APS logDaemon and Client Library

C. Saunders, J. Kowalkowski, R. Soliday
Advanced Photon Source

May 2, 2005

Original version of documentation written Oct. 22, 1996

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. A command line utility, logMessage, is also provided for submitting log messages.

1.1 Client Overview

The client library provides the following calls:

```
int logOpen(LOGHANDLE *h, char *serviceId, char *sourceId, char *tagList);
int logString(LOGHANDLE h, char *valueList);
int logArguments(LOGHANDLE h, ...);
int logArray(LOGHANDLE h, char *valueArray[]);
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 class of log messages. 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 tagList is a space delimited set of tag names which correspond to the sourceId. These define the field names which are given values in subsequent logString() or logArguments() calls.

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

The logString(), logArguments(), and logArray() functions submit a log message. They are identical except for the manner in which you supply the series of tag values. logString() expects a single, space delimited string of tag values. logArguments() expects a series of (char *) arguments, each supplying one tag value. The last argument must be NULL. logArray() expects an array of character pointers, each element pointing to one NULL terminated tag value. The last element must be a NULL pointer.

The command line utility logMessage is provided to permit simple, command line submission of log messages. An example is as follows (tag value follows tag name):

```
logMessage -sourceId=IOC -tag=system RecSupport -tag=subsystem AO
```

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 tag values, 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, and are acknowledged with a single UDP packet.

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 *serviceId, char *sourceId, char *tagList);

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 250 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 255 chars in length. The tagList is a space delimited set of tag names. Tag names may not have spaces in them. Each tag name may be up to 255 chars in length.

- **h** - ptr to pre-allocated LOGHANDLE struct
- **sourceId** - ptr to null terminated string up to 250 chars in length
- **serviceId** - NULL, or ptr to null terminated string up to 255 chars in length.
- **tagList** - ptr to null terminated string. Total length limited by UDP packet size, but each space delimited tag name may be no longer than 255 chars.

Returns:

- LOG_OK - connection opened, sourceId and tagList validated

- LOG_TOOBIG - one of the strings supplied is too long
- LOG_ERROR - unable to open connection to logDaemon
- LOG_INVALID - logDaemon rejected given sourceId and tagList

logString

int logString(LOGHANDLE h, char *valueList);

Sends a log message to the logDaemon. The message is time-stamped with secs and usecs past UNIX epoch automatically. The remaining data comes from valueList, a space delimited string of tag values. There must be a value supplied for each tag defined by the logOpen() call. If a tag value has internal spaces, you must double quote the full value. Ie. "val1 "val2 with spaces" val3". The valueList has no size limit other than that imposed by the UDP packet size.

- **h** - LOGHANDLE from logOpen() call
- **valueList** - ptr to null terminated string of space delimited tag values.

Returns:

- LOG_OK - message logged
- LOG_TOOBIG - the valueList string supplied takes packet over UDP size limit.
- LOG_ERROR - unable to send message to logDaemon

logArguments

int logArguments(LOGHANDLE h, ...);

Sends a log message to the logDaemon. The message is time-stamped with secs and usecs past UNIX epoch automatically. The remaining data comes from a series of (char *) arguments, the last of which must be NULL. Ie. logArguments(h, "val1", "val2", NULL); There must be a value supplied for each tag defined by the logOpen() call.

- **h** - LOGHANDLE from logOpen() call
- **...** - series of (char *) arguments. The last argument must be NULL.

Returns:

- LOG_OK - message logged
- LOG_TOOBIG - the total length of tag values takes packet over UDP size limit.
- LOG_ERROR - unable to send message to logDaemon

logArray

int logArray(LOGHANDLE h, char *valueArray[]);

Sends a log message to the logDaemon. The message is time-stamped with secs and usecs past UNIX epoch automatically. The remaining data comes from valueArray, an array of pointers to null terminated value strings. The last element of the array must be NULL. Ie. val[0] = "this"; val[1] = "that" ; val[2] = NULL; logArray(h, val). There must be a value supplied for each tag defined by the logOpen() call.

- **h** - LOGHANDLE from logOpen() call
- **valueArray** - array of pointers to null terminated strings, last element is NULL

Returns:

- LOG_OK - message logged
- LOG_TOOBIG - the total length of tag values takes packet over UDP size limit.
- LOG_ERROR - unable to send message to logDaemon

logClose

int logClose(LOGHANDLE h);

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

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

Returns:

- LOG_OK - connection closed

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.
[**-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 (in bytes).
[**-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_HOME
LOG_SAVEDIR
LOG_MAXSIZE

A configuration file (log.config) is read at startup time (and anytime a kill -HUP is issued). This file is not required, but does allow you to redirect log messages to particular files based on various combinations of tag values. You may also specify email destinations.

In the absence of a log.config file, each unique sourceId is given its own log file. By default, the file is titled <sourceId>.log.

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

```
#Example log.config file
#-----------------------
# Fields are ~ separated, and as follows:
# sourceId~dest~destName~<tag1>=<val1>~<tagn>=<valn>
# where sourceId is string
#       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.
#       <tag1>=<val1> specifies that tag value supplied
#         in message must match <val1> exactly. Any
#         tag names not given will match anything.
# Specify three log files for msgs from IOC sourceId
IOC~log~iocLogMinor.log~severity=Minor
IOC~log~iocLogMajor.log~severity=Major
IOC~log~iocLogOther.log
# Next, specify email for msg from any sourceId DOOM
DOOM~mail~me@aps.anl.gov you@aps.anl.gov
```

Any incoming log message is matched against this data. If the incoming sourceId matches a line in the config file, then each subsequent incoming tag value will be matched against that specified in the file. If a tag name is not given in a config file line, a match is assumed. All complete matches result in either a log file being written, or an email being sent.

The logDaemon records any new incoming sourceId/tagList combination from logOpen() calls. Once a sourceId/tagList is submitted, you may not redefine it without deleting the corresponding line in the log.sourceId file and restarting the daemon. Note that the server will remember sourceId/tagList associations across invocations.

4 logMessage Utility Reference

Command line options:

logMessage [-serviceId=<string>] -sourceId=<string>
-tag=tag1 val1 ... -tag=tagn valn

Example:
logMessage -sourceId=IOCERR -tag=system AORecSupport -tag=subsystem devAOdac
-tag=type ERROR -tag=message "Out of range value given"