DAYLockFile

Inherits From: Object

Declared In: daymisckit/DAYLockFile.h

Protocols: NXTransport

Class Description

The DAYLockFile class implements a simple UNIX-based file locking mechanism. Simply, if a specific file (the lock file) exists, then the file it protects is considered locked. In this case, no other process is supposed to attempt to use the protected file. The lock file should contain the PID of the process holding the lock, if it exists. This way, if a process dies unexpectedly without removing a lock, a user with appropriate privileges may manually remove the lock after first making sure that the process is in fact dead.

To use a DAYLockFile in your program, simply create it using the standard ±**alloc** and ±**init** methods and then set the name of the lock file via the ±**setFileName:** method. Use the full path unless you want the lock file in the current working directory. Note that ±**setFileName:** requires

an object which responds to the ±**stringValue** method, such as DAYString or an AppKit control, as its argument. Send a ±**lock** message when you wish to obtain a lock and a ±**unlock** message when you are ready to relinquish the lock. If either message fails, nil will be returned. The ±**haveLock** message returns YES if you currently have the lock.

If you make a copy of a DAYLockFile which has a lock, the copy will *not* have the lock since only one object should be able to hold the lock at a time. Note that if you archive a DAYLockFile to a stream, the archived version does *not* have the lock, even if it did when it was archived. Thus you must still send an ±**unlock** message before exiting the process. (In other words, you can't have a lock that persists between processes.) A DAYLockFile sent **bycopy** through the distributed objects system does not hold a lock, either. This prevents the case where multiple objects believe that they hold a lock. Note that the ±**free** method will remove a lock (if still held) before freeing the DAYLockFile.

If you fork child processes, you should be sure that the *children* and not the parent create the DAYLockFile objects. This is because each child should have an independent lock; if you create the lock and then send it a ±**lock** message before forking, each child will think it has the lock, which should never be the case and could cause the sorts of problems that the DAYLockFile is designed to avoid.

Typically, a DAYLockFile should be used whenever more than one process might attempt to write to a file. In order to maintain the lock's integrity, a UNIX call is made which will atomically check for existence of a file and create it if it doesn't already exist. Be forewarned, however, that this type of lock can only be enforced by ªconsent.º If a process blindly accesses a file without first checking the lock (and aborting if it can't get the lock), then the entire locking mechanism becomes worthless. Thus, if you intend to use a DAYLockFile to protect another file, you must *insure* that every process which will access that file also protects it with a DAYLockFile (of the same name, of course). Also, as noted above, the DAYLockFile itself tries to make sure that only one object can actually hold a lock at a given time, and in most cases will be successful, but this can always be intentionally overridden, lending it useless. It is up to you to be sure that it is used appropriately. Due to the fact that the file system does not enforce these types of locks natively,

carelessness will render them utterly useless.

Instance Variables

BOOL **haveLock**;
id **lockFileName**;

haveLock	YES is we currently own/have the lock.
lockFileName	The name of the file we are using as a lock.

Method Types

Initializing a DAYLockFile	± init ± copy ± free
Changing parameters	± setFileName:
Changing status	± lock ± unlock
Getting information about	± fileName ± haveLock
Saving to a file	± read:

± write:

Instance Methods

copy
- **copy**

Makes a copy of a DAYLockFile. This method returns a DAYLockFile which holds no locks.

fileName
- **fileName**

Returns a DAYString which contains the file used as a lock file.

free
- **free**

Frees an instance of DAYLockFile, releasing any locks held by the object if necessary.

haveLock
- (BOOL)**haveLock**

Returns a YES if the DAYLockFile object currently holds the lock file and NO otherwise.

See also: -**lock**, -**unlock**

init

- **init**

Initializes a new instance of DAYLockFile.

lock

- **lock**

Attempts to obtain the lock file. Returns *self* if successful and *nil* if not.

See also: -**unlock**

read:

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

Reads an archive DAYLockFile from a stream. Note that an unarchived DAYLockFile never holds a lock until you request it to obtain the lock using ±**lock**.

See also: -**write:**

setFileName:

- **setFileName:***aString*

Sets the name of the lock file. You should give a full path to the file; if you do not, then the file will be relative to the current working directory of the application.

unlock

- **unlock**

Attempts to release the lock file. Returns *self* if successful and *nil* if not.

See also: -**lock**

write:

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

Archives the DAYLockFile to a stream. Remember that since only one object can hold the lock at a time, the copy written to the stream is considered to not hold the lock.

See also: -**read:**