CaWave User's Guide
(by Ben-chin Cha)

CaWave User’s Guide explains how to use the CaWave functions which were specifically written in PV–WAVE command language and C language for EPICS users. CaWave consists of a special set of external channel access functions which provides the PV–WAVE users with easy and flexible access of channel information across the IOC networks. It also provides a complete set of process variable event monitoring functions.

This document also gives examples how a PV–WAVE user can interface to channel access devices. It is assumed that the user is already familiar with using PV–WAVE[1]. Few simple example modules of using PV–WAVE command language with CaWave functions are also given in this document.

1. External CaWave Link Modules

Two types of modules are defined in CaWave: function and procedure. The function module returns a value or an array of values to PV–WAVE. The procedure module returns nothing. To call a CaWave function a user has to use the ‘print’ value command or define a PV–WAVE variable equal to the return value of the called function. To call a CaWave procedure a user can directly type in the command.

The functions defined in CaWave conform to the following naming convention: all function names begin with the prefix “Ca”, and functions operating on a list of channel names end with the string “List”. In general functions operating on a single channel return a single value and functions operating on a list of channel names return a list of values.

The module names used in PV–WAVE CL version is not case sensitive, therefore a user can use lower case, upper case or mixed case. However, the string names or string values entered for parameter or argument are case sensitive. The string used in CaWave is exact the same as user entered.

1.1 Overview

The available CaWave link functions can be grouped into four categories: functions operating on a single channel, functions operating on a list of channels, monitoring functions operating on a single or multiple channels, and general functions for debugging or error handling. A user must make sure that the channel name entered is defined in the database and the IOC is operating normally when using values returned by these functions in his control program.

The single channel functions include CaGet, CaPut, CaGetString, CaPutString, CaGetValue, CaInfo, CaStatus, CaGetStatus, CaGetCount, CaGetWF, and CaPutWF. They can be used interactively to get a value back for a single channel or to set the value of a specified channel name.

The functions operating on a list of multiple channels include CaGetList, CaPutList, CaGetStringList, CaPutStringList, CaInfoList, CaStatusList, CaGetStatusList, and CaGetValueList. These functions can be used to get a list of values back for specified devices or to set values for a specified list of devices.
The value change event monitoring functions include \texttt{CaAddMonitor}, \texttt{CaWaitEventMonitor}, \texttt{CaEventMonitor}, \texttt{CaGetMonitor}, and \texttt{CaClearMonitor} for operating on a single channel and \texttt{CaAddMonitorList}, \texttt{CaWaitEventMonitorList}, \texttt{CaEventMonitorList}, \texttt{CaGetMonitorList}, and \texttt{CaClearMonitorList} for operating on a list of multiple channel names. They give the user control of adding/removing/getting the value change status of the channels to be monitored.

The general functions include \texttt{Ca}, \texttt{CaHelp}, \texttt{CaEvent}, \texttt{CaSleep}, \texttt{CaClock}, \texttt{CaDebug}, \texttt{CaPendIO}, \texttt{CaPendIOLIST}, \texttt{CaPendEvent}, \texttt{CaFlushEvent}, \texttt{CaError}, \texttt{CaErrortList}, and \texttt{CaSearchList}.

The channel name referred to in this document is the same as the process variable name defined in the DCT[3] database. The input channel name should be an ASCII string with a maximum string length less than 28.

When channel access cannot locate the process variable requested by a user, the device not found warning message is always displayed on the user’s terminal screen. Any channel not found from the IOC network is automatically cleared from the channel access search queue.

The channel access default pend IO timeout used is 0.3 second for a single channel, and 30 seconds for a list of channel names; and the default pend EVENT timeout used is 0.01 second. These timeouts can be reset by the user.

The available procedure and function modules for CaWave, their functionality, and their syntax are described in following sections.

1.2 On-Line Help Commands

To get on-line help for a CaWave function module just enter the function name itself. To get on-line help for a CaWave procedure module just enter the procedure name followed with the ,\texttt{help} keyword.

\texttt{CaHelp}
This module displays the general help information about the CaWave modules.

\texttt{Ca}
This module displays the available channel access related functions defined in CaWave. It returns nothing.

\texttt{Caevent}
This module lists the available channel access event related functions defined for CaWave. It returns nothing.

1.3 General / Debug / Error Handling Modules

\texttt{CaSleep}, \texttt{time}
This procedure causes the process to sleep for the specified time period. Time is measured in real seconds. Time can be any real number, e.g. 0.5. It returns nothing.
CaPendIO, time
This procedure allows the user to reset the TIMEOUT seconds for `ca_pend_io` function. It is used in processing a single channel information. The default timeout is 0.3 second; a TIMEOUT of zero is forever. It returns nothing.

CaPendIOList, time
This procedure allows the user to reset the TIMEOUT seconds for processing a list of channels. The default timeout is 30 seconds; a TIMEOUT of zero is forever. It returns nothing.

CaPendEvent, time
This procedure allows the user to reset the TIMEOUT seconds for `ca_pend_event` function. The default TIMEOUT is 0.01 second; a TIMEOUT of zero is forever. It returns nothing.

CaDebug, i
This procedure sets the runtime printing flag. The default value is initially set to 0 when CaWave is installed. If i is set to a positive integer, more debug information about the channel access function is printed. To debug the value change event, the value of i should be set to 2. The printing flag can be reset to 0 when no more debug information is desired. It returns nothing.

CaClock()
This function returns the elapsed CPU time used in micro seconds. It takes no input argument and returns a long integer number. It returns 0 when first time calls this function.

CaError()
This function queries the error status of the last execution of any channel access command. CaError takes no argument and returns 0 if no channel access error encountered in the last function call; otherwise it returns -1.

CaErrorList(n_list)
The CaErrorList expects a list of channel names as input. It checks for channel access errors for each name entered in the input list. It returns a list of the error codes corresponding to the requested channel names. The return list error codes can be zero or negative. An error code of 0 indicates that the last channel access call completed successfully. An error code of -1 or -2 indicates either that the channel device name was not found across the IOC network or another type of channel access error has occurred (most likely the timeout).

CaSearchList(n_list)
The CaSearchList requires a list of channel names as input. It returns a single value: 0 if all channels in the list were found or -1 if any one of the channels was not found on the IOC network. If -1 is returned, the user can immediately call CaErrorList to find out which channel was not found on the IOC network.

This function can be called at the very beginning to make sure every channel name in user's program exists across the IOC network.
CaFlushEvent

The CaFlushEvent module coupled with CaDebug,2 can be used to check the new events of the monitored list. This function flushes the pending channel access monitored events.

1.4 Single Channel Access Functions

Below gives the functions operated on a single device name. The device name is the same as the process variable name defined in DCT database. A legal name can also be a process variable name appended by '.' and the DCT supported field name. e.g. 'pvname.SCAN'.

A user has to insure that the return values are correct in his procedure module when calling following functions. CaWave always prints detected error information on the user terminal session. At the interactive mode a user can tell the integrity of the return values right away. At the programming mode a user has to use the return code or the error detection functions to check the integrity before using them. The CaError and CaErrorList functions should be used to guarantee the integrity of the return values.

CaGetValue ( 'name' )

To query the numerical value of a channel use the CaGetValue function. The CaGetValue function expects an IOC channel name defined in the database from the user and returns to PV-WAVE the current value of the channel.

CaGet ( 'name' )

Use the CaGet function to query the value, status, and severity of a channel. CaGet expects an IOC channel name defined in the database from the user and returns a list of three double precision values to PV-WAVE. The return double precision list consists of value, status, and severity of the channel.

CaPut ( 'name' , value )

To set the numerical value of a channel use the CaPut function. This function expects two input arguments, channel name and new value. It returns 0 if completed normally, else returns -1. The set value can be real or integer. The synonymous function name CaPutValue can be used for CaPut.

CaGetString ( 'name' )

For some channels the value field is defined in the database as an ASCII string rather than a numerical value. In these cases the function CaGetString should be used to get string value for these channels. The CaGetString function expects an input channel name as an argument and returns a string value for the channel.

CaPutString ( 'name' , 'string_value' )

For those channels where the value field is defined as an ASCII string rather than a numerical value, function CaPutString should be used to put the string value to the record field. CaPutString
expects two input arguments: channel name and new string value. It returns 0 if completed normally, else returns -1.

CaInfo ( 'name' )

The CaInfo function expects a channel name as an input argument. It returns a string of text for the querying device. It consists of the channel name, value, status, severity, operating ranges, and units.

CaGetWF ( 'name' )

The CaGetWF function expects a channel name as an input argument. It returns a double precision array of values for the waveform record. The size of the returned list is the same as the size of the waveform record defined in the IOC database.

CaPutWF ( 'name', v_list )

The CaPutWF function expects input consisting of channel name and a list of values which should not exceed the size of the waveform record. It writes the input value list to the specified waveform record to the IOC. It returns 0 if completed normally, else returns -1.

CaGetCount ( 'name' )

The CaGetCount function expects a channel name as an input argument. It returns the native element count for the specified channel from the IOC network.

CaStatus ( 'name' )

The CaStatus function expects a channel name as an input argument. It returns the status of the specified channel from the current internal data structure which is populated from the previous channel access call.

CaGetStatus ( 'name' )

The CaGetStatus function expects a channel name as an input argument. It updates the internal data structure and returns the new status of the specified channel.

1.5 Multiple Channels Access Functions

Below gives the functions operated on a list of multiple device names.

When calling following functions a user has to insure that the return values are correct before using them in his program module. CaWave always prints detected error information on the user terminal. At the interactive mode a user can tell the integrity of the return values right away. At the programming mode a user has to use the error detection functions to check the integrity before using them. The CaError and CaErrorList functions should be used to guarantee the integrity of the return values.

CaGetValueList ( [ 'name1', 'name2', ... ] )

The CaGetValueList function expects a list of channel names as input. It returns a list of double precision values corresponding to the requested channels.
CaGetList ([ 'name1', 'name2', ... ])

The CaGetList function expects a list of channel names as input. It returns a list of double precision values which contains [value1, status1, severity1, value2, status2, severity2, ...] corresponding to the requested channels.

CaPutList ([ 'name1', 'name2', ... ], [vl1, vl2, ...])

CaPutList expects two lists as input: a list of channel names, and a list of new values to be assigned to the channels. CaPutList write the list of the set values corresponding to the requested channels to the IOC. The input values can be real or integer. It returns 0 if it completed normally, otherwise returns -1.

CaGetStringList ([ 'name1', 'name2', ... ])

For some channels the value field is defined as an ASCII string rather than a numerical value. Function CaGetStringList is used to get a string value for these channels. CaGetStringList expects a list of channel names as input and returns a list of string values for the requested channel names.

CaPutStringList ([ 'nm1', 'nm2', ... ], ['vl1', 'vl2', ...])

For some channels the value field is defined as an ASCII string rather than a numerical value. Function CaPutStringList is used to assign a string value to the record field for these channels. It expects two input arguments: channel name list and new string value list. It returns 0 if it completed normally, otherwise returns -1.

CaInfoList ([ 'nm1', 'nm2', ... ])

The CaInfoList function expects a list of channel names as the input argument and returns a list of double precision values corresponding to the requested channel names. In addition, the channel name, value, status, severity, units, operating ranges, and units are displayed on the user’s terminal for all the requested channel names.

CaStatusList ([ 'nm1', 'nm2', ... ])

CaStatusList expects a list of channel names as input and returns a list of the current status codes corresponding to the requested channel device names from the internal data structure populated from the previous channel access call.

CaGetStatusList ([ 'nm1', 'nm2', ... ])

CaGetStatusList expects a list of channel names as input and returns a list of the current status codes corresponding to the requested channel device names through issuing new channel access calls.

1.6 Channel Access Event Monitoring Functions

CaAddMonitor ( 'name' )

The CaAddMonitor function expects a channel name as the input argument. It adds the specified channel name to the monitored list and automatically adds a change of connection event for the monitored channel. It returns 0 at normal completion, else returns -1.
A new event is generated immediately by adding the event, after each writing of a value to the channel, and after each new connection with the channel's current value.

**CaWaitEventMonitor ( wtime, 'name' )**

The `CaWaitEventMonitor` function expects two input arguments: the specified waiting time in seconds for events to happen, and the channel name of the interested monitored channel. This function returns 0 if any event of the value change happened within the specified time interval and -1 if no new value change event happened within the specified time interval since the last event procession.

**CaEventMonitor ( 'name' )**

The `CaEventMonitor` expects a channel name as the input argument. It queries the status of the value change event for the specified channel name. It returns 1 if the specified channel has never been processed and has undergone at least one value change, and returns 0 if there is no new value change event or if the status of changed value already been processed by any event procession function. The status of the changed event is reset to 0 by `CaEventMonitor`.

The event processing functions for a single device include: **CaEventMonitor, CaGetMonitor, and CaGetMonitorValue**.

**CaGetMonitorValue ( 'name' )**

The `CaGetMonitorValue` function expects a channel name as the input argument. It gets the current value from the data structure for the specified monitored channel name. The status of the changed event is reset to 0 by `CaGetMonitorValue`.

**CaGetMonitor ( 'name' )**

The `CaGetMonitor` function expects a channel name as the input argument. It returns a double precision list of three numbers: value, status, severity from the data structure for the specified monitored channel name. The status of the changed event is reset to 0 by `CaGetMonitor`.

**CaClearMonitor ( 'name' )**

The `CaClearMonitor` function expects a channel name as the input argument. It removes the specified channel name from the event monitored list. It returns 0 at normal completion, else returns -1.

**CaConnect ( 'name' )**

The `CaConnect` function expects a channel name as the input argument. It checks for the reconnection event of a specified channel. When reconnection happens, a user must wait for reestablishment of channel access in order to guarantee the normal operation of channel access functions. The connectivity event is automatically added for the monitored channel. It returns 0 at normal completion, else returns -1.

**CaAddMonitorList ( [ 'nm1', 'nm2', . . . ] )**

The `CaAddMonitorList` function expects a list of channel names as the input argument. It adds the specified channel names to the monitored list and automatically adds change of connection events for the monitored channels. It returns 0 at normal completion, else returns -1.
For each newly-added monitored channel, a new event is generated immediately by after adding the event, after each writing of a value to the channel, and after each new connection with the channel’s current value.

**CaWaitEventMonitorList ( wtime, [ 'nm1', 'nm2', ... ] )**

The CaWaitEventMonitorList expects two input arguments: the specified waiting time in seconds for events to happen, and the list of channel names of the interested monitored list. This function returns 0 if any event of the value change in the name list happened within the specified time interval and -1 if no value change event happened within the specified time interval.

**CaEventMonitorList ( [ 'nm1', 'nm2', ... ] )**

The CaEventMonitorList function expects a list of monitored channel names as the input argument. It queries for the event status of the specified channel names. It returns a list of event statuses corresponding to the specified list of channel names. The elements of the returned list are either 1 or 0. The returned element is 1 if the value of the specified channel has changed and it has not been processed yet. The returned element is 0 if either the value change has been processed by any event procession function or the value has not been changed. The list of status of the changed event is reset to 0 by CaEventMonitorList.

The event processing functions for a list of devices include: CaEventMonitorList, CaGetMonitorList, and CaGetMonitorValueList.

**CaGetMonitorValueList ( [ 'nm1', 'nm2', ... ] )**

The CaGetMonitorValueList function expects a list of monitored channel names as the input argument. It returns a list of double precision values from the data structure for the specified channel names. The list of status of the changed event is reset to 0 by CaGetMonitorValueList.

**CaGetMonitorList ( [ 'nm1', 'nm2', ... ] )**

The CaGetMonitorList function expects a list of monitored channel names as the input argument. It returns a list of double precision values: [ value1, status1, severity1, value2, status2, severity2, ... ] from the data structure for the specified channel names. The list of status of the changed event is reset to 0 by CaGetMonitorList.

**CaClearMonitorList ( [ 'nm1', 'nm2', ... ] )**

CaClearMonitorList expects a list of channel names as the input argument. It removes the specified channel names from the monitored list. It returns 0 at normal completion, else returns -1.

2. **Setup Requirement for CaWave**

   - Create a subdirectory, `pv`, for storing PV-WAVE related program files, and make a soft link to `CaWave.pro` and `CaWave.so`

     ```
     mkdir pv
     ```
This is a one-time setup requirement for all procedures. It assumes that a user always runs PV-WAVE in the `pv` subdirectory.

3. **Procedure for Running CaWave**

PV-WAVE is licensed to run on workstation hera. In order to allow hera to make connection to the local host X server following command is required.

```
xhost +hera
```

3.1 **Running Remote on Hera**

- Invoke a PV-WAVE CL session
  ```bash
  pvcstart
  ```
- Change directory to `pv` from a PV-WAVE CL session
  ```bash
  cd, 'pv'
  ```
- Load the `CaWave` functions into PV-WAVE CL session
  ```bash
  .rnew CaWave
  ```
- Enter channel access functions interactively as described above or load other module files which consist of functions written in PV-WAVE CL and `CaWave`

where `pvcstart` is the script file provided by system administrator to run the PV-WAVE CL version on the workstation hera.

3.2 **Login On Hera**

Remote login hera and run CaWave is given below:

- Remote login on hera
- Change directory to `pv` on hera
  ```bash
  cd pv
  ```
- Setup PV-WAVE environment
  ```bash
  setupwave
  ```
- Invoke a PV-WAVE CL session
  ```bash
  wave
  ```
- Load the `CaWave` functions into PV-WAVE CL session
  ```bash
  .rnew CaWave
  ```
- Set environment DISPLAY if plot is to be used
  ```bash
  setenv,’DISPLAY = localhost:0’
  ```
Enter channel access functions interactively as described above or load other module files which consist of functions written in PV–WAVE CL and CaWave

where \texttt{localhostname:0} specifies the workstation display where graphic output to be displayed.

### 4. Example CaWave Session

An example of running \texttt{CaWave} functions interactively under the Open Look window manager is given below. The boldfaced words are user-supplied command syntax, the italicized face words are comments, and the remaining lines are output automatically generated by \texttt{CaWave} and PV–WAVE CL version, respectively. The Unix host name in the following example is ‘pan’.

\begin{verbatim}
| PV–WAVE CL session

pan: \texttt{pvcstart}  \textit{Start PV–WAVE CL Version}

PV–WAVE CL Version 4.01 (sun4 sunos sparc).
Copyright (C) 1989–1992, Precision Visuals Inc.
All rights reserved. Unauthorized reproduction prohibited.

PV–WAVE v4.01 (Fri Aug 21 1992)

Your current interactive graphics device is: X
If you are not running on an sun4 integrated display use the
\texttt{SET_PLOT} command to set the appropriate graphics device
(if you have not already done so).

The following function keys are defined with PV–WAVE commands:

R1 – Start the PV–WAVE Demonstration/Tutorial System
R2 – Invoke the PV–WAVE Online Help Facility
R3 – Output the PV–WAVE Session Status

NOTE: You must be running from a Shell Tool (not a Command Tool)
in order to use the function keys.

\texttt{WAVE> cd,\textasciitilde pv} \textit{Change directory to pv}
\texttt{WAVE> .rnew CaWave} \textit{Load CaWave external functions}
\texttt{% Compiled module: CAWAVE.}
\texttt{% Compiled module: CAEVENT.}
\end{verbatim}
% Compiled module: CA.
% Compiled module: CAERROR.
% Compiled module: CAPENDVENT.
% Compiled module: CAPENDIO.
% Compiled module: CAPENDIOLIST.
% Compiled module: CASLEEP.
% Compiled module: CADEBUG.
% Compiled module: CACLOCK.
% Compiled module: CACLOCK.
% Compiled module: CAGET.
% Compiled module: CAGET.
% Compiled module: CAGETVALUE.
% Compiled module: CAGETVALUE.
% Compiled module: CAGETSTRING.
% Compiled module: CAGETSTRING.
% Compiled module: CAPUTVALUE.
% Compiled module: CAPUTVALUE.
% Compiled module: CAPUT.
% Compiled module: CAPUT.
% Compiled module: CAPUTSTRING.
% Compiled module: CAPUTSTRING.
% Compiled module: CAGETSTATUS.
% Compiled module: CAGETSTATUS.
% Compiled module: CASTATUS.
% Compiled module: CASTATUS.
% Compiled module: CAGETCOUNT.
% Compiled module: CAGETCOUNT.
% Compiled module: CAGETWF.
% Compiled module: CAGETWF.
% Compiled module: CAPUTWF.
% Compiled module: CAPUTWF.
% Compiled module: CASEARCHLIST.
% Compiled module: CASEARCHLIST.
% Compiled module: CAGETERROR.
% Compiled module: CAGETERROR.
% Compiled module: CAERRORLIST.
% Compiled module: CAERRORLIST.
% Compiled module: CAINFOLIST.
% Compiled module: CAINFOLIST.
% Compiled module: CASTATUSLIST.
% Compiled module: CASTATUSLIST.
% Compiled module: CAGETSTATUSLIST.
% Compiled module: CAGETSTATUSLIST.

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.
% Compiled module: CAGETLIST.
% Compiled module: CAGETLIST.
% Compiled module: CAGETVALUELIST.
% Compiled module: CAGETVALUELIST.
% Compiled module: CAPUTLIST.
% Compiled module: CAPUTLIST.
% Compiled module: CAPUTVALUELIST.
% Compiled module: CAPUTVALUELIST.
% Compiled module: CASETLIST.
% Compiled module: CASETLIST.
% Compiled module: CAGETSTRINGLIST.
% Compiled module: CAGETSTRINGLIST.
% Compiled module: CAPUTSTRINGLIST.
% Compiled module: CAPUTSTRINGLIST.
% Compiled module: CAADDMONITOR.
% Compiled module: CAADDMONITOR.
% Compiled module: CACLEARMONITOR.
% Compiled module: CACLEARMONITOR.
% Compiled module: CAWAITEMVENTMONITOR.
% Compiled module: CAWAITEMVENTMONITOR.
% Compiled module: CAEVENTMONITOR.
% Compiled module: CAEVENTMONITOR.
% Compiled module: CAGETMONITORVALUE.
% Compiled module: CAGETMONITORVALUE.
% Compiled module: CAGETMONITORVSS.
% Compiled module: CAGETMONITORVSS.
% Compiled module: CAGETMONITOR.
% Compiled module: CAGETMONITOR.
% Compiled module: CAADDMONITORLIST.
% Compiled module: CAADDMONITORLIST.
% Compiled module: CACLEARMONITORLIST.
% Compiled module: CACLEARMONITORLIST.
% Compiled module: CAWAITEMVENTMONITORLIST.
% Compiled module: CAWAITEMVENTMONITORLIST.
% Compiled module: CAEVENTMONITORLIST.
% Compiled module: CAEVENTMONITORLIST.
% Compiled module: CAGETMONITORVALUELIST.
% Compiled module: CAGETMONITORVALUELIST.
% Compiled module: CAGETMONITORLIST.
% Compiled module: CAGETMONITORLIST.
% Compiled module: CACONNECT.
% Compiled module: CACONNECT.
% Compiled module: CAFLUSHEVENT.

WAVE> cahelp

To get list of Channel Access related functions, enter:
To get list of CA event related functions, enter:

`caevent`

CHANNEL ACCESS EVENT RELATED FUNCTIONS

FUNCTIONS:
- `caAddMonitor`
- `caClearMonitor`
- `caWaitEventMonitor`
- `caGetMonitor`
- `caAddMonitorList`
- `caClearMonitorList`
- `caWaitEventMonitorList`
- `caGetMonitorList`
- `caConnect`

To get help on a function, just enter the function name.

WAVE> `ca`

CHANNEL ACCESS RELATED PROCEDURES AND FUNCTIONS

PROCEDURES:
- `caPendIo`
- `caPendIOList`
- `caPendEvent`
- `caDebug`
- `caSleep`
- `caFlushEvent`
- `caClock`

FUNCTIONS:
- `caGet`
- `caPut`
- `caInfo`
- `caGetCount`
- `caSearchList`
- `caGetList`
- `caPutList`
- `caInfoList`
- `caGetValue`
- `caPutValue`
- `caStatus`
- `caGetStatus`
- `caGetWF`
- `caPutWF`
- `caGetError`
- `caErrorList`
- `caGetValueList`
- `caPutValueList`
- `caStatusList`
- `caGetStatusList`

To get help on a function, just enter the function name.

To get help on a procedure, enter the procedure name followed by the keyword help. e.g.

`capendio/help`

WAVE> `caget`

Help on CaGet function

`caGet('name')`
a channel accesss function returns a double array of dimension 3, which holds the value, status, and severity for the specified record name.

e.g.
\[
x = \text{caGet('chademoai1')}
\]
or
\[
\text{print, caGet('chademoai1')}
\]

\[
\text{WAVE}> \text{print, caGet('chademoai1')}\]
\[
1.0000000 0.0000000 0.0000000
\]

\[
\text{WAVE}> \text{print, caput('chademobil', 1)}\]
\[
0
\]

\[
\text{WAVE}> x = 'chademobil'
\]

\[
\text{WAVE}> \text{print, cagetvalue(x)}\]
\[
1.0000000
\]

\[
\text{WAVE}> \text{print, cagetstring(x) & print} \]
\hspace{1cm} \text{State 1}
\]

\[
\text{WAVE}> \text{u} = ['chademoai1', 'chademomask1', 'chademobil']
\]

\[
\text{WAVE}> v = \text{cagetvaluelist(u) & print, v}
\]
\[
-5.0000000 199.000000 1.0000000
\]

\[
\text{WAVE}> v(1) = 111
\]

\[
\text{WAVE}> \text{print, caputlist(u, v)}
\]
\[
0
\]

\[
\text{WAVE}> \text{print, cagetvaluelist(u)}\]
\[
-9.0000000 111.000000 1.0000000
\]

\[
\text{WAVE}> \text{print, cagetstringlist[u] & print} \]
\hspace{1cm} \wedge
\]

\%
\hspace{1cm} \text{Syntax error on caGetStringList}
\]

\[
\text{print, cagetstringlist[u] & print}
\]
\hspace{1cm} \%
\hspace{1cm} \text{Syntax error}
\]

\[
\text{WAVE}> \text{cagetstringlist} \]
\hspace{1cm} \%
\hspace{1cm} \text{Help on caGetStringList function}
\]

\[
\text{caGetStringList(['}name1', 'name2', ...])} = \]
\hspace{1cm} this function returns a string list of values for the specified list of record names.
\hspace{1cm} It returns a (null) string for the record not found.
\]

\[
e.g.\]
\[
x=['chademoai1', 'chademoai2']
\]

\[
\text{WAVE}> \text{print, caGetStringList(x)}
\]
\hspace{1cm} \%
\hspace{1cm} \text{Get string value for u device list}
\]

\[
\text{WAVE}> \text{print, cagetstringlist(u) & print} \]
\hspace{1cm} 7 111 State 1
\]

\[
\text{WAVE}> \text{x = "chademomask1"}
\]
\hspace{1cm} \%
\hspace{1cm} \text{Set x to a new device name}
\]

\[
\text{WAVE}> \text{print, caeventmonitor(x)}
\]
\hspace{1cm} \%
\hspace{1cm} \text{Check the event for x device}
\]

\[
\text{Error: chademomask1 is not monitored yet}
\]
\hspace{1cm} \%
\hspace{1cm} \text{Device x is not monitored yet}
\]
CaWave User's Guide

0
WAVE> print,caaddmonitor(x)  Add a monitor for x device
 0
WAVE> print,caeventmonitor(x)  Ok
 1
WAVE> print,caeventmonitor(x)  Check for new event on x
 0
WAVE> cadebug,2  New event true
CaWaveDebug: level 2
WAVE> print,caput(x,100)  Check for new event again
Put value print,caeventmonitor(x)  New event false
CaWavePutValue: chademomask1, value=100.000000
 0
WAVE> caflushevent  Set debug print flag to 2
CaWaveFlushEvent:
WAVE> print,caput(x,123)  Old: name=chademomask1, value=111.000000, stat=0, sevr=0, event=0
Put value print,caeventmonitor(x)  New: name=chademomask1, value=100.000000, stat=0, sevr=0, event=1
CaWavePutValue: chademomask1, value=123.000000
 0
WAVE> print,caeventmonitor(x)  Ok
Check for new event on x
WAVE> print,caput(x,123)  Check for new event again
Put value print,caeventmonitor(x)  New event true
CaWavePutValue: chademomask1, value=123.000000
 0
WAVE> print,caeventmonitor(x)  Check for new event again
CaWaveGetOneEvent: chademomask1, Event_ind = 1
 1
WAVE> print,caeventmonitor(x)  New event false
CaWaveGetOneEvent: chademomask1, Event_ind = 0
 0
WAVE> cadebug,0  Set debug print off
WAVE> print,caclearmonitor(x)  Clear event monitor on x
 0
WAVE> print,cainfolist(u)  Print info and return value of u list

<table>
<thead>
<tr>
<th>DEVICE</th>
<th>TYPE</th>
<th>VALUE</th>
<th>STATUS</th>
<th>SEVR</th>
<th>UNITS</th>
<th>UOPR</th>
<th>LOPR</th>
</tr>
</thead>
<tbody>
<tr>
<td>chademomai1</td>
<td>6</td>
<td>19.000000</td>
<td>3</td>
<td>2</td>
<td>Counts</td>
<td>20.000000</td>
<td>-20.000000</td>
</tr>
<tr>
<td>chademomask1</td>
<td>6</td>
<td>123.000000</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0.000000</td>
<td>0.000000</td>
</tr>
<tr>
<td>chademobi1</td>
<td>3</td>
<td>0.000000</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0.000000</td>
<td>0.000000</td>
</tr>
<tr>
<td></td>
<td>19.000000</td>
<td>123.000000</td>
<td>0</td>
<td>0</td>
<td></td>
<td>Returned value for u</td>
<td></td>
</tr>
</tbody>
</table>

WAVE>
WAVE> print,cagetwf('chademomai1')  Get CA waveform record
 0.000000 0.080467000 0.16041100 0.23931600
0.31666800 0.391967000 0.464723000 0.53446600
0.60074200 0.663123000 0.721202000 0.77460500
0.82298400 0.866025000 0.903450000 0.93501600
0.96051800 0.97979100 0.99270900 0.99918900
0.99918900 0.99270900 0.97979100 0.96051800
5. Example of External CaWave Procedures

Procedure for Taking Data and Plotting Results

PRO graph,name,np, delay, help=help
    if keyword_set(help) then goto, usage
    dt = delay
    val = make_array(np+1,/double)
    for i = 0, np do begin
        val(i) = cagetvalue(name)
        print, val(i)
        casleep, dt
    endfor
    plot, val
    return
usage:
    print,"Usage: graph,'dev-name', npoints, time_delay"
    print,""
END

Procedure for Plotting Waveform Record

PRO plotwf,name, help=help
    if keyword_set(help) then goto, usage
    n = cagetcount(name)
    if n lt 1 then goto, err1
    plot, cagetwf(name)
    return
err1:
    print,'Error in record name - ',name
    print,""
    return
usage:
    print,"Usage: plotwf,'wf_dev-name"
    print,""

Procedure for Creating and Plotting a Sine Waveform Record

ENDPRO sine,name, help=help
    if keyword_set(help) then goto, usage
n = cagetcount(name)
if n lt 1 then goto, err1
v = make_array(n,/double)
for i=0,(n-1) do v(i) = sin(2*3.141592654*i/(n-1))
id = caputwf(name,v)
plot,v
return

err1:
print,'Error in record name – ',name
print,'
return

usage:
print,"Usage: sine,'wf_dev_name"
print,""

END

Procedure for Creating and Plotting a Cosine Waveform Record

PRO cosine,name,help=help
  if keyword_set(help) then goto, usage
  n = cagetcount(name)
  if n lt 1 then goto, err1
  v = make_array(n,/double)
  for i=0,(n-1) do v(i) = cos(4*3.141592654*i/(n-1))
id = caputwf(name,v)
plot,v
return

err1:
print,'Error in record name – ',name
print,'
return

usage:
print,"Usage: cosine,'wf_dev_name"
print,""

END
6. **Timing of CaGet / CaPut in PV–WAVE**

The average execution times in seconds for *CaGet*, *CaPut*, *CaGetString*, *CaPutString* are listed below.

<table>
<thead>
<tr>
<th></th>
<th>CaGet</th>
<th>CaGetString</th>
<th>CaPut</th>
<th>CaPutString</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>0.0030</td>
<td>0.0033</td>
<td>0.0019</td>
<td>0.0022</td>
</tr>
</tbody>
</table>
7. CaWave Related Files

Below is a list of the related CaWave files required for accessing the external link functions which can be linked into PV–WAVE. Detailed information on generating the PV–WAVE link external functions can be found in reference 1.

CaWave Files

CaWave.so  The shared channel access object library to be dynamically linked into PV–WAVE.

CaWave.pro  The library of procedures written in PV–WAVE command language to be linked into PV–WAVE.

References