CaMath User's Guide

(by Ben-chin Cha and Bob Daly)

1 . Introduction

CaMath is an external Mathematica package which can be loaded into Mathematica by a user. **CaMath** consists of a special set of channel access functions which provides the Mathematica users with easy and flexible access of channel information across the IOC networks. It also provides a completes set of process variable event monitoring functions. The available functions for **CaMath**, their functionality, and their syntax are described in following sections.

This document also gives examples how a Mathematica user can interface to channel access devices. It is assumed that the user is already familiar with using Mathematica[1]. Few examples of Mathematica module of using **CaMath** functions are also given in this document.

1.1 New Features

More flexible CaMath function prototypes are adopted starting from this revision. Now CaMath uses the same prototype function name for either single or multiple channel names. Prior this version the suffix `List' name is used in prototype functions for a list of channel names, The old `List' type functions are still available, this ensures that the application written in early version works in this new release too.

DevMath, an external Mathematica link program for Device Access Library, is also available for users. **DevMath** can interface smoothly with **CaMath**. This will be an EPICS added on application tool not an official tool. If any one want to use this package please contact me. The channel names returned by the **devSend** can be used directly by the **CaMath** functions. The available functions for **devMath**, their functionality, and their syntax are also described in this document. The user interface for **devMath** coupled with **CaMath** is also shown in this guide.

1.2 Speedup Performance

When **CaMath** is loaded into Mathematica (**math**), Mathematica will continuously call the ca_pend_event(timeout) function to flush the channel access pending events whenever the Mathematica is idling. This wait time is changeable by the user in a **CaMath** session. The default timeout setting in **CaMath** is 0.1 second. Most of channel access get and put functions in **CaMath** depends on this default wait time setting. If the performance is not acceptable, a user can set this timeout to a smaller value like 0.001 second. This will improve the CaMath performance greatly.

2 . External CaMath Link Functions

The available **CaMath** link functions can be grouped into three categories: functions operating on a single channel or 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 functions defined in **CaMath** conform to the following naming convention: all function names begins with the prefix "**Ca**". Normally 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 channel name referred to in this document is the same as the process variable name defined in the DCT 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 3 seconds for a list of channel names; and the default pend event timeout used is 0.1 second. These timeout values can be reset by the user.

The functionality of each **CaMath** function is described below.

2.1 General / Debug / Error Handling Functions

The general functions include **CaHelp**, **CaDebug**, **CaSleep**, **CaPendIO**, **CaPendIOLIST**, **CaPendEvent**, **CaError**, **CaSearch**, and **CaConnect**.

2.1.1 CaHelp

CaHelp[ ]

This function makes a complete list of the currently available external functions defined for **CaMath**. It returns nothing.

2.1.2 CaSleep

CaSleep[time]

This function 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.

2.1.3 CaPendIO

CaPendIO[time]

This function allows the user to reset the TIMEOUT seconds for ca_pend_io function[2] when getting single channel information. The default timeout is 0.3 second; a TIMEOUT of zero is forever. If time < 0.01 is specified, a value of 0.01 is used. It returns nothing.

2.1.4 CaPendIOLIST

CaPendIOLIST[time]

This function allows the user to reset the TIMEOUT seconds for processing a list of channels. The default timeout is 3 seconds; a TIMEOUT of zero is forever. If time < 0.01 is specified, a value of 0.01 is used. It returns nothing.

2.1.5 CaPendEvent

CaPendEvent[time]

This function allows the user to reset the TIMEOUT seconds for ca_pend_event function. The default TIMEOUT is 0.1 second; a TIMEOUT of zero is forever. If time = 0 is specified, a value of 0.001 is used. It returns nothing.

NOTE: When **CaMath** is loaded into Mathematica, Mathematica repeatedly calls ca_pend_event(timeout) whenever Mathematica is idling. This will continuously wait for timeout and flush the pending CA events. If the response of **CaMath** is unacceptably slow, a user can use this function with a smaller TIMEOUT to improve the performance.

2.1.6 CaPendEventOn

CaPendEventOn[i]

The ca_pend_event function is automatically called in putting values for a list of channels but not for a single channel. This function allows the user to turn on / off the automatically calling ca_pend_event for single channel. Default to 0 is off. Set i to 1 turns on. It returns nothing.

NOTE: Calling this function is not required for **CaMath** user, because currently **CaMath** automatically calls ca_pend_event whenever Mathematica is idle.

2.1.7 CaDebug

CaDebug[i]

This function sets the runtime printing flag. The default value is initially set to 0 when **CaMath** 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. If i is set to 3, the various TIMEOUT values used by **CaMath** is printed. The printing flag can be reset to 0 when no more debug information is desired. It returns nothing.

2.1.8 CaError

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.

CaError[{nm1, nm2, . . . }]

This **CaError** 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 error code can be zero or -1. An error code of 0 indicates that the last channel access call completed successfully. An error code of -1 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). This function can also accept a single channel name as input.

2.1.9 CaSearch

CaSearch[{nm1, nm2, . . . }]

The `CaSearch` 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 CaError 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. This function can also accept a single channel name as input.

2.1.10 CaConnect

CaConnect[nm]

The **CaConnect** function expects a channel name as the input argument. It adds a re-connection event for a specified channel. When re-connection happens, a user must wait for reestablishment of channel access in order to guarantee the normal operation of channel access functions.

2.2 Channel Access Functions

The channel access functions include **CaGet**, **CaPut**, **CaGetString**, **CaPutString**, **CaGetValue**, **CaGetStatus**, **CaInfo**, **CaStatus**, **CaGetCount**, **CaGetType**, **CaGetWF**, and

CaPutWF. They can be used interactively to get values back or set values for a single / multiple channels.

Basically, each channel access function where expects process variable name either a single channel name or a list of channel names can be entered. Normally functions operating on a single channel return a single value and functions operating on a list of channel names return a list of values. For clarity both forms of MathLink functions are provided below.

2.2.1 CaGetValue

CaGetValue[nm]

To query the numerical value of a channel use the **CaGetValue** function. The **CaGetValue** function expects a string containing an IOC channel name defined in the database and returns to Mathematica the current value of the channel.

CaGetValue[{nm1, nm2, . . . }]

This **CaGetValue** function expects a list of channel names as input. It returns a list of the values corresponding to the requested channels. This function is equivalent to the old **CaGetValueList**.

2.2.2 CaGet

CaGet[nm]

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 to Mathematica. The list consists of value, status, and severity of the channel.

CaGet[{nm1, nm2, . . . }]

This `CaGet` function expects a list of channel names as input. It returns a list of lists which contain {{value1, status1, severity1}, {value2, status2, severity2}, . . . } corresponding to the requested channels. This function is equivalent to the old **CaGetList**.

2.2.3 CaPut

CaPut[nm, rl]

To set the numerical value of a channel use the **CaPut** function. This function expects two input arguments, channel name and new value, and returns the new value as real. The set value can be real or integer. A synonymous function name **CaSet** can be used for **CaPut**.

CaPut[{nm1, nm2, . . . }, {vl1, vl2, . . . }]

This **CaPut** expects two lists as input: a list of channel names, and a list of new values to be assigned to the channels. **CaPut** returns the list of the set values corresponding to the requested channels. The input values can be real or integer. This function is equivalent to the old **CaPutList**.

2.2.4 CaGetString

CaGetString[nm]

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.

CaGetString[{nm1, nm2, . . . }]

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

2.2.5 CaPutString

CaPutString[nm, "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 the new string value.

CaPutString[{nm1, nm2, . . . }, {vl1, vl2, . . . }]

For some channels the value field is defined as an ASCII string rather than a numerical value. This function CaPutString is used to assign a string value to the record field for these channels. **CaPutString** expects two input arguments: channel name list and new string value list. It returns the list of strings of the new values written to the IOC. This function is equivalent to the old **CaPutStringList**.

2.2.6 CaInfo

CaInfo[nm]

The `CaInfo` function expects a channel name as an input argument. It returns a string of text for the querying channel. In addition, the channel name, value, status, severity, operating ranges, and units used are displayed on the user's terminal.

CaInfo[{nm1, nm2, . . . }]

This **CaInfo** function expects a list of channel names as the input argument and returns a list of 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. This function is equivalent to the old **CaInfoList**.

2.2.7 CaStatus

CaStatus[nm]

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. It returns -1 if the channel not found.

CaStatus[{nm1, nm2, . . . }]

This **CaStatus** expects a list of channel names as input and returns a list of the current status codes corresponding to the requested channel names from the internal data structure populated from the previous channel access call. The status code with value of -1 indicates channel not found. This function is equivalent to the old **CaStatusList**.

2.2.8 CaGetStatus

CaGetStatus[nm]

The **CaGetStatus** function expects a channel name as an input argument. It returns the most current status of the specified channel from IOC. It returns -1 if the channel not found.

CaGetStatus[{nm1, nm2, . . . }]

This **CaGetStatus** expects a list of channel names as input and returns a list of the most current status codes corresponding to the requested channel names from IOC. The status code with value of -1 indicates channel not found. This function is equivalent to the old **CaGetStatusList**.

2.2.9 CaGetType

CaGetType[nm]

The **CaGetType** function expects a channel name as an input argument. It returns the native field type for the specified channel from the IOC network. It returns -1 if the channel not found.

CaGetType[{nm1, nm2, . . . }]

This **CaGetType** expects a list of channel names as input and returns a list of the native database field types for the specified channels. A value of -1 is returned for the not found channel.

2.2.10 CaGetCount

CaGetCount[nm]

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. It returns 0 if the channel not found.

CaGetCount[{nm1, nm2, . . . }]

This **CaGetCount** expects a list of channel names as input and returns a list of the native database element counts for the specified channels. A value of 0 is returned for the not found channel.

2.2.11 CaGetWF

CaGetWF[nm]

The **CaGetWF** function expects a channel name as an input argument. It returns a list of native type values of the waveform record. The size of the returned list is the same as the count of the native waveform record.

2.2.12 CaPutWF

CaPutWF[nm, {vl1, vl2, . . .}]

The **CaPutWF** function expects input consisting of channel name and a list of values which should not exceed the count of the waveform record. It writes the input list to the specified waveform record to the IOC and returns the native count of the waveform record. The input value list can be a real list or a database native type list.

2.3 Channel Access Monitoring Functions

The value change event monitoring functions include **CaAddMonitor**, **CaWaitEvent-Monitor**, **CaEventMonitor**, **CaGetMonitor**, and **CaClearMonitor** for operating on either a single channel or a list of multiple channel names. They gives the user control of adding / removing / getting the value change status of the channels to be monitored.

2.3.1 CaAddMonitor

CaAddMonitor[nm]

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. A new event is generated immediately after adding the event, after each writing of a value to the channel, and after each new connection with the channel's current value.

CaAddMonitor[{nm1, nm2, . . . }]

This **CaAddMonitor** 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. This function is equivalent to the old **CaAddMonitorList**. For each newly-added monitored channel, a new event is generated immediately after adding the event, after each writing of a value to the channel, and after each new connection with the channel's current value.

2.3.2 CaWaitEventMonitor

CaWaitEventMonitor[wtime, nm]

CaWaitEventMonitor function expects two input arguments: the specified time in seconds for waiting 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 value change event happened within the specified time interval since the last call of **CaGetMonitor**.

CaWaitEventMonitor[wtime, {nm1, nm2, . . . }]

This **CaWaitEventMonitor** expects two input arguments: the specified time in seconds for waiting 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 happened within the specified time interval and -1 if no value change event happened within the specified time interval. This function is equivalent to the old **CaWaitEventMonitorList**.

2.3.3 CaEventMonitor

CaEventMonitor[nm]

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 **CaEventMonitor**.

CaEventMonitor[{nm1, nm2, . . . }]

This **CaEventMonitor** 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 by **CaEventMonitor**. The returned element is 0 if either the value change has already been processed by **CaEvnetMonitor** or the value has not changed. This function is equivalent to the old **CaEventMonitorList**.

2.3.4 CaGetMonitorValue

CaGetMonitorValue[nm]

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 set to 0 if **CaEventMonitor** was not called before calling **CaGetMonitorValue**.

CaGetMonitorValue[{nm1, nm2, . . . }]

This **CaGetMonitorValue** function expects a list of monitored channel names as the input argument. It returns a list of current values from the data structure for the specified channel names. The status of the changed event is set to 0 if CaEventMonitor was not called before calling **CaGetMonitorValue**. This function is equivalent to the old **CaGetMonitorValueList**.

2.3.5 CaGetMonitor

CaGetMonitor[nm]

CaGetMonitor function expects a channel name as the input argument. It gets the current {value, status, severity} from the data structure for the specified monitored channel name. The status of the changed event is set to 0 if **CaEventMonitor** was not called before calling **CaGetMonitor**.

CaGetMonitor[{nm1, nm2, . . . }]

This `CaGetMonitor` function expects a list of monitored channel names as the input argument. It returns a list of current {{value1, status1, severity1}, {value2, status2, severity2}, ... } from the data structure for the specified channel names. The status of the changed event is set to 0 if `CaEventMonitor` was not called before calling `CaGetMonitor`. This function is equivalent to the old `CaGetMonitorList`.

2.3.6 CaClearMonitor

CaClearMonitor[nm]

CaClearMonitor function expects a channel name as the input argument. It removes the specified channel name from the monitored list.

CaClearMonitor[{nm1, nm2, . . . }]

This **CaClearMonitor** expects a list of channel names as the input argument. It removes the specified channel names from the monitored list. This function is equivalent to the old **CaClearMonitorList**.

3 Setup Requirement for CaMath and devMath

This is a one-time setup requirement for all procedures. It assumes that a user always runs **CaMath** in the **math** subdirectory.

Starting form EPICS 3.11.6 the **CaMath** becomes an add_on tool. Both **CaMath** and the script **setupmath** reside in add_on bin direcotry. In the following setup example, it is assumed that the add_on related files reside in the **/net/phebos/epics/add_on** directory. In order to run **setupmath** an environment variable ADD_ON must be defined first.

- Create a subdirectory, **math**, for storing Mathematica-related program files, and setup environment for **CaMath**
- **mkdir math**
- **cd math**
- **setenv ADD_ON /net/phebos/epics/add_on**
- **$ADD_ON/bin/setupmath**

4 Procedure for Running CaMath

- Invoke a **math** session
- Load the file **CaMath.m** into **math**
- Enter the channel access functions as described above or load other macro files which consist of functions written in **math** and **CaMath**

4.1 Example CaMath Session

An example of running **CaMath** 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 **CaMath** and Mathematica (**math**), respectively. The Unix host name in the following example is `pan'.

CaMath Command Input/Output	**Comment**
pan 292: **math**	*Start math session*
Mathematica 2.2 for SPARC	
Copyright 1988-93 Wolfram Research, Inc.	
-- OPEN LOOk graphics initialized --	
In[1]:= **<<CaMath.m**	*Load CaMath into math*
Uninstall::unlink: External package CaMath has not been installed.	
Uninstall::unlink: External package devMath has not been installed.	
CaMath Version 2.1.0 Installed.	
devMath Version 1.0.0 Installed.	
ca_task_initialize	
In[2]:= **CaHelp[]**	*List available CaMath functions*

Available CaMath functions include following:

CaGet[{names}], CaPut[{names},{val}]

CaGetString[{names}], CaPutString[{names},{val_string}]

CaGetValue[{names}], CaGetStatus[{names}]

CaGetCount[{names}], CaGetType[{names}]

CaInfo[{names}], CaStatus[{names}]

CaGetWF["name"], CaPutWF["name",{vals}]

CaAddMonitor[{names}]

CaWaitEventMonitor[wtime,{names}]

CaEventMonitor[{names}]

CaGetMonitorValue[{names}]

CaGetMonitor[{names}]

CaClearMonitor[{names}]

CaError[], CaError[{names}]

CaSearch[{names}], CaConnect["name"]

CaDebug[]

CaSleep[time]

CaPendIO[time]

CaPendIOLIST[time]

CaPendEvent[time]

CaPendEventOn[i]

CaHelp[]

CaMath Command Input/Output	**Comment**
In[3]:= **`CaGet["chademobi1"]`**	*Get value, status, severity of ``chademobi1"*
Out[3]= {0, 0, 0}	
In[4]:= **`CaPut["chademobi1"]`**	
Out[4]= CaPut[chademobi1]	*Put value to ``chademobi1" w/o given value*
In[5]:= **`CaPut["chademobi1",1]`**	*Math evaluation of last command*
Out[5]=1	*Put value of 1 to device ``chademobi1"*
In[6]:= **`x = "chademoai1"`**	
Out[6]= chademoai1	*Assign x to ``chademoai1"*
In[7]:= **`CaGetValue[x]`**	
Out[7]= -7.	*Get value for device x*
In[8]:= **`u = {"chademoai1","chademomask1","chademobi1"}`**	
Out[8]= {chademoai1, chademomask1, chademobi1}	*Set u to a list of device string*
In[9]:= **`v = CaGetValueList[u]`**	
Out[9]= {17., 10., 1.}	*Set v to the return list from caGetValueList*
In[10]:= **`v`**	
Out[10]= {17., 10., 1.}	
In[11]:= **`v[[2]]=100`**	*List contents of list v*
Out[11]= 100	
In[12]:= **`v`**	*Reset the 2nd element of v to 101*
Out[12]= {17., 100., 1.}	
In[13]:= **`CaPutList[u,v]`**	*List contents of list v again*
Out[13]= {17., 100., 1.}	
	Put new values v to devices specified by u

CaMath Command Input/Output	**Comment**
In[14]:= **CaGetValue[u[[2]]]**	*Get the value of the 2nd element of u*
Out[14]= 101.	
In[15]:= **CaGetStringList[u]**	*Get the string values of the elements*
Out[15]= {"-11","100","State 1"}	*of u*
In[16]:= **u = {"chademobi1","chademomask1"}**	
Out[16]= {chademobi1, chademomask1}	*Reset the elements of the list u*
In[17]:= **CaAddMonitorList[u]**	
In[18]:= **CaDebug[2]**	*Add channels of u to the monitor list*
In[19]:= **CaGetList[u]**	*Turn on run-time debug option*
Old: nmae=chademobi1, value=1.000000, stat=7, sevr=1,error=2	*Get the value of List u*
New: nmae=chademobi1, value=1.000000, stat=7, sevr=1, error=0	*Debug output*
Old: nmae=chademomask1, value=100.000000, stat=0, sevr=0, error=-2	
New: nmae=chademomask1, value=100.000000, stat=0, sevr=0, error=0	
caMathGetVSSArray:	
chademobi1 value=1.000000, stat= 7, sevr=1	
chademomask1 value=100.000000, stat= 0, sevr=0	
Out[19]= {{1., 7, 1}, {100., 0, 0}}	*Return values from CaGetList[u]*
In[20]:= **CaEventMonitorList[u]**	*Check event status of the monitoring list u*
caGetEventList: name=chademobi1, value=1.000000, status=7, event=1	
caGetEventList: name=chademomask1, value=100.000000, status=0, event=1	*Debug output*
Out[20]= {1, 1}	
n[21]:= **CaEventMonitorList[u]**	
Out[21]= {0, 0}	
In[22]:= **CaPutList[u,{0,22}]**	*Recheck the event status of the u again*
Old: nmae=chademobi1, value=1.000000, stat=7, sevr=1, event=1	*No value changes since last call*
New: nmae=chademobi1, value=0.000000, stat=0, sevr=0, event=1	*Put new values to the monitored channels*
Old: nmae=chademomask1, value=100.000000, stat=0, sevr=0, event=1	
New: nmae=chademomask1, value=22.000000, stat=0, sevr=0, event=1	
Old: nmae=chademobi1, value=0.000000, stat=0, sevr=0, error=-2	
	Debug output
New: nmae=chademobi1, value=0.000000, stat=0, sevr=0, error=0	
Old: nmae=chademomask1, value=22.000000, stat=0, sevr=0, error=-2	
New: nmae=chademomask1, value=22.000000, stat=0, sevr=0, error=0	
Out[22]= {0, 22.}	*Return list of CaPutList*

CaMath Command Input/Output	Comment
In[23]= **CaClearMonitorList[u]**	*Remove list u from the monitoring list*
In[24]:= **CaDebug[0]**	*Turn off run-time debug*
In[25]:= **CaInfoList[u]**	*Get info about all devices in u*

DEVICE TYPE VALUE STATUS SEVR UNITS UOPR LOPR

chademobi1 3 0.000000 0 0 0.000000 0.000000

chademomask1 6 22.000000 0 0 0.000000 0.000000

Out[25]= {0, 22. }

In[26]:= **??CaGetWF**

CaGetWF[name] - returns the native values of the waveform array

of the specified record-name string name.

CaGetWF[x_String] := ExternalCall[LinkObject["CaMath", 1, 1],

CallPacket[34, {x}]]

In[26]:= **CaGetWF["chademowf1"]**	*Get waveform record*

Out[26]= {0.308825, 0.308825, 0.308825, 0.808898, 1., 0.809137, 0.309211, >
-0.308822, -0.808896, -1., -0.809138, -0.309214, 0.30882, 0.808895, 1., >
0.80914, 0.309216, -0.308817, -0.808893, -1., -0.809141, -0.309219, >
0.308815, 0.308815, 0.308815, 0.808892, 1., 0.809143, 0.309221, >
-0.308812, -0.80889, -1., -0.809144, -0.309224, 0.30881, 0.808889, 1., >
0.809146, 0.309226, -0.308807}

In[27]:= **Quit**	*Quit math session*
ca_task_exit	
Uninstall CaMath functions	
pan 293 :	*Return to Unix prompt*

In the above example, Out[4] echoes Mathematica's evaluation of In[4], because an error in the number of arguments is found in CaPut. CaPut expects two arguments: device name and set value. Legitimate device names and values must be entered for normal operation of the channel access functions. To ensure the right data is obtained, a user should check the status returned by channel access functions by calling CaError[].

5 External devMath Link Functions

The available device access library functions include **devSend**, **devCount**, and **devMatch**. 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 and returns the names of found channels or device instances. It provides a simple handle for user to map devices to channels. It allows mathematica users freely to access device library. Only their usages in Mathematica are included here. Please refer reference [4] for detail description and definition of device library. These functions will work only if the device server **idev_srv** is running on the host.

5.1 Device Access Library Functions

The available device access library functions include **devSend**, **devCount**, and **devMatch**. The names of the device access library interface functions are kept same as in the device access library. The syntax of device access functions used in Mathematica is given below.

5.1.1 devCount["dev_name"]

For a user specified device name, this function returns a list which consists of the number of count and the device names.

5.1.2 devSend["dev_name"]

This function lists the matched atomic device instance definitions, and returns the list of instance names in the device. The function **devInfo["dev_name"]** is defined same as **devSend["dev_name"]** in **devMath**.

5.1.3 devSend["dev_name","msg",{v_list}]

Sends indicated message (msg) to named device. The v_list argument is required only when the message implies a write operation. The v_list is an list which can be numerical or string type.

5.1.4 devMatch["str*"]

For a user specified wild card string, this function returns a list of the device names which matches the wild card search.

5.2 Example devMath Session

devMath Command Input / Output	Comment

pan 300: **math**

Mathematica 2.2 for SPARC

Copyright 1988-93 Wolfram Research, Inc.

-- OPEN LOOk graphics initialized --

In[1]:= **Install["devMath"]**

devMath Version 1.1.0 Installed.

Out[6]= LinkObject[CaMath, 2, 2]

Start math session

Install devMath

In[2]:= **devSend["chademo"]**

```
BO1      set     chademobo1
BO1      read    chademobo1
BO1      status  chademobo1
BO1      on      chademobo1
BO1      off     chademobo1
AI2      set     chademoai2
AI2      read    chademoai2
AI2      status  chademoai2
AI2      on      chademoai2
AI2      off     chademoai2
MASK1    set     chademomask1
MASK1    read    chademomask1
MASK1    status  chademomask1
MASK1    on      chademomask1
MASK1    off     chademomask1
```

Out[2]= {BO1, AI2, MASK1}

Query a composite device named "chademo"

It shows the atomic device definition of this composite device and returns the 3 instances BO1, AI2, MASK1 to Mathematica

In[3]:= **devSend["chademo","read"]**

```
BO1      read    chademobo1
AI2      read    chademoai2
MASK1    read    chademomask1
BO1      0
AI2      0
MASK1    0
```

Out[3]= {chademobo1, chademoai2, chademomask1}

Send "read" message to composite device "chademo"

It display read value and returns the correspnoding 3 process variable names to Mathematica

In[4]:= **devSend["chademo","set",{1,22,333}]**

Send "set" message with a list of new value to the composite device "chademo"

```
BO1      set     chademobo1
AI2      set     chademoai2
MASK1    set     chademomask1
```

Out[4]= {chademobo1, chademoai2, chademomask1}

It sets values and returns the PVs to Mathematica

In[5]:= **devSend["chademo","read"]**

Send "read" message to composite device "chademo"

```
BO1      read    chademobo1
AI2      read    chademoai2
MASK1    read    chademomask1
BO1      1
AI2      22
MASK1    333
```

Out[5]= {chademobo1, chademoai2, chademomask1}

It display read value and returns the PVs to Mathematica

In[6]:= **Install["CaMath"]**

Install "CaMath" into Mathematica

CaMath Version 2.1.0 Installed.

Out[6]= LinkObject[CaMath, 2, 2]

In[7]:= **y = devInfo["chademo"]**

Set y = devInfo["chademo"]

```
BO1      set     chademobo1
BO1      read    chademobo1
BO1      status  chademobo1
BO1      on      chademobo1
BO1      off     chademobo1
AI2      set     chademoai2
AI2      read    chademoai2
AI2      status  chademoai2
AI2      on      chademoai2
AI2      off     chademoai2
MASK1    set     chademomask1
MASK1    read    chademomask1
MASK1    status  chademomask1
MASK1    on      chademomask1
MASK1    off     chademomask1
```

Out[7]= {BO1, AI2, MASK1}

In[8]:= **y = devInfo["chademo","read"]**

Send "read" msg to composite device "chademo" and return PVs as y

```
BO1          read    chademobo1
AI2          read    chademoai2
MASK1        read    chademomask1
BO1     1
AI2     22
MASK1   333
```

Out[8]= {chademobo1, chademoai2, chademomask1}

I
n[9]:= **CaPut[y,{1,2,3}]**

Use CaPut set the new values to y list

Out[9]= {1., 2., 3.}

In[10]:= **CaGetValue[y]**

Use CaGetValue to read y list

Out[10]= {1., 2., 3.}

In[11]:= **Quit**

6 Example of External Math Functions

6.1 Function for Taking Data and Plotting Results

```
Graph::usage = "Graph[\"devname\",points,seconds] - Plots data versus real time\n
    devname - specifies the requested device name\n
    points  - specifies the data points to be taken\n
    seconds - specifies the time interval in seconds"

Graph[in_String,np___Integer,dt___Real|dt___Integer] := (
    t0=SessionTime[];
    npoints = If[np>0, np, 20];
    interval = If[dt>0, dt, 1];
    mplot=Table[0,{npoints}];
    For[i=1,i<=npoints,i++,
        (
        t=SessionTime[]-t0;
```

```
        x=N[CaGetValue[in]];
        mplot[[i]]={t,x};
        Print[{t,x}];
        Pause[interval]
         )
      ];

ListPlot[mplot,PlotJoined->True,Frame->True,
     FrameLabel->{"Time (Sec)", "Value",in," "}]
 )
```

6.2 Function for Plotting Waveform Record

```
PlotWF::usage = "PlotWF[\"WFrecordname\"] \n
        WFrecordname - specifies the requested waveform \n
                device name"
PlotWF[in_String] := (
     x= CaGetWF[in];
     Print[x];
     ListPlot[x,PlotJoined->True]
 )
```

6.3 Function for Creating and Plotting Waveform Record

```
sine::usage = "sine[\"devname\"] - set and plot the sine wave form\n"

sine[in_String]:=(
   npoints=CaGetCount[in];
   x=Table[0,{npoints}];
   For[i=1,i<=npoints,i++,
            (  ii=Pi*2/npoints*i;
```

```
        x[[i]]=N[Sin[ii]];
      )
    ];
    Print[x];
    CaPutWF[in, x];
    ListPlot[x,PlotJoined->True,Frame->True,
                    FrameLabel->{in, "Value","PLOT OF SINE WAVE "," "}]
 )

cosine::usage = "cosine[\"devname\"] - set and plot the sine wave form\n"

cosine[in_String]:=(
    npoints=CaGetCount[in];
   x=Table[0,{npoints}];
    For[i=1,i<=npoints,i++,
      (  ii=Pi*2/npoints*i;
        x[[i]]=N[Cos[ii]];
       )
    ];
    Print[x];
    CaPutWF[in, x];
    ListPlot[x,PlotJoined->True,Frame->True,
                    FrameLabel->{in, "Value","PLOT OF COSINE WAVE "," "}]
 )

updown::usage = "updown[\"devname\"] - set and plot the sine wave form\n"

updown[in_String]:=(
    npoints=CaGetCount[in];
   x=Table[0,{npoints}];
    For[i=1,i<=npoints,i++,
      (  x[[i]] = If[i>10,20-i,i];
        x[[i]] = If[i>30,i-40,x[[i]]];
```

```
    )
  ];
  Print[x];
  CaPutWF[in, x];
  ListPlot[x,PlotJoined->True,Frame->True,
              FrameLabel->{in, "Value","PLOT OF COSINE WAVE "," "}]
)

downup::usage = "downup[\"devname\"] - set and plot the sine wave form\n"

downup[in_String]:=(
  npoints=CaGetCount[in];
  x=Table[0,{npoints}];
  For[i=1,i<=npoints,i++,
    (  x[[i]] = If[i>10,i-20,-i];
       x[[i]] = If[i>30,40-i,x[[i]]];
    )
  ];
  Print[x];
  CaPutWF[in, x];
  ListPlot[x,PlotJoined->True,Frame->True,
              FrameLabel->{in, "Value","PLOT OF COSINE WAVE "," "}]
)
```

7 Timing of CaGet / CaPut in Mathematica

The average execution times in seconds for CaGet , CaPut, CaGetString, CaPutString are listed below. The CaPendEvent[0.001] is set first before the following timing analysis.

CaGet	**CaGetString**	**CaPut**	**CaPutString**
0.016	0.017	0.009	0.007

8 CaMath Related Files

Below is a list of the related CaMath files required for running and linking with Mathematica. Information on how to prototype and generate the mathlink external functions can be found in reference 3.

setupmath	The unix script file sets up the CaMath environment for users.
CaMath	The mathlink external program consists of various channel access functions.
devMath	The mathlink external program consists of device access library functions interfacing with CaMath function.
CaMath.m	The script file used by Mathematica to install the external program package CaMath and devMath.
caDef.m	The template script file used by Mathematica to define the EPICS alarm status and severity codes.

9 References

[1] Wolfram, Stephen. Mathematica: A System for Doing Mathematics by Computer. Reading, MA: Addison-Wesley, 1991 (2nd edition).

[2] Hill, Jeff. Channel Access Reference Manual. Technical Note AT8-89SYS001, LANL, September 1991.

[3] Wolfram Research. Mathematica Technical Report : MathLink External Communication in Mathematica.

[4] Saunders, Claude W. Device Access Library : User's Guide and Reference (Draft).