Release 1.0 Copyright ©1994 by Steve Quirk All Rights Reserved.

MiscThreadedObject

Inherits From: Object

Declared In: MiscThreadedObject.h

Class Description

A threaded object can run in it's own thread. Simply send a -runInNewThread message to the instance and a thread will be created and the instance's 'run' method will be invoked. The thread will be destroyed when the run method returns. To be useful, a subclass must override the run method - this implementation simply returns self.

By default, the instance is sent a 'free' message after the 'run' method returns. Override this behaviour by sending 'freeAfterRun:NO'. Of course, don't call 'free' until after run returns.

To aid debugging, the name of the thread will be set to whatever is returned by [self name], thus showing something useful in gdb's 'tl' command. Override -name if you want to customize that.

Instance Variables

cthread_t **objThread**;
thread_info_t **info**;
thread_info_t **schedInfo**;
BOOL **freeOnReturnFromRun**;

objThread	The object's cthread handle.
info	Used to hold data returned from thread_info.
schedInfo	Used to hold data returned from thread_info.
freeOnReturnFromRun	Flag to indicate whether instance should be freed when the run method returns.

Method Types

- abort
+ errno
- fork
- freeAfterRun
- freeAfterRun:
+ getThreadLimit
- join
- resume
- run
- runInNewThread
+ setThreadLimit:
- suspend

- switchTo
- threadInfo
- threadSchedInfo

Class Methods

errno
+ (int)**errno**

Returns the current thread's **errno** value.

See also: intro(2) Unix man page.

getThreadLimit
+ (int)**getThreadLimit**

Returns the maximum number of threads for this task. A limit of 0 indicates that limits are not enforced.

See also: **setThreadLimit:**

setThreadLimit:
+ (void)**setThreadLimit:**(int)*newLimit*

Sets the maximum number of threads for this task to *newLimit*. Specify zero if you want no limit.

See also: **getThreadLimit**

Instance Methods

abort

- **abort**

Interupt the receiver. **abort** interrupts system calls; it's usually used along with **suspend**, which stops the receiver from executing any more user code. Sending **abort** to a thread that isn't suspended is risky, since it's difficult to know exactly what system trap, if any, the thread might be executing and whether an interrupt return would cause the thread to do something useful

See also: suspend, cthread_abort()

fork

- **fork**

Fork an instance. The **fork** message creates a new thread of control and sends the instance a **run** message. The thread is not detached and may be waited upon with the **join** method.

The new thread will be named according to [self name], which defaults to the class name.

See also: join, runInNewThread, cthread_fork(), cthread_name()

freeAfterRun

- (BOOL)**freeAfterRun**

Returns a flag indicating whether the instance will be freed when it's **run** method returns.

See also: freeAfterRun:, run

freeAfterRun:

- **freeAfterRun:**(BOOL)*yesOrNo*

Indicates whether the instance should be freed after the **run** method returns. By default, a MiscThreadedObject will be sent a **free** message following **run**. This allows an application to start the object and forget about it.

See also: run, freeAfterRun

join

- (any_t)**join**

This method suspends the sender until the receiver completes its **run** method. Returns 0 if **run** returned self, 1 otherwise.

See also: fork, run, cthread_join()

resume

- **resume**

Decrements the receiver's suspend count.

See also: suspend, thread_resume()

run

- **run**

The focus of action in a MiscThreadedObject. To be useful, subclass must implement this method.

See also: runInNewThread, freeAfterRun

runInNewThread

- **runInNewThread**

Causes the receiver to be forked and detached. You cannot send a **join** message to an instance that has been detached in this way.

See also: fork, cthread_detach()

suspend

- **suspend**

Suspends the receiver by incrementing the suspend count and prevents the thread from executing any more user-level instructions.

See also: resume, threadInfo, thread_suspend()

switchTo

- **switchTo**

Causes the scheduler to do a context switch into the receiver's thread. The sender's thread will be rescheduled.

See also: thread_switch(*t* ,*SWITCH_OPTION_NONE*,*0*)

threadInfo

- (thread_info_t)**threadInfo**

Returns a pointer to the receiver's *thread_basic_info* struct. The struct is defined as

```
struct thread_basic_info {
    time_value_t    user_time;     /* user run time */
    time_value_t    system_time;   /* system run time */
    int             cpu_usage;     /* scaled cpu usage percentage */
    int             base_priority;/* base scheduling priority */
```

```
    int             cur_priority; /* current scheduling priority */
    int             run_state;    /* run state (see below) */
    int             flags;        /* various flags (see below) */
    int             suspend_count;/* suspend count for thread */
    long            sleep_time;   /* number of seconds that thread
                                     has been sleeping */
};
```

See also: threadSchedInfo, thread_info()

threadSchedInfo

- (thread_info_t)**threadSchedInfo**

Returns a pointer to the receiver's *thread_basic_info* struct. The struct is defined as

```
struct thread_sched_info {
    int             policy;       /* scheduling policy */
    int             data;         /* associated data */
    int             base_priority;/* base priority */
    int             max_priority;   /* max priority */
    int             cur_priority; /* current priority */
    boolean_t       depressed;    /* depressed ? */
    int             depress_priority; /* priority depressed from */
};
```

See also: threadInfo, thread_info()