CMLOG Documentation
Version 2.x

Jie Chen
William Watson III

chen@jlab.org
watson@jlab.org

July 25, 2001

1 What Is CMLOG ?

1.1 Description

CMLOG is a distributed message logging system which could be used by any application or system desiring to log messages to centralized log files, and display distributed messages on set of displays. It supports C++ and C application interfaces for logging messages (*Logging Client APIs*) , and has C++ application interfaces for searching/retrieving messages from a dedicated logging server(*Browser APIs*).

1.2 Features

- Messages being logged are of type cdevData that is flexible to allow applications to log data of any types.
- Logging client APIs are in a single library called libcmlog.a (so). It has C++ and C callable routines that allow any applications to log messages to the server.
- Logging clients send messages to a client daemon that buffers all incoming messages from all clients on the host and sends messages to the server.
- Logging client APIs have a CDEV interface.
- Logging client APIs can be safely called inside *Threads*.
- Logging client APIs can be safely called inside *Interrupt Service Routines* (ISR).
- Applications that wish to search/retrieve messages from the server can use a single C++ library libcmlogb.a.
- Browser API provides server crash notification mechanism.
- All messages in the logging files are time stamped and organized in B+ tree that allow fast data look up by time
- The server is implemented in Multi-Threaded (Multi-Process) fashion to improve network throughput and responsiveness
- The server supports asynchronous network I/O operations.
- The server supports runtime configuration.
- A sample Motif implementation of browser is provided.

1.2.1 Server Configuration Parameters

A CMLOG server can take a configuration file to configure itself to suit to different site. The followings summarize these parameters.

Name/Description	Examples
cmlogServerPort	8900
A server runs on this UDP port.	
cmlogMaxClntConnections	64
Maximum number of machines connecting to a server.	
cmlogMaxBrowserConnections	16
Maximum number of browsers connecting to a server.	
cmlogDefaultDirectory	/tmp
A server runs in this directory.	
cmlogLogFile	cmlogServer.log
The server log file resides in the above directory.	
cmlogDatabaseName	/usr/local/cmlog_%s
The prefix for server database filenames.	
cmlogSecondaryDatabase	/usr/local/1997/cmlog_%s
The prefix for server secondary database filenames.	
cmlogCxtDatabaseName	/usr/local/cmlogcxt_%s
The prefix for server context database filenames.	
cmlogSecondaryCxtDatabase	/usr/local/1997/cmlogcxt_%s
The prefix for server secondary context database filenames.	
cmlogTagTableName	/usr/local/cmlogTagTable
Tag table file name for a server.	
cmlogDbaseChangeInterval	1440
Database name changes every 1440 minutes (24 hrs).	

1.2.2 Run Time Environment Variables

There are two run time environment variables that effect initial handshakes among server, clients and browsers. The CMLOG_HOST environment variable denotes that a server is running on this host. All clients and browser send udp packet to this host instead of using broadcast. The CMLOG_PORT environment variable denotes a UDP port on which a server is running.

1.3 Platforms

CMLOG has been compiled for Solaris, Solaris-lp64 (64 Bit), HP_UX (9.x), HP_UX (10.x) using either CC or aCC and RedHat Linux on Intel and Alpha (6.x & 7.x). The logging client API has been compiled and tested on mv162(7), PowerPC (many thanks to David Abbott of DAQ group at Jefferson Lab), niCpu30 (Ported by Ron MacKenzie at SLAC), and pc486 (Ported by Peregrine M. McGehee at LEDA Project of LANL) running VxWorks 5.2(3).A Makefile and Makefile.config are provided. These files also provide starting point to port CMLOG to other platforms.

1.4 Limitations

1.4.1 Limit on Multi-Servers in a single subnet

The CMLOG server runs on a well known UDP port. All logging clients and browsers use a broadcast message to the port to establish network connection to the server. If one wishes to have more than one logging server on the net, one has to use run time configuration method to start a different server. All logging clients and browsers can use CMLOG_PORT environment variable to communicate with new server.

1.4.2 Limit on recovery of lost messages

In case of server crash, all messages from logging clients that are still logging messages will be lost. In future, CMLOG will provide a way to save these messages which then can be merged into the centralized database by a simple utility.

Name/Description	Value	Where
CMLOG_CLNT_PORT	8909	cmlogConfig.h
CLient Daemons run at this UDP port.		
CMLOG_MIN_KEY_PAGE	10	cmlogConfig.h
Minimum number of keys on page.		
CMLOG_PAGE_SIZE	4096	cmlogConfig.h
Database page size.		
CMLOG_CLNT_PIPE	/pipe/cmlog_1	cmlogConfig.h
The pipe device on vxworks .		
CMLOG_CLNT_PIPE	/tmp/cmlog_ip	cmlogConfig.h
The named pipe on Unix machines.		
CMLOG_NUM_PROC_THREADS	10	cmlogConfig.h
Number of threads to process incoming I/O messages.		
CMLOG_NUM_SVC_PROC	2	cmlogConfig.h
Number of processes to process query messages.		
CMLOG_DSHMEM_SIZE	80 k	cmlogConfig.h
Size of a shared memory holding logging messages.		
CMLOG_CSHMEM_SIZE	20 k	cmlogConfig.h
Size of a shared memory holding querying messages.		
CMLOG_DBASENAME_SIZE	128	cmlogConfig.h
Maximum size of database file names.		

Table 1: Most of static parameters for cmlog system.

Note: Linux uses clone system call to implement pthread. If cmlogServer is compiled using pthread on linux, limit CMLOG_NUM_PROC_THREADS to a small number.

1.4.3 Limit on logging files

All logging files must be in locations specified either by cmlogDatabaseName (cmlogCxtDatabaseName) or by cmlogSecondaryDatabase (cmlogSecondaryCxtDatabase) in the server configuration file. The format of cmlogSecondaryDatabase can be either a single full file path that is not the same as the cmlogDatabaseName or a list of full file paths. For example, one can specify the following way to tell the server to search three directories:
cmlogDabaseName: /cmlog/file/data/cmlog_%s
cmlogSecondaryDatabase: /cmlog/file/data/1997/cmlog_%s:/cmlog/file/data/1998/cmlog_%s

1.4.4 Limit on query capabilities

Currently one can retrieve all messages logged into the logging files within a specified time window. The CMLOG currently does have simple querying methods that are similar to logic expressions in C language.

Name/Description	Value
CMLOG_SUCCESS	0
Execution finished successfully.	
CMLOG_ERROR	-1
Execution finished abnormally.	
CMLOG_INVALIOBJ	1
Invalid cdev object.	
CMLOG_INVALIDARG	2
Caller uses wrong arguments.	
CMLOG_INVALIDSVC	3
Wrong cdev service	
CMLOG_NOTCONNECTED	4
Logging clients or browsers are not connected to the server.	
CMLOG_IOFAILED	5
Low Level I/O operations failed.	
CMLOG_CONFILICT	6
conflicts of data types or tags	
CMLOG_NOTFOUND	7
Query process find no answer.	
CMLOG_TIMEOUT	8
Connection timeout	
CMLOG_CONVERT	9
cdevData conversion error	

Table 2: Error codes of cmlog system.

1.4.5 Limit from static parameters

Some of parameters are hard coded in the server. They can be easily changed to accommodate the needs of different sites. These parameters are usually related to run time characters of the server. They may influence the performance of the server. We may study the effect of these parameters on performance in the coming months. See Table 1.

2 Application Program Interfaces

2.1 Introduction

This section describes basic functions application programmers can use to log messages to or retrieve messages from a server.

2.2 CMLOG Error Code

All CMLOG APIs use a set of error code to describe the execution status of a method. Table 2 and Table 3 summarize of the error code which can be found in cmlog.h.

CMLOG_NOTCONSERVER	80
Cannot connect to server.	
CMLOG_NOTFOUNDSERVER	81
Cannot find server anywhere.	
CMLOG_CONN_TIMEOUT	82
Connection timeout.	
CMLOG_FILTERED	83
Logging messages have been filtered.	
CMLOG_NOFILTERING	84
No filtering applied.	
CMLOG_DROPPED	85
Messages are dropped.	
CMLOG_BADIO	86
Writing/reading on a bad socket.	
CMLOG_OVERFLOW	87
Messages overflow the buffer (vxWorks).	
CMLOG_INCOMPLETE	88
Unfinished IO.	
CMLOG_CBK_FINISHED	89
IO request is finished.	
CMLOG_QUERY_PAUSED	90
Query callback paused by the browser.	
CMLOG_QUERYMSG_ERR	91
Query Message Syntax error.	
CMLOG_THREAD_ERR	92
Query Thread Initialization error.	
CMLOG_QUERY_CANCELED	93
Long Query Request is Canceled.	

Table 3: More error codes of cmlog system.

2.3 Logging Client API

2.3.1 C++ interface

C++ interfaces for logging clients are organized into a single C++ class called **cmlogClient**.

- `cmlogClient (char* progname)`
  - Description

This is a constructor for cmlogClient. This can be only accessed when UNIX applications are compiled. It takes an argument that denote the program name. If no progname is given, default name from command argument will be used on Unix machines.

- `cmlogClient* logClient (void)`
  - Description

    This is a constructor for cmlogClient on vxWorks.

- `~cmlogClient (void)`
  - Description

    This is the destructor for cmlogClient. It will close network connection and free all resources.

- `int connect (int connetionRetries = 3)`
  - Description

    This method establishes network connection between the logging client and the client daemon. It tries to connect to the daemon in 'connectionRetries' times. If there is no client daemon present, the routine will try to start one. It returns CMLOG_SUCCESS upon successful connection, and returns CMLOG_ERROR upon failure. This routine also sends additional information such as user name, process id to the server once the connection is established.

- `int connect (cdevData& cxt, int connetionRetries = 3)`
  - Description

    This routine is very similar to the above routine except that callers have control over what to send to the server upon connection.

- `int disconnect (int connetionRetries = 3)`
  - Description

    This routine disconnect logging clients from the client daemon. It returns CMLOG_SUCCESS upon success, and returns CMLOG_ERROR upon failure.

- `int logMsg (char* format, ...)`
  - Description

    A convenience routine to log a message to the server. The format string must be in the form of tagged items. It returns CMLOG_SUCCESS, CMLOG_FILTERED or CMLOG_NOCONSERVER.
  - Example

```
logMessages (void)
{
cmlogClient client;

if (client.connect () == CMLOG_SUCCESS)
client.logMsg (``severity = \%d text = \%s'', 10, ``hello'');
}
```

- `int postError (int verbosity, int severity, int code, char* facility, char* format, ...)`
  - Description

    This routine offers another way to log message with given verbosity, severity, error code, and facility name. The format still must be in the form of tagged items as described in the above routine.

- `int postData (cdevData& data)`
  - Description

    This routine is a general routine to allow programs to log any cdevData into the database. If the data does not contain a valid time stamp, a current time stamp will be issued.

- `int postStaticData (cdevData& data)`
  - Description

    This routine is a general routine to allow programs to log a static cdevData into the database. This rouine is useful for vxworks.

- `int setThrottle (char* tag, int size, int limit, double dt)`
  - Description

    This routine is used to set throttling on a particular tag. After this routine has been called, only 'limit' number of messages will be logged within time interval 'dt'. The 'size' is size of the list holding all throttled messages. It return CMLOG_SUCCESS or CMLOG_ERROR.Note a subsequent this call overrides previous call. In addition number of dropped messages is saved in the subsequent messages using a tag of "dropped".

- `int getThrottle (char** tag,int& size,int& limit,double& dt)`
  - Description

    This routine is used to get throttling parameters. It returns CMLOG_SUCCESS if there is a throttling on a particular tag (Callers should free tag after using it). It returns CMLOG_ERROR if there is no throttling.

- `int setSeverityThreshold (int thresh)`

  - Description

    This routine sets severity threshold. Any messages with severity less than the threshold will not be logged. It return CMLOG_SUCCESS and CMLOG_ERROR.

- `int getSeverityThreshold (void) const`
  - Description

    This routine gets severity threshold.

- `int setVerbosityThreshold (int thresh)`
  - Description

    This routine sets verbosity threshold. Any messages with verbosity larger than the threshold will not be logged. It return CMLOG_SUCCESS and CMLOG_ERROR.

- `int getVerbosityThreshold (void) const`
  - Description

    This routine gets verbosity threshold.

- `int logMsgI (char* format, int arg0-9)`
  - Description

    This routine is designed for use in the Interrupt Service Routines of vxWorks only. The format must be in the form of tagged items only. It return CMLOG_SUCCESS or CMLOG_ERROR.

- `int postErrorI (int verbosity, int severity, int code, char* facility, char* format, int arg0-5)`
  - Description

    This routine is designed for use in the Interrupt Service Routines of vxWorks only. The format must be in the form of tagged items only. It return CMLOG_SUCCESS or CMLOG_ERROR.

2.3.2 C Interface

C interfaces of logging clients can be used through a new data type **cmlog_client_t**. Any programs that wish to log messages to the server have to create a handler of type cmlog_client_t. All subsequent calls use the handle to log messages.

- `cmlog_client_t cmlog_open (char* progname)`
  - Description

    Establish network connection to the client daemon. It returns 0 if there is no connection established.

- `void cmlog_close (cmlog_client_t handle)`

  - Description

    Close network connection.

- `int cmlog_logmsg (cmlog_client_t handle, int verbosity, int severity, int code, char* facility, char* format, ...)`
  - Description

    Log messages with given verbosity, severity, code, and facility. The format string must be in the form of tagged items.

- `int cmlog_logtext (cmlog_client_t handle, char* format, ...)`
  - Description

    Log messages described with the format string. The format string must be in the form of tagged items.

- `int cmlog_logtextI (cmlog_client_t handle, char* format, int arg0-9)`
  - Description

    A special routine for ISR of vxWorks.

- `int cmlog_logmsgI (cmlog_client_t handle, int verbosity, int severity, int code, char* facility, char* format, int arg0-5)`
  - Description

    A special routine for ISRs of vxWorks.

- `int cmlog_set_severity_threshold (cmlog_client_t handle, int threshold)`
  - Description

    Set severity threshold for all messages. Any messages with severity smaller than threshold will not be logged.

- `int cmlog_get_severity_threshold (cmlog_client_t handle, int* threshold)`
  - Description

    Get severity threshold for all messages.

- `int cmlog_set_verbosity_threshold (cmlog_client_t handle, int threshold)`
  - Description

    Set verbosity threshold for all messages. Any messages with verbosity greater than threshold will not be logged.

- `int cmlog_get_verbosity_threshold (cmlog_client_t handle, int *threshold)`
  - Description

    Get verbosity threshold for all messages.

- `int cmlog_set_throttle (cmlog_client_t handle, char* tag, int valuerange, int limit, double dt)`
  - Description

    This routine is used to set throttling on a particular tag. After this routine has been called, only 'limit' number of messages will be logged within time interval 'dt'. The 'size' is size of the list holding all throttled messages. It return CMLOG_SUCCESS or CMLOG_ERROR. Note a subsequent this call overrides previous call. In addition number of dropped messages is saved in the subsequent messages using a tag of "dropped".

- `int cmlog_get_throttle (cmlog_client_t handle, char** tag, int *valuerange, int *limit, double *interval)`
  - Description

    This routine is used to get throttling parameters. It returns CMLOG_SUCCESS if there is a throttling on a particular tag (Callers should free tag after using it). It returns CMLOG_ERROR if there is no throttling.

2.3.3 Some Simple Examples

The following is a simple C++ example to log messages to the CMLOG server.

```
#include <cmlogClient.h>
#include <unistd.h>

#ifndef __vxworks
main (int argc, char** argv)
#else
clientTest (char* name)
#endif
{
    int status;
#ifndef __vxworks
    cmlogClient *client = new cmlogClient (argv[0]);
#else
    cmlogClient *client = cmlogClient::logClient ();
#endif

    if (client->connect () == CMLOG_SUCCESS) {
        cdevData::insertTag (322, "highVIn");
        cdevData::insertTag (323, "highVOut");

        client->setThrottle ("highVIn", 100, 25, 1.0);

        cdevData data;
        for (int i = 0; i < 100; i++) {
             sprintf (temp, "Test program error happend %d\n", i);
```

```
            status = client->postError (32, 12, CMLOG_ERROR,
                                        "EPICS",
                                        highVIn = %d highVOut = %d text = %s",
                                        i, i*100, temp);
    }
    client->disconnect ();
  }
  else
    printf ("Cannot connect to daemon\n");
}
```

The following is the sample makefile for the above C++ code (clientTest.cc) on a Unix machine (Let us assume this machine is a hpux machine).

```
CXX = CC
CXXFLAGS = -I$(CMLOG)/include -D_CMLOG_BUILD_CLIENT

OBJS = clientTest.o
LIBS = -L$(CMLOG)/lib/hpux -lcmlog

all: clientTest

clientTest: $(OBJS)
        rm -f $@
        $(CXX) -o $@ $(LIBS)

.cc.o:
        rm -f $@
        $(CXX) $(CXXFLAGS) -c $< -o $@

clean:
        rm -f *.o clientTest core *~
```

The following is the sample makefile for the above C++ code (clientTest.cc) on a mv162 machine (Let us assume this board is running vxWorks 5.2)

```
CXX = c++68k
CXXFLAGS = -I$(CMLOG)/include -D_CMLOG_BUILD_CLIENT -O \
        -I$(VX_VW_BASE)/h -fstrength-reduce -fforce-mem \
        -finline-functions -fno-builtin -fno-for-scope -nostdinc \
        -DCPU=MC68040 -DCPU_FAMILY=MC680X0 -ansi -pipe \
        -Dvxworks -msoft-float

OBJS = clientTest.o

MAKELIB = ld68k -r -o

all: clientTest
```

```
clientTest: $(OBJS)
        rm -f $@
        $(MAKELIB) -o $@ $(OBJS)

.cc.o:
        rm -f $@
        $(CXX) $(CXXFLAGS) -c $< -o $@

clean:
        rm -f *.o clientTest core *~
```

The following is a simple c code example using the CMLOG client APIs to log messages.

```
#include <stdio.h>
#include <string.h>
#include <cmlog.h>

#ifndef __vxworks
main (int argc, char** argv)
#else
client_test (char* name)
#endif
{
  int status;
  cmlog_client_t cl;
  int i = 0;
  char message[256];

#ifndef __vxworks
  cl = cmlog_open (argv[0]);
#else
  cl = cmlog_open (name);
#endif

  if (cl == 0) {
    fprintf (stderr, "cannot open cmlog client\n");
    exit (1);
  }

  for (i = 0; i < 100; i++) {
    sprintf (message, "Test error c interface at %d\n", i);
    status = cmlog_logmsg (cl, 32, 12, CMLOG_ERROR,
                           "EPICS", "value = %d status = %d text = %s",
                           i*10, 0, message);
#ifndef __vxworks
    sleep (1);
```

```
#else
    taskDelay (sysClkRateGet ());
#endif
  }

  cmlog_close (cl);
}
```

Once again we show a example Makefile for the above c test code (client_test.c).

```
CXX = CC
CC  = cc
CFLAGS = -I$(CMLOG)/include -D_CMLOG_BUILD_CLIENT
OBJS = client_test.o
LIBS = -L$(CMLOG)/lib/hpux -lcmlog

all: client_test

client_test: $(OBJS)
	rm -f $@
	$(CXX) -o $@ $(OBJS) $(LIBS)

.c.o:
	rm -f $@
	$(CC) $(CFLAGS) -c $< -o $@

clean:
	rm -f *~ *.o core client_test
```

The following is the sample makefile for the above c code (client_test.c) on a mv167 machine (Let us assume this board is running vxWorks 5.3)

```
CC = cc68k
CFLAGS = -I$(CMLOG)/include -D_CMLOG_BUILD_CLIENT -O \
	-I$(WIND_BASE)/target/h -fstrength-reduce -fforce-mem \
	-finline-functions -fno-builtin -nostdinc \
	-DCPU=MC68040 -DCPU_FAMILY=MC680X0 -ansi -pipe \
	-Dvxworks

OBJS = client_test.o

MAKELIB = ld68k -r -o

all: client_test

client_test: $(OBJS)
	rm -f $@
	$(MAKELIB) -o $@ $(OBJS)
```

```
.c.o:
	rm -f $@
	$(CXX) $(CFLAGS) -c $< -o $@

clean:
	rm -f *.o client_test core *~
```

2.3.4 CDEV Interface On Unix

A cmlogService.so can be built if CDEV package is present. A simple extension to existing DDL file allows caller to make cdev call to log messages to the server. The extension looks like the following:

```
service cmlog {
        tags {PV}
}

class CMLOG  {
    verbs {set}
    attributes {
msg cmlog {};
       }
}
CMLOG :
cmlog
;
```

Then a typical cdev call can be used to log messages.

```
#include <cdevSystem.h>
#include <cdevRequestObject.h>
#include <cdevData.h>

int main (int argc, char** argv)
{
    cdevSystem& system = cdevSystem::defaultSystem ():

    cdevData data;
    cdevRequestObject* obj;
    obj = cdevRequestObject::attachPtr (``cmlogClient'',``set msg'');
    if (obj) {
        data.insert (``severity'', 10);
        data.insert (``text'', ``error happend'');
        obj->send (data, 0);
    }
}
```

2.3.5 VxWorks Issues

CMLOG logging client APIs are very easy to use on vxWorks. However one has to load cmlogClientD (client daemon) before loading libcmlog.a when one boots vxWorks target. Two convenient routines of cmlogVxLogDisable() and cmlogVxLogEnable() will disable and enable cmlog system respectively. In addition all tasks share a single tag table and a single cmlogClient.

3 Browser APIs

CMLOG provides a set of C++ APIs for applications to retrieve messages previously logged in the database. The I/O requests sent from browsers to the server are in the form of cdevData that has *start* and *end* tagged values specifying the time interval within which the query is performed. In addition caller can also specify a selection rule which can be inserted into the outbound cdevData with tag *queryMsg*. Most of APIs are organized in a C++ class called *cmlogBrowser*. In addition there is one class called *cmlogPacket* which holds array of *cmlog_cdevMessage*s. In spite of the complex nature of *cmlog_cdevMessage*, only a single member function *getData()* is used in the APIs. The following subsection will describe only related member functions of *cmlogPacket* and *cmlog_cdevMessage*.

3.1 Related C++ Classes

- `cmlog_cdevMessage`
  - `cdevData* getData (void)`
    - Description
      Returns a pointer of cdevData that is contained by cmlog_cdevMessage.
- `cmlogMessage`
  - `operator cmlog_cdevMessage& (void)`
    - Description
      Convert cmlogMsg to cmlog_cdevMessage by returning a reference of cmlog_cdevMessage.
- `cmlogPacket`
  - `cmlogMsg** messages (void)`
    - Description
      Returns a list of *cmlogMsg*s that is a wrapper of *cmlog_cdevMessage*. Callers should free all items in the list, and list itself.
  - `unsigned long numberOfData (void)`
    - Description
      Returns number of *cmlogMsg*s in the above list.

The following is a simple example that gets array of *cdevData*s from a pointer to a *cmlogPacket*.

```
#include <cmlogBrowser.h>
```

```
main (int argc, char** argv)
{
    cmlogPacket* packet;
    cdevData* data;

    /* assume we have this cmlogPacket through some ways */
    if (packet) {
        cmlogMsg** msgs = packet->messages ();
        for (int i = 0; i < packet->numberOfData (); i++) {
            cmlog_cdevMessage& imsg = (*msgs[i]);
            data = imsg.getData ();
        }

        for (i = 0; i < data->numberOfData (); i++)
            delete msgs[i];
        delete []msgs;
    }
}
```

3.2 cmlogBrowser

- `Definition of callback function`

  typedef void (*cmlogBrCallback)(int status, void* arg, cmlogPacket* data);

- `cmlogBrowser (void)`
  - Description

    Constructor for *cmlogBrowser*.

- `~cmlogBrowser (void)`
  - Description

    Destructor for *cmlogBrowser*. If network connection is established, this closes network connection and clean all resources.

- `int connect (int numConnectionRetries = 3)`
  - Description

    Tries to connect to a CMLOG server 'numConnectionRetries' times. It returns CMLOG_SUCCESS upon connection, and returns CMLOG_ERROR if it fails to connect.

- `int disconnect (void)`
  - Description

    Disconnects this browser from a server. It returns CMLOG_SUCCESS.

- `int connected (void) const`

- Description

  Returns 1 for connected browser.

- `int disconnectCallback (cmlogBrCallback cbk, void* arg)`

  - Description

    Registers a callback function to this browser. In case of server crash, this function will be called. It returns CMLOG_SUCCESS upon success, and returns CMLOG_ERROR if this callback function is already registered.

- `int queryCallback (char* msg, cdevData& data, cmlogBrCallback cbk, void* arg)`

  - Description

    This routine is used to query the server to find messages. Currently only four messages are supported. They are **"query"**, **"stopQuery"**, **"monitorOn []"** and **"monitorOff []"** where []s denote attributes of a CMLOG server. At present there two attributes that can be monitored. One is **loggingData** which symbolizes currently incoming logging data to the server. Another is **allTags** that holds all tag strings and corresponding tag values of the server.
    The cdevData object 'data' must contain time interval which is specified by **"start"** and **"end"** tagged values. In addition it may contain a tagged item with tag **"numberItems"** to tell a server how many messages a browser wants to retrieve and it may contain a tagged item with tag **"queryMsg"** to tell server to only return messages that satisfy the conditions specified by the query message.

- Example

  Here is an example that does simple query.

```
char* rules = ``severity==1 && status == 0'';
cdevData data;
/* 861759125 represents Tue Apr 22 21:32:45 EDT 1997 */
data.insert (``start'', 861759125);
data.insert (``end'',   861759445);
data.insert (``numberItems'', 100);
data.insert (``queryMsg'', rules);

browser.queryCallback (``query'', data,
                                      callback, (void *)0);
```

- `int getFd (void) const`

  - Description

    Returns the socket file descriptor to callers. This socket file descriptor can then be used inside a event loop to monitor network IO of browsers.

- `int pendIO (double seconds)`

  - Description

Checks network I/O events upto 'seconds' long. If seconds equals to 0.0, it becomes a polling routine. It returns CMLOG_SUCCESS, CMLOG_TIMEOUT, CMLOG_BADIO or CMLOG_IOERROR.

- `int pendIO (void)`
  - Description

    It is very similar to the above. It waits forever for network I/O events. It returns CMLOG_BADIO, CMLOG_IOERROR or CMLOG_SUCCESS.

3.3 A Simple Browser Example

Here is a simple example that does a single query to a server.

```
#include <cmlogBrowser.h>

static void qcallback (int status, void* arg, cmlogPacket* data)
{
    cdevData* res = 0;
    int severity;
    char text[1024];
    char host[80];


    if (data) {
        cmlogMsg** msgs = data->messages ();
        for (int i = 0; i < data->numberOfData (); i++) {
             cmlog_cdevMessage& idata = (*msgs[i]);
             res = idata.getData ();

             if (res) {
                 res->get ("host", host, sizeof (host));
                 res->get ("severity", &severity);
                 res->get ("text", text, sizeof (text));

                 printf ("From Host %s with Severity %d: %s\n", host, severity, text);
             }
        }
        for (i = 0; i < data->numberOfData (); i++)
             delete msgs[i];
        delete []msgs;
    }
}

char *selection = ``facility == 'EPICS' && severity == 10'';
```

```
main (int argc, char **argv)
{
    cmlogBrowser browser;
    char         command[1024];
    char         attr[80];
    int          status;

    if (browser.connect () == CMLOG_SUCCESS) {
        printf ("Connect to the cmlogServer\n");

        printf ("Enter cmlog test command\n");
        scanf ("%s", command);
        if (strcasecmp (command, "query") == 0) {
            printf ("Enter start and end time in double value\n");
            double start, end;
            scanf ("%lf %lf", &start, &end);

            cdevData data;
            data.insert ("start", start);
            data.insert ("end", end);
    data.insert (‘‘queryMsg’’, selection);

            status = browser.queryCallback (command, data, qcallback, 0);
            browser.pend ();
       }
    }

}
```

The following is a sample Makefile for above C++ code (browserTest.cc) on a Unix machine (let us assume this machine is a Solaris)

```
CXX = CC
CXXFLAGS = -I$(CMLOG)/include -D_CMLOG_BUILD_CLIENT

OBJS = browserTest.o
LIBS = -L$(CMLOG)/lib/solaris -lcmlogb -lsocket -lnsl

all: browserTest

borwserTest: $(OBJS)
        rm -f $@
        $(CXX) -o $@ $(LIBS)

.cc.o:
        rm -f $@
        $(CXX) $(CXXFLAGS) -c $< -o $@
```

```
clean:
        rm -f *.o browserTest core *~
```

3.4 Query Messages from Browser

Applications may send a selection rule to the server to filter out messages that do not qualify. A selection rule is specified as a character string with syntax of C logic expressions using known tag names as variables of expressions. For example the following selection specifies that a caller wants messages to have facility "epics" and severity with value 3:
facility=='epics' && severity==3
In addition pattern match can be performed similar to SQL such as "text like 'error message"'. If applications provide wrong syntax, the server will send CMLOG_QUERYMSG_ERR back. The following is the definition of syntax for query messages:

```
messages : message
| messages && messages
| messages || messages
| ! messages
| (message)

message : tagname logicop tagvalue
```

where logicop could be ==, <, >, ! =, <=, >= and like, and tagvalue can be data of type of int, double, float, and single quoted char string.

4 Sample Motif Browser: cmlog

A sample X window Motif implementation of browser is implemented. This browser supports monitoring/querying with multiple windows. Currently it only supports simple query with specified time interval with logic expressions as selection message. The browser is configured by a configuration file that specifies what tags the browser is interested. The geometry layout of browser can be adjusted to fit widths of tagged values.

The following command line options are supported as of version 2.0:

- -c: Automatic connect to the server.
- -u: Automatic connect to the server and put in the updating mode.
- -q: Automatic connect to the server and perform a query with specified query message and a time interval from config file.
- -hu: Does everything as the previous option plus putting in the updating mode when the query is finished.
- -f: Use a user specified configuration file.
- -cfg: Use a user specified configuration file.
- -h Print out command line help menu.

The cmlog searches a configuration file in the following order: 1) -f or -cfg, 2) .cmlogrc in the user home directory and 3) CMLOG_CONFIG environment variable specifying a file location. If syntax errors are detected during start up of cmlog, the default setting will be used.

The cmlog chooses a default file to save current settings in the following way. If there is a configuration file provided other than the CMLOG_CONFIG file, the configuration file will be used as the default. If there is no configuration file, .cmlogrc in the user home directory will be the default file. Finally if the configuration file is the CMLOG_CONFIG, the .cmlogrc in the user home directory is the default.

In addition the cmlog allows users to save current display into an ASCII file from "Save Screen" menu, to cancel a long query, to convert set of integer values of a tag to strings, to display values in different colors and to blink, to bell or to do an action (user script) color coded values (bell and user actions will only be effective in the updating mode).

The following is a sample configuration file for cmlog:

```
# Configuration file for cmlog
# Name of this config
name MCC Control
# Type          Title          Tag          Width
Col          Facility                  facility                98
Col          Host                      host                    90
Col          User                      user                    48
Col          Time                      cmlogTime               178
Col          Status                    status                  49
Col          Severity                  severity                79
Col          Message                   text                    250
# Buffer size for updating the message
ubufsize     2000
# Default query message
queryMessage    ``none''    -36:0 now
# Default update selection message
updateMessage    ``none''
# Window Width
windowWidth  977
# Window Height
windowHeight  500
#Code Conversion
codeConversion ``none'' Severity
{
0          Info         Green
1          Info1        Blue
2          Info2        LightBlue
3          Warning      Yellow       flash:bell:action
4          Fatal        Red          flash
}
#Code Conversion
```

```
codeConversion ''Facility == 'EPICS' '' status
{
100         Info       Blue
101         Info1      Blue
102         Info2      LightBlue
103         Warning    Yellow      blink:bell:dosomething
104         Fatal      Red          doother:flash
}
```

Note: "none" in queryMessage or updateMessage stands for no message.
Note: End time "now" in queryMessage stands for current time.
Note: Correct syntax to specify time interval in the queryMessage can be one of the followings: "H stands for hour, M stands for minutes, m stands for months, d stands for days in a month and Y stands for long year (1999 not 99).
Note: The second field of codeConversion line identifies sources of data. In particular cmlog converts integer fields to string representations the same way regardless data sources if "none" is specified.
Note: The first column within a code conversion denotes integer values that will be converted to string representations that are shown in the second column. The values in third column give background color of those strings. The optional value in the forth column enables blinking of color coded values, ringing a bell or doing a user defined action.

5 Miscellaneous Information

This section summaries all executables and libraries of CMLOG.

- Executables for Unix Machines
  - cmlogServer

    This is the server for CMLOG system. It takes one argument that is configuration file. If no argument is provides, default configuration parameters are used.
  - cmlogClientD

    This is the client daemon running on a Unix machine. We suggest this process is started by root. The cmlogClientD can also be compiled using _CMLOG_CLIENTD_AUTO_QUIT option to tell the clientDaemon to quit when the cmlogServer exits.
  - cmlog_activate

    This is a csh script to start cmlogClientD.
  - cmlog

    This is the Motif browser for CMLOG system.
  - cmlogMsg

    This is a simple message logging porgram to let users log messages into cmlog system. It may be used as a quick test for cmlog system. It can be invoked as cmlogMsg "any messages".
  - cmlogCleanup

This exists only on HP_UX and Linux. It cleans all IPC resources if the server crashes unexpectly.

- cmlogAdmin

  This is an administration utility. It is partially finished. It can be used to shutdown the server and all client daemons and to check some statistical information of the server.

- Executables for vxWorks
  - cmlogClientD

    This is the clientDaemon running on a vxWorks target. It runs at priority 190. This must be loaded first before any other CMLOG libraries/executables.

  - cmlogVxLogMsg

    This is a utility that can catch all *logMsg* calls of vxWorks. It then will send catched messages to the above daemon. It runs at priority of 200.

- Libraries for Unix Machines
  - libcmlog.a(so)

    This is the logging client library. Applications need to link with this library to log messages to a server.

  - libcmlogb.a

    This is the browser library.

  - cmlogService.so

    This is the cmlog service layer for CDEV.

- Libraries for vxWorks
  - libcmlog.a

    This is the logging client library. Applications need to link with this library to log messages to a server. This library has to be loaded after cmlogClientD.

6 Acknowledgment

Special thanks to Ron MacKenzie at SLAC who has spent long hours to design, discuss and test cmlog version 2.0. Many thanks to David Abbott at Jefferson Lab for his effort to port and test cmlog on different platforms running VxWorks and Linux. Many thanks to Johannes van Zeijts at BNL and Thomas Birke at BESSY for their testing and valuable suggestions. Finally thanks to anyone who have made any suggestion, comment and feedback to the system.