CaWave 3.1 (Release Notes)

Ben-chin Cha and Tim Mooney
Argonne National Laboratory
Advanced Photon Source
Experimental Facilities Division/Beamline Control
Draft (July 19, 1995)

1. CaWave 3.1 Announcement

Overview

A new CaWave release, 3.1, has been created and is ready for user testing.

The primary improvements are in new capabilities for channel-access monitoring. CaWave now provides three types of channel monitors: SIMPLE, QUEUE, and SCAN. The SIMPLE monitor now supports arrays in the same way as the SIMPLE monitor supports scalars. A SIMPLE monitor returns only the most recently cached value(s) of the monitored channel. A QUEUE monitor stores value changes into a buffer until the buffer is either full or has been read. The SCAN monitor stores current values of selected process variables(PVs) into arrays in response to a single trigger process variable.

Because different types of monitors allocate space differently, a consistent set of Add/Get/Clear monitor functions must be used. To change the monitor type for a process variable, one must first clear the old monitor and then add a new monitor of the desired type.

To support these new features, the high-level channel-access library has been modified extensively. A version function has been added so that clients have the information required to support both old and new versions. The version function returns the current version of CaWave; of the high-level, channel-access interface layer; and of the low-level EPICS channel-access library.

In the high-level CA interface library, the default timeout for single process variables has been changed to 5 seconds; the timeout for lists of PVs and for arrays, has been changed to 10 seconds. An initialization routine is provided that resets the timeouts to 1 and 3 seconds for single PVs and lists of PVs, respectively. The initialization routine also turns on the option of automatically calling ca_pend_event after every single ca_put function call.

2. Changes in CaWave 3.1

New Functions

New procedures and functions added to CaWave include the following (descriptions of these new functions are given in the next section):

- **caVersion**
- **caInit**
- **caTimeStamp**
- **caMonitor**
- **caMonitorScan**

Bugs Fixed

Bugs fixed include the following:

- monitor on DBR_STRING type field is possible
- monitor on DBR_ENUM type field can return string or numeral value
- **caget**, which failed to get the string array back, is fixed
- **caerror** function is fixed
- **casearch** on array problems is fixed
- **caput** on null string causing the IDL to crash is fixed.

On-line Help

On-line help on available functions can be accessed by the **caHelp** command. The on-line help is grouped into two categories **ca** and **caEvent**. Instructions about each commands are given by the **caHelp** routine.

Setup for IDL

The script files *setupidl* and *idl_startup.pro* are provided in the /usr/local/epics/extensions/bin/sun4 directory. The *setupidl* will properly set up two environment variables CAIDL_SHARE and IDL_STARTUP for the user. The setup procedure at the APS for running CaWave is given below. (Note, different site or system may need to modify the *setupidl* to suit its own system environments.)

1. In order to properly access all the EPICS extensions tools, a user should make sure that the /usr/local/epics/extensions/bin/sun4 is in his search path.
2. **source /usr/local/epics/extensions/bin/sun4/setupidl**

   The following environment variables are set by *setupidl*

   CAIDL_SHARE=/usr/local/epics/extensions/bin/sun4/CaIDL.so
   IDL_STARTUP=/usr/local/epics/extensions/bin/sun4/idl_startup.pro
   IDL_DIR=/usr/local/idl
   IDL_PATH=+/usr/local/idl
   IDL_DEVICE=X
3. A user can start the IDL by entering the Unix command:

   idl

   The *CaWave.pro* and *CaWidgets.pro* are automatically loaded into the IDL, the procedures **caVersion** and the **caInit** are automatically called by the *idl_startup.pro.* At the end of start-up, a user will get the IDL prompt.

The IDL start-up file *idl_startup.pro* allows the user to use the EPICS_EXENSIONS and EPICS_EXTENSIONS_PVT environments to include more directories in the user search path for accessing private versions of IDL programs.

In case a user wants to override the default setting, (s)he can provide a private version of *idl_startup.pro* to override the default setting.

3. New Functions/Procedures in CaWave 3.1

The names of procedures and functions defined in CaWave can be entered in any combination of upper and lower case in IDL. However, all letters making up process variable names must have the correct case.

CaVersion

This procedure returns the version of CaWave currently loaded into the IDL. A typical string returned by **caVersion** follows:

CaWave Version 3.1(Lca2.1) (EPICS Version 3.12.0Beta13)

This version string indicates that CaWave 3.1 is loaded into the IDL; that it uses version 2.1 of the high-level channel-access library, Lca; and that it is linked with the low-level library of EPICS version 3.12.0Beta13.

Example

```
caVersion
```

CaInit

This procedure resets the timeouts used by the CaWave functions. It sets the timeouts used by ca_pend_io to 1.0 second for a single process variable and to 3.0 seconds for a list of PVs or a non scalar type of PV. It resets the timeout used by ca_pend_event to 0.001 seconds. It also turns on the option of automatically calling ca_pend_event at the end of each **caput** to a single PV.

Example

```
caInit
```

caTimeStamp

CaTimeStamp('name')

This function returns the time stamp for the monitored process variable 'name'. The time stamp reflects the last value change occurring for the monitored name. The returned string consists of date and time information.

Example

print,caTimeStamp('chademoai1')

CaMonitor

caMonitor('name', op_keyword, type_keyword, maxqueue=n)

This function can Add/Get/Clear monitoring of the specified process variable 'name'.

This function supports the SIMPLE, and QUEUE monitor types, defined in CaWave, with a single function. It uses two keywords, operation and type, to determine which monitor function will be called, and it returns the values accordingly.

It provides two types of process variable monitors: a simple, current-value-change monitor and a queue-value-change monitor (QUEUE type). For a simple, value-change monitor, the name can accept a list of process variable names; for a QUEUE-type monitor, the name can only accept a single process variable name.

The simple monitor can also be used to monitor the array or non scalar type record. In getting the monitored non scalar type of values back, the name can only accept a single process variable name. If an array of PV names is requested in getting the values back, only the first value of the monitored PV names is returned.

For a given PV name, different space is allocated for each different type of monitor. A consistent type of Add/Get/Clear monitor function should be used. To change a monitor type for a PV, a user first has to clear the old type monitor then add the new monitor type.

A change-of-connection event is automatically added for each of the monitor channels.

Op_keyword:

/ADD

Add CA monitor for the specified PV 'name'
Return 0 for success, -1 for failure

/CLEAR

Clear CA monitor and free space for the specified 'name'.
Return 0 for success, -1 for failure.

/GET

Get monitor values back for the specified 'name'.
Different values are returned for different types of monitors.

If the PV name is a scalar type of record, the current value in double precision is returned.

If the PV name is a vector type of record, a native type array of the current values is returned. The default returned array size is the same as the native array size. The maxqueue can be used to specify the number of values to be returned for a vector type of record.

If a QUEUE type is specified, a real array is returned. The first number is the buffer full indicator, the second number is the number of values, and the rest are monitor values.

The overflow indicator can be 0 or 1; 1 indicates that the buffer is full, any new value change will be lost. The number of values returned cannot exceed the maxqueue; it can be -1 if nothing is returned or if the function fails.

The queue buffer is cleared after each /Get call and is ready to queue for a new value. If no more data is returned by the monitor queue, a list of two integer numbers [0 -1] is returned.

Type_keyword:

/QUEUE

Queues the value changes for the monitored channel until the user reads them. For queueing the monitor, the same maxqueue must be used in the /Add and /Get options.

MAXQUEUE=n

The number n specifies the size of an array for a QUEUE-type monitor. Data may be lost if the size specified is too small, and the data come too fast, i.e., before a user is able to cache them.

Examples

An example of using the simple monitor is given below. In this example, the process variable name used is ‘chademoai1.VAL’.

```
print,caMonitor(‘chademoai1.VAL’,/add)
print,caMonitor(‘chademoai1.VAL’,/get)
print,caMonitor(‘chademoai1.VAL’,/clear)
```

An example of using the queue monitor is given below. In this example, the process variable name used is ‘chademoai1.VAL’, and the buffer used for queuing is 100. The buffer is automatically cleaned at the end of each get. The maximum number of values that can be queued is 100.

```
print, caMonitor(‘chademoai1.VAL’,/add,/queue,maxqueue=100)
print, caMonitor(‘chademoai1.VAL’,/get,/queue,maxqueue=100)
print, caMonitor(‘chademoai1.VAL’,/clear,/queue)
```

CaMonitorScan

CaMonitorScan(‘name’, pvnames, op-keyword, max=n)

This function can Add/Get/Clear/Zero the monitor of the specified process variable ‘name’ that triggers the scanning for the input list of pvnames.

It automatically places monitors on each of the process variables in the list ‘pvnames’. It uses the op-keyword to determine what type of operation will be performed on the monitor. The max=n keyword specifies the array size to be used in allocating space for each process variable. This function returns the values according to the op-keyword specified.

A change-of-connection event is automatically added for each of the monitor channels

Op_keyword:

/ADD

Add CA monitors for the specified ‘name’ and all PVs in pvnames.
Return 0 for success, -1 for failure.

/CLEAR

Clear the whole set of CA monitors and free space for the specified ‘name’
Return 0 for success, -1 for failure.

/GET

Get the monitor real arrays back for the specified set of pvnames.
Return -1 for failure, 0 for no data. If scanning has started, the scan data so far caught by CaWave are returned as a real array. The first number is the current number of values detected and is followed by the exact number of sequential sets of real values corresponding to ‘pvnames’.

/ZERO

At the end of the scan, the old data caught by CaWave are kept in memory. Before doing a new scan, zero out the old scan data.

MAX=n

The number n specifies the number of data points that can be acquired.

Example

An example of using scan monitor is given below. In this example the scan trigger control process variable name is ‘cha:scan1.VAL’, the pvnames to be acquired include ‘cha:scan1.R1CV’, ‘cha:scan1.R2CV’, ‘cha:scan1.R3CV’, ‘cha:scan1.R4CV’, ‘cha:scan1.D1CV’, ’cha:scan1.D2CV’, ’cha:scan1.D3CV’, and ‘cha:scan1.D4CV’. The maximum number of data to be acquired is 100.

```
pvnames=[ ’cha:scan1.R1CV’, ’cha:scan1.R2CV’,’cha:scan1.R3CV’, $
        ’cha:scan1.R4CV’, ’cha:scan1.D1CV’,’ cha:scan1.D2CV’,  $
        ’cha:scan1.D3CV’, ’cha:scan1.D4CV’ ]

print,caMonitorScan(‘cha:scan1.VAL’,pvnames,/add,max=100)
print,caMonitorScan(‘cha:scan1.VAL’,pvnames,/zero,max=100)
print,caMonitorScan(‘cha:scan1.VAL’,pvnames,/get,max=100)
print,caMonitorScan(‘cha:scan1.VAL’,pvnames,/clear,max=100)
```