DAYLogFile

Inherits From: Object

Declared In: daymisckit/DAYLogFile.h

Protocols: NXTransport

Class Description

A DAYLogFile is used to manage logging information to a UNIX text file. You simply pass it lines of text±as DAYString objects±to append to the log file. If you wish, and this is necessary if multiple processes might write to the log file, you can have the DAYLogFile make use of a DAYLockFile to provide a proper lock on the log file. You can log messages to stdout, stderr, or an arbitrary file.

To use a DAYLogFile in your application, simply create the object using ±**alloc** and ±**init** and then set up the file to be used for logging. This may be done by using any one of the following methods: ±**setFileName:**, ±**usestdout**, or ±**usestderr**. To add a line to the log file, send the ±**addLineToLogFile:** message, which takes a DAYString (or other object which responds to ±**stringValue**) as an argument and returns a YES if successful. The ±**addLineToLogFile:**

method automatically handles opening and closing the file for you, each time that you call it. As such, it can be somewhat slow and resource intensive.

Alternatively, if you plan to write many lines to the log file at a time, then open the log file with an ±**openFile** message, write your lines with the ±**appendToLogFile:andFlush:** method, and then, when you are finished, send the ±**closeFile** message. Note that all three messages return a NO if they are unsuccessful in any way. If, when appending lines, you do not flush them right away, repeated lines will be compressed into a single line saying ªThe last line was repeated x times.º The disadvantage to this is that during the whole time, you are holding the log file locked. If multiple processes need to log problems, then this is probably a bad idea, since they will either have to give up on logging or spin wait. (A way to get around this is to spin off a separate thread to do the logging for the main thread.)

Instance Variables

BOOL **fileOpen**;
BOOL **special**;
FILE ***file**;
id **lastLine**;
id **lockFile**;
id **fileName**;
int **repeats**;

file	File we are writing to; only valid if fileOpen == YES.
fileName	The name of the file being logged to.
fileOpen	YES if the log file is currently opened for appending.

lastLine	The line most recently output by the DAYLogFile object
lockFile	A supporting DAYLockFile (if the log file is protected by a lock)
repeats	How many times the last line has been sent to the file.
special	YES if the DAYLogFile is logging to stderr or stdout.

Method Types

Initializing a DAYLogFile	± init ± copy ± free
Changing parameters	± setFileName: ± setLockFile: ± usestderr ± usestdout
Getting information about	± fileName ± file ± fileIsOpen ± special ± lockFile
Saving to a file	± read: ± write:
Updating the log file	± addLineToLogFile:

± appendToLogFile:andFlush:
± openFile
± closeFile

Instance Methods

addLineToLogFile:

- (BOOL)**addLineToLogFile:***aString*

Opens the log file, obtaining a lock if necessary, and appends *aString* to the file. The argument *aString* should be an object which responds to the ±**stringValue** method. After appending the message, this method then closes the file and returns a YES if successful. A failure is noted by returning a NO.

See also: -**appendToLogFile:andFlush:**

appendToLogFile:andFlush:

- (BOOL)**appendToLogFile:***aString* **andFlush:**(BOOL)*flushFlag*

Appends *aString* to the log file. Before calling this method, you must first open the log file. This method returns a YES upon success and a NO if it fails. This method attempts to coalesce identical lines into the message ªLast line repeated x times.º in order to save space on disk.

See also: -**addLineToLogFile**, -**closeFile**, -**openFile**

closeFile

- (BOOL)**closeFile**

Flush and close the log file if open. If the log file is not open or the flush or close are unsuccessful, a NO is returned. If all is well, a YES is returned.

See also: -**openFile**

copy

- **copy**

Returns a new instance of DAYLogFile which has been initialized to use the same files for logging and locking as the receiver. The copy does not hold any locks or open files which are held by the original object; this is because the file should only be accessible from one place at a time.

See also: -**openFile**, ±**read:**

file

- (FILE *)**file**

Returns the UNIX file which is used for logging. The return value is valid only if the ±**fileIsOpen** message returns a YES. Except for unusual circumstances, you should never need to use this method.

See also: -**openFile**, ±**read:**

fileName

- **fileName**

Returns a DAYString containing the filename of the file the DAYLogFile is logging to.

See also: -**openFile**

fileIsOpen

- (BOOL)**fileIsOpen**

Returns YES if the log file is open and the DAYLogFile holds any locks associated with the log file. Returns NO if this is not the case.

See also: -**openFile**, ±**closeFile**

free

- **free**

Frees the DAYLogFile and it's associated objects (lock file and filename).

init

- **init**

Initializes a new instance of DAYLogFile. It is up to you to finish initialization by providing the DAYLogFile with a path to the log file, and, if desired, a DAYLockFile object.

lockFile

- **lockFile**

Returns the instance of DAYLockFile which is used to lock the log file. Returns nil if no lock file is being used.

See also: -**setLockFile:**

openFile

- (BOOL)**openFile**

Opens the log file for appending. If a lock file has been set up, the DAYLogFile will attempt to obtain a lock on the file before actually opening it. If successful,m this method returns YES; it returns NO upon failure.

See also: -**closeFile**, ±**setFileName:**

read:

- **read:**(NXTypedStream *)*stream*

Restores a DAYLogFile from a stream. An archived DAYLogFile never holds a lock on a logfile and does not hold the file open, thus the new object will be like a newly initialized DAYLogFile object.

See also: -**copy**, **±write:**

setFileName:

- **setFileName:***aString*

Sets the name of the file used for logging. Unless you wish the file to be relative to the application's current working directory, you should specify a full path name for the file.

setLockFile:

- **setLockFile:***newLockFile*

Uses the *newLockFile* as the locking mechanism on the log file. It is up to the programmer to provide a properly initialized DAYLockFile object (or subclass) as the parameter.

See also: -**copy**

special

- (BOOL)**special**

Returns a YES if logging to either stdout or stderr. Returns a NO if logging to a UNIX text file.

See also: -**usestderr**, ±**usestdout**, -**setFileName:**

usestderr

- **usestderr**

Sets up the DAYLogFile to log to stderr instead of a UNIX file.

See also: ±**usestdout**, -**setFileName:**

usestdout

- **usestdout**

Sets up the DAYLogFile to log to stdout instead of a UNIX file.

See also: ±**usestderr**, -**setFileName:**

write:

- **write:**(NXTypedStream *)stream

Archives the DAYLogFile to a stream. An archived DAYLogFile never holds a lock on a logfile and does not hold the file open.

See also: ±**read:**