replied to by the server within 5 additional seconds then the client will assume that the server is no longer present on the network and disconnect. Disconnecting implies notification of any application programs that may be using Channel Access to communicate with the server in question. Disconnecting also implies releasing any resources reserved while communicating with the missing server. The parameter EPICS_CA_CONN_TMO is specified in floating point seconds. Note that for short time outs, the delay until disconnect will also depend on the rate at witch applications poll into the CA client library (and allow CA background activity to be attended to).

```
UNIX:
% setenv EPICS_CA_CONN_TMO 15.0

vxWorks:
Add the following line to the file "resource.def".
EPICS_CA_CONN_TMO DBF_STRING 15.0
% iocInit "resource.def"
```

When a CA server initializes it initially sends "beacon" messages to each address specified in EPICS_CA_ADDR_LIST, and also any addresses auto configured from network interfaces found, with a very short period between "beacons". However, this short period is doubled each time a "beacon" is sent so that it will rapidly reached a plateau specified by EPICS_CA_BEACON_PERIOD. The parameter EPICS_CA_BEACON_-PERIOD is specified in floating point seconds. For efficient operation it is recommended that EPICS_CA_BEACON_PERIOD be set to at least one half of the value specified for EPICS_CA_CONN_TMO.

```
UNIX:
% setenv EPICS_CA_BEACON_PERIOD 7.5

vxWorks:
Add the following line to the file "resource.def".
EPICS_CA_BEACON_PERIOD DBF_STRING 7.5
% iocInit "resource.def"
```

1.6 Configuring the Time Zone

While the CA client library does not translate in between the local time and raw EPICS time zone independent time stamps, many EPICS client side applications do call core EPICS libraries which provide this service. Therefore, users will need to configure the correct time zone prior to using EPICS client side application programs for the first time. To set the correct time zone users must compute the number of positive minutes west of GMT (maximum 720 inclusive) or the negative number of minutes east of GMT (minimum -720 inclusive) of the local time zone. This integer value is then placed in the variable EPICS_TS_MIN_WEST.

```
UNIX:
% setenv EPICS_TS_MIN_WEST 420

vxWorks:
Add the following line to the file "resource.def".
```

```
EPICS_TS_MIN_WEST DBF_STRING 420
% iocInit "resource.def"
```

TABLE 3. Time Zone Examples

Time Zone	EPICS_TS_MIN_WEST
USA Eastern	300
USA Central	360
USA Mountain	420
USA Pacific	480
Alaska	540
Hawaii	600
Japan	-540
United Kingdom	-60
Germany	-120

CHAPTER 2 Notes

Throughout this document CA will be used as an acronym for channel access.

If the server for a channel is located in a different node than the client then the ca_XXX() operations that communicate with the server return status indicating the validity of the request. In other words success returned from the client library routine indicates that it checked your request, it appeared to be valid, and it forwarded it to the server. Unsuccessful status codes returned from the client library are listed with each routine in this manual. Operations that appear to be valid to the client can still fail in the server. An example of this type of error is attempting to write the string “off” to a floating point field (writing the string “3.3” would be accepted). When the server detects a failure an exception handler is executed in the client. The default exception handler just prints a message on the console (“ca_add_exception_event()” on page 25). Not waiting for status to return from the server after each remote operation results in significant performance gains. If the server for the channel and the client are located on the same node then the ca_XXX() operations bypass the server and directly interact with the IOC database. So in this instance the ca_XXX() routines always return the status of the operation directly to the caller with no opportunity for asynchronous notification of failure via an exception handler.

All requests which require interaction with a CA server are accumulated (buffered) and not forwarded to the IOC until one of ca_flush_io(), ca_pend_io(), ca_pend_event(), or ca_sg_pend() are called allowing several operations to be efficiently sent over the network in one message. Currently this applies to ca_search(), ca_put(), ca_put_callback(), ca_get(), ca_get_callback(), ca_add_event(), ca_clear_event(), ca_clear_channel(), ca_sg_put(), ca_sg_get(), ca_modify_user_name(), and ca_modify_host_name().

Any process variable values written into your program’s variables by ca_get() should not be referenced by your program until ECA_NORMAL has been received from ca_pend_io().

All arguments of type"chtype" expect a constant from the set of DBR_XXXX. These constants, defined in db_access.h, enumerate which of the standard data types you wish to transfer. Some routines have arguments requiring that you specify an address at which channel access is to write a value of type DBR_XXXX. Care should be taken to ensure that you have reserved space of sufficient size. Use the structures and size returning MACROS/arrays provided in db_access.h for this purpose. See also the MACROS manual page in this document.

For routines which require an argument specifying the number of elements no more than the channel's native count may be requested. The channels native count is available from ca_element_count(). If less elements than the channels native count are requested the requested values will be fetched from the beginning of the channels array of values. Currently channel access limits the number of elements in an array to be no more than approximately 16k divided by the size of one element in the array.

Channel connections across the network are inherently transient. A handler routine may be installed to be run whenever an IO channel connects or disconnects. Channels are always initially disconnected. An argument provided to the user supplied handler distinguishes between connect and disconnect events. A different handler routine may be installed for each channel. *After a connection handler is established on a particular channel by the user CA will neither block for or print the results of this channel's search, or value fetch associated with the search, while in ca_pend_io().* Your connection handler (if any) will be run from inside ca_search() if your code - the CA client - and the channel searched for are both hosted by the same IOC (if it is a local channel).

CA attempts to provide an identical environment on several OS. One of the few exceptions to this involves the way that event handlers are run under each OS. Under vxWorks only, event handlers are allowed to preempt the main (initiating) channel access thread. This was allowed because preemption might be exploited for useful purposes in some cases. However, once an event handler begins it is always allowed to run to completion prior to running another event handler. That is events never preempt events. Event preemption is currently not supported under VMS because of reentrancy problems with the MULTINET socket library. When running under VMS or UNIX event handlers will not be executed unless inside of a CA client library routine such as ca_pend_event().

For proper operation CA must periodically be allowed to take care of background activity. This requires that your application must either wait in one of ca_pend_event(), ca_pend_io(), or ca_sg_block() or it must call ca_pend_event() at least every 15 seconds. See "ca_add_fd_registration()" on page 30 for more information about applications that perform better with multiplexed file descriptor IO.

For portability reasons CA functions do not return status following UNIX convention. The error number and the severity of the status are embedded in the CA function return. Applications shouldn't test the success of a CA function call by checking to see if the returned value is zero as is the UNIX convention. Below are several methods to test CA function returns. See "ca_signal()" on page 24 for more information on this topic.

```
status = ca_XXXX();
```

```
SEVCHK( status, "It failed and this is why ... ");

if(status & CA_M_SUCCESS){
  printf("The requested operation didn't complete successfully\n");
}

if(status != ECA_NORMAL){
  printf("Something off normal occurred while performing the operation");
}
```

CHAPTER 3 Function Calls

NAME ca_task_initialize()

SYNOPSIS

```
#include      <cadef.h>
int           ca_task_initialize();
```

ARGUMENTS

none

DESCRIPTION

Should be called once prior to making any of the other channel access calls.

COMMENTS

Invoked once at first call to any ca_xxx() function if you forget.

RETURNS

"ECA_NORMAL - Normal successful completion" on page 38
"ECA_ALLOCMEM - Failed, unable to allocate memory" on page 38

SEE ALSO

"ca_task_exit()" on page 13

NAME ca_task_exit()

SYNOPSIS

```
#include      <cadef.h>
int           ca_task_exit();
```

ARGUMENTS **none**

DESCRIPTION Close channel access facility and free any resources allocated.

COMMENTS Performed automatically at task (or process) exit.

RETURNS "ECA_NORMAL - Normal successful completion" on page 38

SEE ALSO "ca_task_initialize()" on page 13

NAME ca_search()

SYNOPSIS

```
#include           <cadef.h>
int ca_search_and_connect (
  char             *CHANNEL_NAME;
  chid             *PCHID;
  void             (*USERFUNC)(struct connection_handler_args ARGS);
  void             *PUSER;
);
```

ALIASES

`intca_search(NAME,PCHID)`

ARGUMENTS

CHANNEL_NAME **R**

NULL terminated ASCII channel name string. ASCII channel names are of the form "<channel name>.[<field name>]". If the field name and the separator is omitted the ".VAL" field is assumed. For Example "RFHV01" and "RFHV01.VAL" reference the same channel.

PCHID **RW**

Channel identifier written into user supplied channel id pointed to by PCHID

USERFUNC **R**

Optional address of user subroutine to be run when the connection state changes. Casual users of channel access may wish to set this field to nill or 0 if they do not need to have a connection handler run.

PUSER **R**

The value of this argument is stored in a pointer sized user available field associated with the specified channel. See the MACROS manual page for reading and writing this field. Casual users of channel access may wish to set this field to nill or 0.

DESCRIPTION

This function requests that the CA library should attempt to establish and maintain a virtual circuit between the caller's application and a named variable in a CA server. The circuit may be initially connected or disconnected depending on the state of the network and the location of the channel. If there are sufficient resources to initiate this process then the routine writes a channel identifier into the application supplied variable pointed to by PCHID and returns ECA_NORMAL. This identifier can be used with all of the other channel access calls that operate on a channel.

A channel will only become connected after channel access is able to locate a channel access server that has access to the named variable and only if channel access is able to find a path through the network to this server. All of the channel access routines that send a request to a channel access server will refuse to operate on a channel that is disconnected. There are two ways to obtain asynchronous notification when a channel enters a connected state. The first and simplest method requires that you call ca_pend_io() prior to using a channel id returned by ca_search(). The second requires that you establish a connection handler by supplying a non-zero USERFUNC. This connection handler is called whenever the connection state of the channel changes. If you have installed a connection handler then ca_pend_io() will *not* block waiting for this channel's connection to occur.

COMMENTS

The MACRO ca_state(CHID) can be used to test the connection state of a channel. Valid connections may be isolated from invalid ones with this MACRO if ca_pend_io() times out.

All remote operation requests such as the above are accumulated (buffered) and not forwarded to the IOC until one of ca_flush_io(), ca_pend_io(), ca_pend_event()*, or ca_s-g_pend() are called allowing several operations to be efficiently sent over the network in one message.*

RETURNS

"ECA_NORMAL - Normal successful completion" on page 38
"ECA_BADTYPE - Invalid DBR_XXXX type" on page 38
"ECA_STRTOBIG - Unusually large string supplied" on page 38
"ECA_ALLOCMEM - Unable to allocate memory" on page 38
"ECA_GETFAIL - A local database get failed" on page 38

SEE ALSO

"ca_pend_io()" on page 22
"ca_change_connection_event()" on page 25
"macros" on page 28

NAME

ca_clear_channel()

SYNOPSIS

```
#include      <cadef.h>
int           ca_clear_channel (
  evid              CHID
);
```

ARGUMENTS

CHID **R**

identifier of the channel to close (returned by "ca_search()" on page 14).

DESCRIPTION

Close a channel created by ca_search().

All remote operation requests such as the above are accumulated (buffered) and not forwarded to the IOC until one of ca_flush_io(), ca_pend_io(), ca_pend_event()*, or ca_s-g_pend() are called allowing several operations to be efficiently sent over the network in one message.*

COMMENTS

Clearing a channel does not cause its disconnect handler to be called. Clearing a channel does remove any events (monitors) registered for that channel. If the channel is currently connected then resources are freed only some time after this request is flushed out to the server,

RETURNS

```
"ECA_NORMAL - Normal successful completion" on page 38
"ECA_BADCHID - Corrupted CHID" on page 38
```

SEE ALSO

"ca_add_event()" on page 20

NAME

ca_put()

SYNOPSIS

```
#include     <cadef.h>
int ca_array_put(
       chtype           TYPE,
       unsigned long    COUNT,
       chid             CHID,
       void             *PVALUE
)
```

ALIASES

```
intca_bput(chid CHID, char *PVALUE);
intca_rput(chid CHID, float *PVALUE);
intca_put(chtype TYPE, chid CHID, void *PVALUE);
```

ARGUMENTS

TYPE **R**

The external type of the value to be written. Conversion will occur if this does not match the native type. Specify one from the set of DBR_XXXX in db_access.h

COUNT **R**

Element count to be written to the specified channel. This must match the array pointed to by PVALUE.

CHID **R**

Channel identifier

PVALUE **R**

Pointer to a value or array of values provided by the application to be written to the channel.

DESCRIPTION

Write a value or array of values to a channel.

COMMENTS

All remote operation requests such as the above are accumulated (buffered) and not forwarded to the IOC until one of ca_flush_io(), ca_pend_io(), ca_pend_event()*, or ca_sg_pend() are called allowing several operations to be efficiently combined into one network frame.*

RETURNS

```
"ECA_NORMAL - Normal successful completion" on page 38
"ECA_BADCHID - Corrupted CHID" on page 38
"ECA_BADTYPE - Invalid DBR_XXXX type" on page 38
"ECA_BADCOUNT - Requested count larger than native element count" on
page 38
"ECA_STRTOBIG - Unusually large string supplied" on page 38
"ECA_PUTFAIL - A local database put failed" on page 38
```

SEE ALSO

"ca_flush_io()" on page 24

NAME

ca_put_callback()

SYNOPSIS

```
#include <cadef.h>
int ca_array_put_callback (
       chtype          TYPE,
       unsigned long   COUNT,
       chid            CHID,
       void            *PVALUE,
       void            (*PFUNC)(struct event_handler_args ARGS),
       void            *USERARG,
);
```

ARGUMENTS

TYPE **R**

The external type of the value to be written. Conversion will occur if this does not match the native type. Specify one from the set of DBR_XXXX in db_access.h

ARGUMENTS

COUNT **R**

Element count to be written to the specified channel. This must match the array pointed to by PVALUE.

CHID **R**

Channel identifier

PVALUE **R**

Pointer to a value or array of values provided by the application to be written to the channel.

PFUNC **R**

address of user subroutine to be run when the requested operation completes

USERARG **R**

pointer sized object passed to user function above

DESCRIPTION

Write a value (or array of values) to a channel and run the user supplied asynchronous call back only after the initiating put operation and all actions resulting from the initiating put operation complete.

COMMENTS

The structure"event_handler_args" is defined in the file ''cadef.h''.

All remote operation requests such as the above are accumulated (buffered) and not forwarded to the IOC until one of ca_flush_io(), ca_pend_io(), ca_pend_event(), or ca_sg_pend() are called allowing several operations to be efficiently combined into one network frame.

If a connection is lost and then resumed outstanding put requests are not automatically reissued following reconnect.

RETURNS

```
"ECA_NORMAL - Normal successful completion" on page 38
"ECA_BADCHID - Corrupted CHID" on page 38
"ECA_BADCOUNT - Requested count larger than native element count" on
page 38
"ECA_BADTYPE - Invalid DBR_XXXX type" on page 38
"ECA_BADTYPE - Invalid DBR_XXXX type" on page 38
"ECA_ALLOCMEM - Unable to allocate memory" on page 38
```

SEE ALSO

"ca_pend_event()" on page 23, "ca_flush_io()" on page 24

NAME

ca_get()

SYNOPSIS

```
#include <cadef.h>
int ca_array_get (
       chtype          TYPE,
       unsigned long   COUNT,
       chid            CHID,
       void            *PVALUE
);
```

ALIASES

```
intca_bget(chid CHID, char *PVALUE);
/* NOTE: bget fetches the binary value as a string */
intca_rget(chid *CHID, void *PVALUE);
intca_get(chtype TYPE, chid CHID, void *PVALUE);
```

ARGUMENTS

TYPE **R**

The external type of the value to be returned. Conversion will occur if this does not match the native type. Specify one from the set of DBR_XXXX in db_access.h

COUNT **R**

Element count to be read from the specified channel. Must match the array pointed to by PVALUE.

CHID **R**

Channel identifier

PVALUE **R**

Pointer to an application supplied buffer where the current value of the channel is to be written.

DESCRIPTION

Read a value from a channel.

COMMENTS

The returned channel value cant be assumed to be stable in the application supplied buffer until after ECA_NORMAL is returned from ca_pend_io().

See ca_get_callback() if it is important to know when an individual get completes.

All remote operation requests such as the above are accumulated (buffered) and not forwarded to the IOC until one of ca_flush_io(), ca_pend_io(), ca_pend_event()*, or ca_sg_pend() are called allowing several operations to be efficiently sent over the network in one message.*

If a connection is lost and then resumed outstanding get requests are not automatically reissued following reconnect

RETURNS

"ECA_NORMAL - Normal successful completion" on page 38
"ECA_BADTYPE - Invalid DBR_XXXX type" on page 38
"ECA_BADCHID - Corrupted CHID" on page 38
"ECA_BADCOUNT - Requested count larger than native element count" on page 38
"ECA_GETFAIL - A local database get failed" on page 38

SEE ALSO

"ca_pend_io()" on page 22
"ca_get_callback()" on page 19

ca_get_callback()

NAME

ca_get_callback()

SYNOPSIS

```
#include <cadef.h>
int ca_array_get_callback (
       chtype          TYPE,
       unsigned long   COUNT,
       chid            CHID,
       void            (*USERFUNC)(struct event_handler_args ARGS),
       void            *USERARG
);
```

ALIASES

```
intca_bget_callback(CHID, USERFUNC, USERARG);
intca_rget_callback(CHID, USERFUNC, USERARG);
intca_get_callback(TYPE, CHID, USERFUNC, USERARG);
```

ARGUMENTS

TYPE **R**

The external type of the returned value. Conversion will occur if this does not match the native type. Specify one from the set of DBR_XXXX in db_access.h

COUNT **R**

The element count to be read from the specified channel.

CHID **R**

Channel index

USERFUNC **R**

Address of user subroutine to be run when the requested operation completes.

USERARG **R**

Pointer sized object passed to user function above

DESCRIPTION

Read a value from a channel and run a callback when the operation completes.

COMMENTS

The structure "event_handler_args" is defined in the file ‘‘cadef.h’’.

All remote operation requests such as the above are accumulated (buffered) and not forwarded to the IOC until one of ca_flush_io(), ca_pend_io(), ca_pend_event()*, or ca_s-*

g_pend() are called allowing several operations to be efficiently sent over the network in one message.

Note however that ca_pend_io() will not wait for the delivery of values requested by ca_get_callback().

If a connection is lost and then resumed outstanding gets are not reissued.

RETURNS

"ECA_NORMAL - Normal successful completion" on page 38
"ECA_BADCHID - Corrupted CHID" on page 38
"ECA_BADCOUNT - Requested count larger than native element count" on page 38
"ECA_BADTYPE - Invalid DBR_XXXX type" on page 38
"ECA_GETFAIL - A local database get failed" on page 38
"ECA_ALLOCMEM - Unable to allocate memory" on page 38

SEE ALSO

"ca_clear_event()" on page 22
"ca_flush_io()" on page 24

NAME

ca_add_event()

SYNOPSIS

```
#include <cadef.h>
int ca_add_masked_array_event (
        chtype           TYPE,
        unsigned long    COUNT,
        chid             CHID,
        void             (*USERFUNC)(struct event_handler_args ARGS),
        void             *USERARG,
        evid             *PEVID,
        unsigned long    MASK
);
```

ALIASES

```
int ca_add_array_event(TYPE,COUNT,CHID,USERFUNC,USERARG,
                        (float)0, (float)0,(float)0,PEVID);
int ca_add_event(TYPE, CHID, USERFUNC, USERARG, PEVID);
```

ARGUMENTS

TYPE **R**

type of returned value- conversion will occur if it does not match native type. Specify one from the set of DBR_XXXX in db_access.h

COUNT **R**

element count to be read from the specified channel - a count of zero specifies the DB native count.

CHID **R**

channel index

USERFUNC **R**

address of user subroutine to be run when events occur

USERARG **R**

pointer sized object passed to user function above

PEVID **W**

pointer to user supplied event id which is initialized by this subroutine call. User's event id can later be used to clear a specific event. PEVID may be omitted by passing a NULL pointer.

MASK **R**

A mask with bits set for each of the event trigger types required. The event trigger mask must be a logical or of one or more of the following constants.
DBE_VALUE:
Trigger events when the channel value exceeds the monitor dead band
DBE_LOG:
Trigger events when the channel value exceeds the archival dead band
DBE_ALARM:
Trigger events when the channel alarm state changes.
For aliases which do not include a trigger specification events will be triggered when the channel's value exceeds the monitor dead band or when the channel's alarm state changes (that is DBE_VALUE | DBE_ALARM).

Arguments of the form "(float) 0)" are reserved for future use. Your code will remain upwardly compatible if you specify "(float) 0" as specified.

DESCRIPTION

Specify a subroutine to be run whenever significant changes occur to a channel. A significant change can be a change in the channels value, alarm status, or severity status. Events may be added and deleted from both connected and disconnected channels. The specified USERFUNC is always called once immediately after establishing each new connection with the process variable or immediately from within ca_add_event() if it is a local channel. If an event is added prior to channel connection the requested count will be set to the native count of the channel if the requested count is larger.

COMMENTS

The structure"event_handler_args" is defined in the file ''cadef.h''. *All remote operation requests such as the above are accumulated (buffered) and not forwarded to the IOC until one of* ca_flush_io(), ca_pend_io(), ca_pend_event()*, or ca_sg_pend() are called allowing several operations to be efficiently sent over the network in one message.*

If at any time after registering the event read access to the specified channel is lost then the call back will be called immediately, indicating that read access was lost with the status argument. When read access is restored normal event processing will resume.

RETURNS

"ECA_NORMAL - Normal successful completion" on page 38
"ECA_BADCHID - Corrupted CHID" on page 38
"ECA_BADTYPE - Invalid DBR_XXXX type" on page 38
"ECA_ALLOCMEM - Unable to allocate memory" on page 38
"ECA_ADDFAIL - A local database event add failed" on page 38

SEE ALSO

"ca_pend_event()" on page 23
"ca_flush_io()" on page 24

NAME

ca_clear_event()

SYNOPSIS

```
#include <cadef.h>
int     ca_clear_event(evid EVID);
```

ARGUMENTS

EVID **R**

event id returned by add_event()

DESCRIPTION

Remove a subroutine from a list of those run whenever significant changes occur on an IO channel. Event is disabled immediately. However resources are not recovered until the operation has completed in the server. *All remote operation requests such as the above are accumulated (buffered) and not forwarded to the IOC until one of* ca_flush_io(), ca_pend_io(), ca_pend_event()*, or ca_sg_pend() are called allowing several operations to be efficiently sent over the network in one message.*

COMMENTS

RETURNS

“ECA_NORMAL - Normal successful completion” on page 38
“ECA_BADCHID - Corrupted CHID” on page 38

SEE ALSO

“ca_add_event()” on page 20

NAME

ca_pend_io()

SYNOPSIS

```
#include <cadef.h>
int ca_pend_io(float TIMEOUT);
```

ARGUMENTS

TIMEOUT **R**

time out after TIMEOUT seconds (TIMEOUT of zero is forever).

DESCRIPTION

Flushes the send buffer and then waits until outstanding queries complete or the specified time out expires. At this time queries include calls to ca_get() and calls to ca_search() which don’t specify a connection handler. If ECA_TIMEOUT is returned then failure must be assumed for *all* outstanding queries. Operations can be reissued followed by another ca_pend_io(). This routine will only block on outstanding queries issued after the last call to ca_pend_io() or the start of program execution whichever occurs later in time. *If no queries are outstanding then ca_pend_io() will flush the send buffer and return immediately without processing any additional outstanding channel access labor.*

COMMENTS

Any values written into your program’s variables by a channel access synchronous request should not be referenced by your program until ECA_NORMAL has been received from ca_pend_io().

The delay specified to ca_pend_io() should be long enough to take care of what needs to be done taking into account worst case network delays. If you find that operations are timing out during situations when the network isn't broken then you may need to increase the specified time-out. If ca_pend_io() prematurely times out then the requests are lost and must be reissued.

This routine will handle channel access background activity while it is waiting.

RETURNS

"ECA_NORMAL - Normal successful completion" on page 38
"ECA_EVDISALLOW - Function inappropriate for use within an event handler" on page 39

SEE ALSO

"ca_get()" on page 18
"ca_search()" on page 14
"ca_test_io()" on page 23

NAME

ca_test_io()

SYNOPSIS

```
#include <cadef.h>
int     ca_test_io();
```

ARGUMENTS

none

DESCRIPTION

Determines if any queries are outstanding. This routine will only report the status of outstanding query operations issued after the last call to ca_pend_io() or the start of program execution whichever occurs later in time.

RETURNS

"ECA_IODONE - IO operations completed" on page 39
"ECA_IOINPROGRESS - Some IO operations still in progress" on page 39

SEE ALSO

"ca_pend_io()" on page 22

NAME

ca_pend_event()

SYNOPSIS

```
#include     <cadef.h>
int ca_pend_event(float TIMEOUT);
```

ARGUMENTS

TIMEOUT **R**

time out after TIMEOUT seconds.

DESCRIPTION

Flushes the send buffer and waits for asynchronous events for TIMEOUT seconds (TIMEOUT of zero is forever). This routine will not return before the time-out expires and all unfinished channel access labor has been processed.

COMMENTS

A very short TIMEOUT such as 0.0001 seconds will result in a poll (i.e. channel access will just check for file descriptor activity and then return).

This routine will handle channel access background activity while it is waiting. If it is called with a very short time-out it will dispatch channel access background backlog each time that ca_pend_io() is called.

RETURNS

`"ECA_NORMAL - Normal successful completion" on page 38`
`"ECA_TIMEOUT - The operation timed out" on page 38`
`"ECA_EVDISALLOW - Function inappropriate for use within an event handler" on page 39`

SEE ALSO

"ca_add_event()" on page 20

NAME

ca_flush_io()

SYNOPSIS

```
#include <cadef.h>
int ca_flush_io();
```

ARGUMENTS

none

DESCRIPTION

Send (flush) buffered IO requests. Normally performed by ca_pend_io() or ca_pend_event(). However, some users may wish to flush remote operation request messages early so they can perform operations in parallel with outstanding remote operations.

COMMENTS

Outstanding requests are also sent whenever the buffer which holds them becomes full.

RETURNS

`"ECA_NORMAL - Normal successful completion" on page 38`

SEE ALSO

NAME

ca_signal()

SYNOPSIS

```
#include <cadef.h>
int ca_signal(long CA_STATUS, char *CONTEXT_STRING);
```

ALIASES

```
void SEVCHK(CA_STATUS, CONTEXT_STRING)
```

ARGUMENTS

CA_STATUS R

The status (error code) returned from a channel access function.

CONTEXT_STRING R

A null terminated character string to print as error context.

DESCRIPTION

Print the error message character string associated with the supplied channel access error code and the supplied error context. If the error code indicates an unsuccessful operation a stack dump is printed, if this capability is available on the local operating system, and execution is terminated. SEVCHK is a macro envelope around ca_signal which only calls ca_signal() if the supplied error code indicates an unsuccessful operation. SEVCHK is the recommended error handler for those applications which do not wish to write code testing the status returned from each channel access call.

EXAMPLES

```
status = ca_task_initialize(...);
SEVCHK(status,"Unable to start the CA client");
```

COMMENTS

If the application only wishes to print the message associated with an error code or test the severity of an error there are MACROS provided for this purpose.

RETURNS

"ECA_NORMAL - Normal successful completion" on page 38

SEE ALSO

NAME

ca_change_connection_event()

SYNOPSIS

```
#include <cadef.h>
int  ca_change_connection_event (
       chid CHID,
       void  (*USERFUNC)(struct  connection_handler_argshandlerargs)
);
```

ARGUMENTS

CHID **R**

channel index

USERFUNC **R**

address of user subroutine to be run when the connection state changes

DESCRIPTION

This function installs a user supplied handler routine which will be run whenever an IO channel connects or disconnects.

COMMENTS

See the ca_search() manual page for a description of the preferred mechanism for accessing this functionality.

The structure "connection_handler_args" is defined in the file ''cadef.h''. Installing a null exception handler will cause the default handler to be reinstalled.

RETURNS

"ECA_NORMAL - Normal successful completion" on page 38

SEE ALSO

"ca_search()" on page 14

NAME

ca_add_exception_event()

SYNOPSIS

```
#include <cadef.h>
int  ca_add_exception_event(
       void  (*USERFUNC)(struct  exception_handler_argsHANDLERARGS),
       void  *USERARG
);
```

ARGUMENTS

USERFUNC R

address of user subroutine to be run when exceptions occur

USERARG R

pointer sized object passed to user function above

DESCRIPTION

Synchronous channel access status is returned as status from each channel access function (subroutine). However sometimes client initiated errors occur in the server asynchronously to the client thread. These errors are normally handled by printing them at the client's standard out. The routine documented here allows the application to replace the default asynchronous exception handler with a application specified function and argument. This allows the application to take corrective action to an exception when required.

COMMENTS

The structure"exception_handler_args" is defined in the file ‘‘cadef.h’’. Note that the contents of some arguments are invalid in the context of some error messages. For instance, a failed get will supply the address in the client task where the returned value was requested to be written. For other failed operations the value of the ADDR argument should not be used.

Installing a null exception handler will cause the default handler to be reinstalled.

More information needs to be provided about which arguments are valid with each exception. More information needs to be provided correlating exceptions with the routines which cause them.

```
/*
 *      Default Exception Handler
 */
LOCAL void ca_default_exception_handler(
                      struct exception_handler_args args)
{
        char *pName;

        if(args.chid){
                pName = ca_name(args.chid);
        }
        else{
                pName = "?";
        }

        /*
         * LOCK around use of sprintf buffer
         */
        LOCK;
        sprintf(sprintf_buf,
                "%s - with request chan=%s op=%d data type=%s count=%d",
                args.ctx,
                pName,
                args.op,
                dbr_type_to_text(args.type),
                args.count);
        ca_signal(args.stat, sprintf_buf);
        UNLOCK;
}
```

RETURNS

"ECA_NORMAL - Normal successful completion" on page 38

ca_replace_printf_handler ()

NAME

SYNOPSIS

```
#include <stdarg.h>
#include <cadef.h>
int     ca_replace_printf_handler(
              char        *pFormat;
              va_list     args;
        );
```

ARGUMENTS

PFORMAT **R**

This is the address of user supplied subroutine to be run when CA prints formatted text.

ARGS **R**

A variable argument list (see ANSI C 'stdargs.h') similar in format to the argument expected by ANSI C 'vfprintf()'.

DESCRIPTION

This entry point is provided for applications that are particular about how CA error messages are handled. By default CA error messages are sent to 'stderr'.

EXAMPLES

```
#include <cadef.h>

status = ca_replace_printf_handler (my_printf);
SEVCHK (status, "failed to install my printf handler");

/*
 *      my_printf()
 */
int my_printf(char *pformat, va_list args)
{
        int             status;

        status = vfprintf(
                        stderr,
                        pformat,
                        args);
        return status;
}
```

COMMENTS

'cadef.h' includes 'stdarg.h' if it is compiled by an ANSI compiler. Installing a NULL handler will cause the default handler to be reinstalled.

RETURNS

"ECA_NORMAL - Normal successful completion" on page 38

ca_replace_access_rights_event()

NAME

SYNOPSIS

```
#include <cadef.h>
int  ca_replace_access_rights_event(
       chid CHAN,
       void (*PFUNC)(struct access_rights_handler_args ARGS)
);
```

ARGUMENTS

CHAN **R**

address of user subroutine to be run when exceptions occur

PFUNC **R**

address of user supplied call back function

DESCRIPTION

Replace the existing access rights state change asynchronous call back handler.

COMMENTS

The call back is called whenever CA connects to a process variable immediately before CA calls the applications connection handler. The call back is called whenever CA disconnects from a process variable immediately after the disconnect call back is called. The call back is always called once immediately after installation if the process variable is already connected. The call back is called when ever the access rights state of connected channel changes.

The structure"access_rights_handler_args" is defined in the file ‘‘cadef.h’’. A nill function pointer is initially installed and can be used to disable an existing handler installation.

RETURNS

"ECA_NORMAL - Normal successful completion" on page 38

SEE ALSO

"macros" on page 28
"ca_modify_user_name()" on page 31
"ca_modify_host_name()" on page 31

NAME

macros

```
#include <cadef.h>
chtype  ca_field_type(CHID)
unsigned ca_element_count(CHID)
char *  ca_name(CHID)
void *  ca_puser(CHID)
enum channel_stateca_state(CHID)
char *  ca_message(STATUS)
char *  ca_host_name(CHID)
int     ca_read_access(CHID)
int     ca_write_access(CHID)
#include<db_access.h>
unsigned dbr_size_n(TYPE,COUNT)
unsigned dbr_size[TYPE]
unsigned dbr_value_size[TYPE]
```

ARGUMENTS

CHID **R**

channel index

STATUS

status returned from a channel access function.

DESCRIPTION

The MACRO ca_field_type() returns the native type for the specified IO channel. This type will be one from the set of DBF_XXXX in db_access.h. The MACRO ca_element_count() returns the native array element count for the specified IO channel. The MACRO ca_name() returns the name provided as an argument to ca_search when the supplied channel id was created. The MACRO ca_puser() returns or sets a pointer sized region available with each channel for use at the users discretion. The MACRO ca_state() returns an enumerated value indicating the current state of the specified IO channel. See cadef.h for a description of each of the potential channel states. The

MACRO ca_message() returns a message character string corresponding to a status code provided by channel access. The MACRO ca_host_name() returns a pointer to a character string which contains the name of the host to which a channel is currently connected. The macro ca_read_access() returns TRUE if the client currently has read access to the specified channel and FALSE otherwise. The macro ca_write_access() returns TRUE if the client currently has write access to the specified channel and FALSE otherwise.

The MACRO dbr_size_n(TYPE, COUNT) returns the size in bytes for a DBR_XXXX with n elements. The Array dbr_size[TYPE] returns the size in bytes for a DBR_XXXX. The Array dbr_value_size[TYPE] returns the size in bytes for the value of a DBR_XXXX.

EXAMPLES

```
struct  users_data mydata;
struct users_data *pmydata;
enum channel_statemystate;
type = ca_field_type(waveform_chid);
nelements = ca_element_count(waveform_chid);
status = ca_add_event(...)
printf("the  operation completed with status: %s\n",ca_message(status));
ca_puser(chid) = (void *) &mydata;
pmydata = (struct users_data *) ca_puser(chid)
if(pmydata->notdone)try_again();
mystate = ca_state(chid);
printf("The current connection is %s\n", ca_host_name(chid));
if(ca_write_access(chid) && ca_read_access(chid)){
}
```

COMMENTS

SEE ALSO

cadef.h, caerr.h, db_access.h

NAME

ca_test_event()

```
#include <cadef.h>
void ca_test_event(
  struct  event_handler_args HANDLERARGS
}
```

ARGUMENTS

HANDLERARGS **R**

Arguments passed to all channel access event handlers.

DESCRIPTION

A canned event handler for debugging purposes. ca_test_event() prints information about the event and then returns.

EXAMPLES

```
void ca_test_event();
status = ca_add_event(type, chid, ca_test_event, NULL, NULL);
SEVCHK(status, " .... ");
```

COMMENTS

SEE ALSO

ca_add_event()

NAME

ca_add_fd_registration()

```
#include <cadef.h>
int  ca_add_fd_registration(
  void  (USERFUNC*)(void *USERARG, int FD, int OPENED),
  void  *USERARG
)
```

ARGUMENTS

USERFUNC **R**

Pointer to a user supplied C function returning null with the above arguments.

USERARG **R**

User supplied pointer sized variable passed to the above function.

FD **R**

A file descriptor.

OPENED **R**

Boolean argument is true if the file descriptor was opened and false if the file descriptor was closed.

DESCRIPTION

For use with the services provided by a file descriptor manager (IO multiplexor) such as "”fdmgr.c”. A file descriptor manager is often needed when two file descriptor IO intensive libraries such as the EPICS channel access client library and the X window system client library must coexist in the same UNIX process.

This function allows an application code to be notified whenever the CA client library places a new file descriptor into service and whenever the CA client library removes a file descriptor from service.

Specifying USERFUNC=NULL disables file descriptor registration (this is the default).

EXAMPLES

```
int s;
static struct myStruct aStruct;
void fdReg(pStruct, fd, opened)
struct myStruct*pStruct;
intfd;
intopened;
{
  if(opened)printf(”fd %d was opened\n”, fd);
  else printf(”fd %d was closed\n”, fd);
}
s = ca_add_fd_registration(fdReg, &aStruct);
SEVCHK(s, NULL);
```

COMMENTS

When using this function it is advisable to call it only once prior to calling any other CA function.

RETURNS

“ECA_NORMAL - Normal successful completion” on page 38

SEE ALSO

NAME

ca_modify_user_name()

```
#include <cadef.h>
int  ca_modify_user_name(char *PUSERNAME);
```

ARGUMENTS

PUSERNAME **R**

Pointer to a user supplied user name string.

DESCRIPTION

Replace the default user name communicated to the CA server for this client.

EXAMPLES

```
int status;
status = ca_modify_user_name("Flintstone");
SEVCHK(status,NULL);
```

COMMENTS

RETURNS

"ECA_NORMAL - Normal successful completion" on page 38
"ECA_STRTOBIG - Unusually large string supplied" on page 38
"ECA_ALLOCMEM - Failed, unable to allocate memory" on page 38

SEE ALSO

"ca_replace_access_rights_event()" on page 27
"macros" on page 28

NAME

ca_modify_host_name()

```
#include <cadef.h>
int  ca_modify_host_name(char *PHOSTNAME);
```

ARGUMENTS

PUSERNAME **R**

Pointer to a user supplied host name string.

DESCRIPTION

Replace the default host name communicated to the CA server for this client.

EXAMPLES

```
int status;
status = ca_modify_host_name("Bedrock");
SEVCHK(status,NULL);
```

COMMENTS

RETURNS

"ECA_NORMAL - Normal successful completion" on page 38
"ECA_STRTOBIG - Unusually large string supplied" on page 38
"ECA_ALLOCMEM - Unable to allocate memory" on page 38

SEE ALSO

"ca_replace_access_rights_event()" on page 27
"macros" on page 28

NAME

ca_sg_create()

```
#include <cadef.h>
int  ca_sg_create(
  CA_SYNC_GID *PGID
);
```

ARGUMENTS

PGID **RW**

Pointer to a user supplied CA_SYNC_GID.

DESCRIPTION

Create a synchronous group and return an identifier for it. A synchronous group can be used to guarantee that a set of channel access requests have completed. Once a synchronous group has been created then channel access get and put requests may be issued within it using ca_sg_get() and ca_sg_put() respectively. The routines ca_sg_block() and ca_sg_test() can be used to block for and test for completion respectively. The routine ca_sg_reset() is used to discard knowledge of old requests which have timed out and in all likelihood will never be satisfied.

EXAMPLES

```
CA_SYNC_GID  gid;
status = ca_sg_create(&gid);
SEVCHK(status, "Sync group create failed");
```

COMMENTS

Any number of asynchronous groups can have application requested operations outstanding within them at any given time.

RETURNS

"ECA_NORMAL - Normal successful completion" on page 38
"ECA_ALLOCMEM - Failed, unable to allocate memory" on page 38

SEE ALSO

"ca_sg_delete()" on page 32
"ca_sg_block()" on page 33
"ca_sg_test()" on page 34
"ca_sg_reset()" on page 34
"ca_sg_put()" on page 35
"ca_sg_get()" on page 36

NAME

ca_sg_delete()

```
#include <cadef.h>
int  ca_sg_delete(
  CA_SYNC_GID GID
);
```

ARGUMENTS

GID **R**

Identifier of the synchronous group to be deleted.

DESCRIPTION

Deletes a synchronous group.

EXAMPLES

```
CA_SYNC_GID  gid;
status = ca_sg_delete(gid);
SEVCHK(status, "Sync group delete failed");
```

COMMENTS

RETURNS

"ECA_NORMAL - Normal successful completion" on page 38
"ECA_BADSYNCGRP - Invalid synchronous group" on page 39

SEE ALSO

"ca_sg_create()" on page 32

NAME

ca_sg_block()

```
#include <cadef.h>
int  ca_sg_block (
  CA_SYNC_GID GID,
  float      timeout
)
```

ARGUMENTS

GID **R**

Identifier of the synchronous group.

DESCRIPTION

Flushes the send buffer and then waits until outstanding requests complete or the specified time out expires. At this time outstanding requests include calls to ca_sg_array_get() and calls to ca_sg_array_put(). If ECA_TIMEOUT is returned then failure must be assumed for *all* outstanding queries. Operations can be reissued followed by another ca_sg_block(). This routine will only block on outstanding queries issued after the last call to ca_sg_block(), ca_sg_reset(), or ca_sg_create() whichever occurs later in time. *If no queries are outstanding then ca_sg_block() will return immediately* ***without*** *processing any additional outstanding channel access labor.*

EXAMPLES

```
CA_SYNC_GID gid;
status = ca_sg_block(gid);
SEVCHK(status, "Sync group block failed");
```

COMMENTS

Any values written into your program's variables by a channel access synchronous group request should not be referenced by your program until ECA_NORMAL has been received from ca_sg_block().

This routine will handle channel access background activity while it is waiting.

RETURNS

"ECA_NORMAL - Normal successful completion" on page 38
"ECA_TIMEOUT - The operation timed out" on page 38
"ECA_EVDISALLOW - Function inappropriate for use within an event handler" on page 39
"ECA_BADSYNCGRP - Invalid synchronous group" on page 39

SEE ALSO

"ca_sg_test()" on page 34
"ca_sg_reset()" on page 34

NAME

ca_sg_test()

```
#include <cadef.h>
int  ca_sg_test (
  CA_SYNC_GID GID
)
```

ARGUMENTS

GID **R**

Identifier of the synchronous group.

DESCRIPTION

Test to see if all requests made within a synchronous group have completed.

EXAMPLES

```
CA_SYNC_GID gid;
status = ca_sg_test(gid);
```

COMMENTS

RETURNS

"ECA_IODONE - IO operations completed" on page 39
"ECA_IOINPROGRESS - Some IO operations still in progress" on page 39

SEE ALSO

NAME

ca_sg_reset()

```
#include <cadef.h>
int  ca_sg_reset(
  CA_SYNC_GID GID
)
```

ARGUMENTS

GID **R**

Identifier of the synchronous group.

DESCRIPTION

Reset the number of outstanding requests within the specified synchronous group to zero so that ca_sg_test() will return **ECA_IODONE** and ca_sg_block() will not block unless additional subsequent requests are made**.**

EXAMPLES

```
CA_SYNC_GID  gid;
status = ca_sg_reset(gid);
```

COMMENTS

RETURNS

"ECA_NORMAL - Normal successful completion" on page 38
"ECA_BADSYNCGRP - Invalid synchronous group" on page 39

SEE ALSO

NAME

ca_sg_put()

```
#include <cadef.h>
intca_sg_array_put(
  CA_SYNC_GID GID,
  chtype      TYPE,
  unsigned longCOUNT,
  chid        CHID,
  void        *PVALUE
)
```

ARGUMENTS

GID **R**

Identifier of the synchronous group.

TYPE **R**

type of supplied value- conversion will occur if it does not match native type. Specify one from the set of DBR_XXXX in db_access.h

COUNT **R**

element count to be written to the specified channel (matches the array pointed to by PVALUE)

CHID **R**

channel index

PVALUE **R**

A pointer to an application supplied buffer containing the value or array of values

DESCRIPTION

Write a value (or array of values) to a channel while updating the outstanding request count of a synchronous group.

COMMENTS

All remote operation requests such as the above are accumulated (buffered) and not forwarded to the IOC until one of ca_flush_io(), ca_pend_io(), ca_pend_event(), or ca_sg_pend() are called allowing several operations to be efficiently sent in one network frame.

RETURNS

"ECA_NORMAL - Normal successful completion" on page 38
"ECA_BADSYNCGRP - Invalid synchronous group" on page 39
"ECA_BADCHID - Corrupted CHID" on page 38
"ECA_BADTYPE - Invalid DBR_XXXX type" on page 38
"ECA_BADCOUNT - Requested count larger than native element count" on page 38
"ECA_STRTOBIG - Unusually large string supplied" on page 38
"ECA_PUTFAIL - A local database put failed" on page 38

SEE ALSO

ca_flush_io()

NAME

ca_sg_get()

```
#include <cadef.h>
intca_sg_array_get(
  CA_SYNC_GID GID,
  chtype      TYPE,
  unsigned longCOUNT,
  chid        CHID,
  void        *PVALUE
)
```

ARGUMENTS

GID **R**

Identifier of the synchronous group.

TYPE **R**

External type of returned value. Conversion will occur if this does not match native type. Specify one from the set of DBR_XXXX in db_access.h

COUNT **R**

Element count to be read from the specified channel. Must match the array pointed to by PVALUE)

CHID **R**

channel index

PVALUE **R**

Pointer to application supplied buffer that is to contain the value or area of values to be returned

DESCRIPTION

Read a value from a channel while updating the outstanding request count of a synchronous group.

COMMENTS

The process variable values written into your program's variables by a channel access synchronous request should not be referenced by your program until ECA_NORMAL has been received from ca_sg_block() or until ca_sg_test() returns ECA_IODONE.

All remote operation requests such as the above are accumulated (buffered) and not forwarded to the IOC until one of ca_flush_io(), ca_pend_io(), ca_pend_event(), or ca_sg_pend() are called allowing several operations to be efficiently sent in one network frame.

If a connection is lost and then resumed outstanding gets are not reissued.

RETURNS

"ECA_NORMAL - Normal successful completion" on page 38
"ECA_BADSYNCGRP - Invalid synchronous group" on page 39
"ECA_BADCHID - Corrupted CHID" on page 38
"ECA_BADCOUNT - Requested count larger than native element count" on page 38
"ECA_BADTYPE - Invalid DBR_XXXX type" on page 38
"ECA_GETFAIL - A local database get failed" on page 38

SEE ALSO

"ca_pend_io()" on page 22
"ca_flush_io()" on page 24
"ca_get_callback()" on page 19

NAME

ca_import()

```
int ca_import(TASKID);
int TASKID;
```

ARGUMENTS

TASKID **R**

vxWorks task identifier.

DESCRIPTION

Each vxWorks task which calls channel access receives a channel access context. This context is shared by the calling task and several additional satellite tasks created to handle channel access events. Normally channel identifiers created in a context cant be used from other vxWorks tasks. Within an IOC a group of cooperating channel access client tasks all use the same channel access client context. Using the same context implies that all of the cooperating tasks use the same TCP/IP connections and other internal resources to access remote servers. It is sometimes necessary to create an additional task that will run in parallel with an existing channel access context. If so you must call ca_import() from this new task with the task identifier of one of the tasks in the desired channel access context prior to calling any channel access functions.

COMMENTS

Available on vxWorks only.

EXAMPLES

```
void new_task();

sp(new_task,  taskIdSelf());
.
.

void  new_task(parent)
int parent;
{
  SEVCHK(ca_import(parent), NULL);
.
.
.
}
```

RETURNS

"ECA_NORMAL - Normal successful completion" on page 38
"ECA_NOCACTX - Import from a task without a CA context failed" on page 39

SEE ALSO

"ca_import_cancel()" on page 38

NAME

ca_import_cancel()

```
int ca_import_cancel(TASKID);
int TASKID;
```

ARGUMENTS

TASKID **R**

vxWorks task identifier.

DESCRIPTION

Reverses the effect of ca_import().

EXAMPLES

```
SEVCHK(ca_import_cancel(taskIdCurrent), NULL);
```

COMMENTS

Available on vxWorks only.

RETURNS

"ECA_NORMAL - Normal successful completion" on page 38
"ECA_NOCACTX - Import from a task without a CA context failed" on page 39

SEE ALSO

"ca_import()" on page 37

NAME

ca_channel_status()

```
int ca_channel_status(TASKID)
int TASKID;
```

ARGUMENTS

TASKID **R**

vxWorks task identifier.

DESCRIPTION

Available on vxWorks only. Prints status for each channel in use by the specified vxWorks task. Can be run from the vxWorks shell.

COMMENTS

RETURNS

"ECA_NORMAL - Normal successful completion" on page 38

3.1 Return Codes

ECA_NORMAL -	Normal successful completion
ECA_ALLOCMEM -	Failed, unable to allocate memory
ECA_BADTYPE -	Invalid DBR_XXXX type
ECA_STRTOBIG -	Unusually large string supplied
ECA_ALLOCMEM -	Unable to allocate memory
ECA_BADCHID -	Corrupted CHID
ECA_BADCOUNT -	Requested count larger than native element count
ECA_PUTFAIL -	A local database put failed
ECA_GETFAIL -	A local database get failed
ECA_ADDFAIL -	A local database event add failed
ECA_TIMEOUT -	The operation timed out

ECA_EVDISALLOW - Function inappropriate for use within an event handler
ECA_IODONE - IO operations completed
ECA_IOINPROGRESS - Some IO operations still in progress
ECA_BADSYNCGRP - Invalid synchronous group
ECA_NOCACTX - Import from a task without a CA context failed