APS logDaemon and Client Library

Claude Saunders, Jim Kowalkowski

Table of Contents

Operations Analysis Group, Controls Group
Accelerator Systems Division
Argonne National Laboratory
Dec. 13, 1995

1. Introduction

This document serves as a User's Manual and Reference for the logDaemon and client library. This package provides a general distributed message logging system. A logDaemon may be started anywhere on a subnet. A client which has linked in the client library is provided functions to open a connection to the logDaemon, log messages, and close the connection. The logDaemon maintains one or more log files (in simple ascii or SDDS format) and an e–mail list based on specifications in a configuration file. Incoming messages are logged to the appropriate file and/or result in e–mail being sent.

1.1. Client Overview

The client library provides the following calls:

```
int logOpen(LOGHANDLE *h, char *sourceId, char *serviceId);
int logGetVerbosity(LOGHANDLE h);
int logSetVerbosity(LOGHANDLE *h, int verbosity);
int logMessage(LOGHANDLE h, char *system, char *subsystem,
        int verboseLevel, char *format, ...);
int logClose(LOGHANDLE h);
```

A connection to the logDaemon is opened with logOpen(). The user provides a pre–allocated LOGHANDLE for use in subsequent calls. The sourceId is an arbitrary string designed to identify the general class of user (ie. IOC, SCRIPT, etc..). The serviceId ptr may be NULL, in which case the default logDaemon is contacted. Alternately, a specific logDaemon may be requested by name (must agree with name given logDaemon at startup).

The LOGHANDLE is then valid for all other calls until logClose() is called on it, at which time it may be re–used.

All messages have a verbosityLevel from 1 to 99. By default, all levels are accepted by logMessage(). You may disallow messages above a certain level via the logGetVerbosity() and logSetVerbosity() calls. Any logMessage() call above the set verbosity level is ignored.

The logMessage() function has a number of fixed arguments followed by an arbitrary message string in the printf() style. The system name is required, and should denote the context in which your log message is being produced. For example, an EPICS record support module would supply the record support module name. The subsystem name is optional, but should denote the subsystem from which an error originated. For example, the same record support module may know it's error resulted from a failed call in a given device support module. The subsystem should be the name of the device support module.

In this fashion, a hierarchy of error messages may later be reconstructed, showing the propagation of errors up from low level software.

High detail messages would typically be given a high verboseLevel value. Low detail, routine messages would be given a low value. The user is free to utilize the verboseLevel argument in any fashion, though.

1.2. logDaemon Overview

The logDaemon may be started anywhere on a subnet. It is a single–threaded, UDP based server. Various environment variables and/or command–line options specify what port to use, where the log file directory is, etc.... Most importantly, a configuration file is read which specifies how incoming log messages are to be distributed among one or more files based on the various fields, and whether e–mail should be sent.

The client library will broadcast for the logDaemon using a specific id. The logDaemon with that id will respond, notifying the client library of its IP address and port. All subsequent log messages are transmitted via a single UDP packet. No acknowledgement of successful receipt is given.

The logDaemon can be configured to write a simple ascii file format, one log message per line, or to write an SDDS format log file.

A max–log–file–size may be given. In this case, the log file will be copied to a save directory whenever the size is exceeded. The save directory utilizes file generations, so the log files will reside in the save directory as log.0, log.1, log.2, etc. A simple browsing tool can reconstruct the full history of messages, including those in the currently active log file.

2. Client Library Reference

#include <logDaemonLib.h>

logOpen

int logOpen(LOGHANDLE *h, char *sourceId, char *serviceId);

Open a connection with the logDaemon (not in TCP sense, though, since library is UDP based). User must provide a ptr to a pre–allocated LOGHANDLE. The sourceId is an arbitrary string up to 254 chars in length. The serviceId ptr may be NULL, in which case the default logDaemon is contacted. Otherwise, serviceId is an arbitrary string up to 254 chars in length.

- **h** – ptr to pre–allocated LOGHANDLE struct
- **sourceId** – ptr to null terminated string up to 254 chars in length
- **serviceId** – NULL, or ptr to null terminated string up to 254 chars in length.

Returns:

- 0 – ok
- –1 – error

logGetVerbosity

int logGetVerbosity(LOGHANDLE h);

Retrieve current upper verbosity limit. After a logOpen(), it is set to 99 (max) by default.

- **h** – LOGHANDLE from logOpen() call

Returns:

- current upper verbosity limit

logSetVerbosity

int logSetVerbosity(LOGHANDLE *h, int verbosity);

Set the verbosity upper limit for the currently open log session.

- **h** – ptr to LOGHANDLE from logOpen() call
- **verbosity** – new upper verbosity limit

Returns:

- new upper verbosity limit

logMessage

int logMessage(LOGHANDLE h, char *system, char *subsystem, int verboseLevel, char

*format, ...);

Sends a log message to the logDaemon. The message is time–stamped with secs and usecs past UNIX epoch automatically. A system name must be provided, but subsystem may be a NULL ptr. The verboseLevel may be from 1 to 99. The remaining arguments function like a printf() call and provide the arbitrary text portion of the message.

- **h** – LOGHANDLE from logOpen() call
- **system** – ptr to null terminated string up to 254 chars in length
- **subsystem** – ptr to null terminated string up to 254 chars in length, or NULL
- **verboseLevel** – from 1 to 99
- **format** – printf style format string
- ...

Returns:

- 0 – ok
- –1 – error

logClose

int logClose(LOGHANDLE h);

Close up logDaemon "connection". LOGHANDLE may be reused after closing.

3. logDaemon Reference

Command line options:

```
logDaemon
[-m (text|SDDS)] Log file format. This  option only available if SDDS compiled
in.
[-i <serviceId>] Text name for logDaemon. Defaults if not given.
[-f <config file name>] Configuration file  name. Defaults to log.confg (see -e
option).
[-p <UDP port>] UDP port for daemon to listen on. Defaults if not given.
[-a <log file name>] Default log file if no config file is given. Defaults to
log.file.
[-r] Remove any current log files at startup, and start with fresh ones.
[-h <home dir>] Use this directory for log files. Defaults to current dir.
[-o <save dir>] Use this directory for saved log files. Defaults to ./save.
[-s <max size>] Copy a log file to save dir if it exceeds this size.
[-e] Print example of a config file to stdout.
```

Environment Variables (corresponds to above options in general):

```
LOG_SERVER_ID
```

```
LOG_PORT
LOG_CONFIG
LOG_DEFAULT
LOG_HOME
LOG_SAVEDIR
LOG_MAXSIZE
```

A sample config file (log.config) is as follows:

```
#Example log.config file
#----------------------
# Fields are : separated, and as follows:
# sourceId:system:verboseLevel:dest:destName
# where sourceId is string or *
#       system is string or *
#       verboseLevel is # or range #-# from 1 to 99
#       dest is the string log or mail
#       destName is a log file name if dest is log,
#         or space delimited list of email addrs if
#         dest is mail.
# Specify two log files for msgs from IOC sourceId
IOC:*:1-49:log:iocLogLow.log
IOC:*:50-99:log:iocLogHigh.log
# Next, specify email for msg from any system named DOOM
# Note, if sourceId is IOC, above lines take precedence.
*:DOOM:*:mail:me@aps.anl.gov you@aps.anl.gov
# Finally, specify catch all log for all else
*:*:*:log:catch-all.log
```

The logDaemon writes the following fields for each log message received. The exact format of the output depends on whether you have chosed text or SDDS mode at startup time.

- secs – seconds past unix epoch (time stamp from client clock)
- usecs – microseconds part
- sourceId – string from logOpen() call
- verboseLevel – integer from 1 to 99
- system –
- subsystem –
- message