ThreadedApp

Inherits From: Application : Responder : Object

Declared In: ThreadedApp.h

Class Description

ThreadedApp is a subclass of Application that supports multithreaded operation in a simple way. It contains the functionality of the OpenStep NSThread object, but also provides increased support for interaction with the AppKit. The AppKit is not thread-safe and only the main thread can use it. All other threads must message the main thread to perform any functions (drawing, etc) that involve the AppKit. ThreadedApp does this by introducing the notion of *callback* methods. This is a flexible mechanism whereby a thread can request that the main application thread invoke a method on its behalf. This allows a thread to have virtually full access to the AppKit in a way that is easy to understand and use.

The callback mechanism uses Mach messages to make requests to the main AppKit thread. The main thread only receives the messages when it is in an event loop, so it is crucial that the main thread remains responsive to events. (This is only good user-interface design anyways.)

ThreadedApp provides support for locking shared data or code via the CJRLock and CJRConditionLock classes. These classes allow the main AppKit thread to "block" while waiting to acquire a lock, but still remain responsive to events so that other threads may still perform callbacks.

There are three priorities for callback methods: **fast**, **safe**, and **slow**. Fast callbacks are used when a thread needs to display a result, or to change some shared data. An example usage is

```
while( running ){
    /* ... code to update a value */
```

```
        // update the display to reflect the new value
        [NXApp callbackTarget:textField perform:@selector(setIntValue:) with:(id)value];
    }
```

In this case, the thread requests the application (NXApp) to invoke textField's `setIntValue:` method with the parameter `value`. Note that `value` needs to be cast to an `id`, much like using Object's `perform:with:` method. Fast callbacks are invoked directly from the Mach message handler which is dispatched from within the DPSGetEvent function. Because of this, fast callback methods cannot get events (since DPSGetEvent is not reentrant). In particular, this means that fast callback methods should not use CJRLocks since acquiring a lock from within the main thread involves running a modal-event loop (see CJRLock). Fast callback messages are sent with a priority of NX_MODALRESPTHRESHOLD + 1 and will therefore get through any user modal event loops.

Safe callback methods are able to use the full functionality of the AppKit and run their own modal event loops if desired. Safe callback methods are not run directly from the Mach message handler. Instead, the Mach message handler posts an EV_CALLBACK event to the application's event queue. This is an NX_APPDEFINED event, handled by ThreadedApp's applicationDefined: method which performs the actual call-back. Because the safe callback method is dispatched in response to a regular application event, it may do anything you might normally do in response to an event. Safe callback methods also are able to use CJRLocks and CJRConditionLocks to access shared data. Safe callback messages are sent with a priority of NX_MODALRESPTHRESHOLD + 1 and will therefore get through any user modal event loops.

Slow callbacks are like safe callbacks, but are sent with a message priority of NX_BASETHRESHOLD and will therefore not interrupt any high priority things a user might be doing (attention panels, scrolling, other modal loops).

Fast, safe and slow callback methods can be blocking or non-blocking. The **callbackTarget:perform:** methods are non-blocking: they don't wait for the invoked method to return. The **callbackAndWaitTarget:perform:** methods block until the main thread has executed the method and returned the result. The result is an `id`, but can be typecast to any `int`-sized type.

If a callback method is invoked from the main thread of execution, the Object's perform: method is directly used to invoke the method and no messages are sent.

Some basic guidelines for writing multi-threaded applications using ThreadedApp:

1. The main AppKit thread just runs the event loop and responds to events. If responding to an event takes any amount of time, it should be done in a separate thread. If the main AppKit thread is not responsive to events, other threads will not be able to perform callbacks.
2. Fast callback methods have lower overhead and should be used in most cases Ð to perform display updates or to access shared data. Fast callback methods can be used to ensure that all access to a piece of shared data is pipelined through the main thread. This is a simple "lockless" method of synchronizing access to data, but it is somewhat slower that using locks (mutexes).
3. If a thread callback method needs to use locks, it should use a safe callback method and the CJRLock class to ensure that the main thread doesn't block and become non-responsive to events.
4. If a thread needs to interact with the user, it should use a slow callback method to avoid interrupting the user.

Note that callback methods go through the window server, so if you have a lot of threads and callbacks, you will see an increased load on the window server. It is also possible for a lot of callbacks to overwhelm the window server's message queue which will result in a thread pausing until the queue becomes available.

Instance Variables

None declared in this class.

Method Types

Creating and freeing instances + new
- free

Getting and peeking at events - getNextEvent:
- getNextEvent:waitFor:threshold:

Handling user actions and events - applicationDefined:

Managing threads	± detachNewThreadSelector:toTarget:withObject: ± exitCurrentThread ± sleepCurrentThread:
Performing method callbacks	± callbackAndWaitTarget:perform: ± callbackAndWaitTarget:perform:with: ± callbackTarget:perform: ± callbackTarget:perform:with: ± safeCallbackAndWaitTarget:perform: ± safeCallbackAndWaitTarget:perform:with: ± safeCallbackTarget:perform: ± safeCallbackTarget:perform:with: ± slowCallbackAndWaitTarget:perform: ± slowCallbackAndWaitTarget:perform:with: ± slowCallbackTarget:perform: ± slowCallbackTarget:perform:with:

Class Methods

new

+ **new**

Initializes Mach ports to receive messages on and set the Objective-C runtime to be thread-safe. Also initializes a global variable (cthread_t)**mainThread** that is the cthread identifier for the main AppKit thread. Returns **self**.

Instance Methods

applicationDefined:

- **applicationDefined:**(NXEvent *)*theEvent*

Invoked when the application receives an application-defined (NX_APPDEFINED) event. This method performs the safe and slow callbacks. The delegate's applicationDefined: method does **not** get invoked. Returns **self**.

callbackAndWaitTarget:perform:

- **callbackAndWaitTarget:***aTarget*
  perform:(SEL)*aSelector*

This method performs a fast callback to the main thread of execution. The given method will be invoked by the main
thread of execution as if it were called by

```
[aTarget perform:aSelector]
```

Any thread may invoke this method. This method waits until the called-back method finishes execution and returns the value that the called-back method returned (typecast to an `id`).

See also: ± callbackAndWaitTarget:perform:with:

callbackAndWaitTarget:perform:with:

- **callbackAndWaitTarget:***aTarget*
  perform:(SEL)*aSelector*
  with:*anObject*

This method performs a fast callback to the main thread of execution. The given method will be invoked by the main
thread of execution as if it were called by

```
[aTarget perform:aSelector with:anObject]
```

The argument *anObject* can be any parameter, but it must be typecast to an `id`. Any thread may invoke this method. This method waits until the called-back method finishes execution and returns the value that the

called-back method returned (typecast to an `id`).

See also: ± callbackTarget:perform:with:, **- slowCallbackAndWaitTarget:perform:with:**

callbackTarget:perform:
- (void)**callbackTarget:***aTarget*
perform:(SEL)*aSelector*

This method performs a fast callback to the main thread of execution. The given method will be invoked by the main
thread of execution as if it were called by

```
[aTarget perform:aSelector]
```

Any thread may invoke this method. This method is non-blocking: the caller does not wait for the method to return.

See also: ± callbackTarget:perform:with:

callbackTarget:perform:with:
- (void)**callbackTarget:***aTarget*
perform:(SEL)*aSelector*
with:*anObject*

This method performs a fast callback to the main thread of execution. The given method will be invoked by the main
thread of execution as if it were called by

```
[aTarget perform:aSelector with:anObject]
```

The argument *anObject* can be any parameter, but it must be typecast to an `id`. Any thread may invoke this method. This method is non-blocking: the caller does not wait for the method to return.

See also: ± callbackAndWaitTarget:perform:with:, **- slowCallbackTarget:perform:with:**

detachNewThreadSelector:toTarget:withObject:

- (void)**detachNewThreadSelector:**(SEL)*aSelector*
  toTarget:*aTarget*
  withObject:*anObject*

Starts the method in a new thread of execution. The method is invoked as if it were called by

```
[aTarget perform:aSelector with:anObject]
```

Any thread may invoke this method.

See also: - **exitCurrentThread**, - **sleepCurrentThread:**

exitCurrentThread

- **exitCurrentThread**

Exits the caller's thread of execution.

See also: - **detachNewThreadSelector:toTarget:withObject:**, - **sleepCurrentThread:**

free

- **free**

Removes Mach message handlers and deallocates the Mach ports before calling Application's free method.

getNextEvent:

- (NXEvent *)**getNextEvent:**(int)*mask*

This event overrides the Application class's method to include NX_APPDEFINED events in the mask if they aren't already included, and to dispatch them using the **applicationDefined:** method. This allows callbacks and EV_UNLOCK events to get through unsuspecting modal loops such as for handling mouse-down events.

See also: - **getNextEvent:waitFor:threshold:**

getNextEvent:waitFor:threshold:
- (NXEvent *)**getNextEvent:**(int)*mask*
waitFor:(double)*timeout*
threshold:(int)*level*

This event overrides the Application class's method to include NX_APPDEFINED events in the mask if they aren't already included, and to dispatch them using the **applicationDefined:** method. This allows callbacks and EV_UNLOCK events to get through unsuspecting modal loops such as for handling mouse-down events.

See also: - **getNextEvent:**

safeCallbackAndWaitTarget:perform:
- **safeCallbackAndWaitTarget:***aTarget*
perform:(SEL)*aSelector*

This method performs a safe callback to the main thread of execution. The given method will be invoked by the main
thread of execution as if it were called by

```
[aTarget perform:aSelector]
```

Any thread may invoke this method. This method waits until the called-back method finishes execution and returns the value that the called-back method returned (typecast to an `id`).

See also: - **safeCallbackAndWaitTarget:perform:with:**

safeCallbackAndWaitTarget:perform:with:
- **safeCallbackAndWaitTarget:***aTarget*
perform:(SEL)*aSelector*
with:*anObject*

This method performs a safe callback to the main thread of execution. The given method will be invoked by the main
thread of execution as if it were called by

```
[aTarget perform:aSelector with:anObject]
```

The argument *anObject* can be any parameter, but it must be typecast to an `id`. Any thread may invoke this method. This method waits until the called-back method finishes execution and returns the value that the called-back method returned (typecast to an `id`).

See also: ± callbackAndWaitTarget:perform:with:, **- safeCallbackTarget:perform:with:**

safeCallbackTarget:perform:

- (void)**safeCallbackTarget:***aTarget*
  perform:(SEL)*aSelector*

This method performs a safe callback to the main thread of execution. The given method will be invoked by the main
thread of execution as if it were called by

```
[aTarget perform:aSelector]
```

Any thread may invoke this method. This method is non-blocking: the caller does not wait for the method to return.

See also: ± callbackTarget:perform:with:, **- safeCallbackAndWaitTarget:perform:with:**

safeCallbackTarget:perform:with:

- (void)**safeCallbackTarget:***aTarget*
  perform:(SEL)*aSelector*
  with:*anObject*

This method performs a safe callback to the main thread of execution. The given method will be invoked by the main

thread of execution as if it were called by

```
[aTarget perform:aSelector with:anObject]
```

The argument *anObject* can be any parameter, but it must be typecast to an `id`. Any thread may invoke this method. This method is non-blocking: the caller does not wait for the method to return.

See also: ± callbackTarget:perform:with:, **- safeCallbackAndWaitTarget:perform:with:**

sleepCurrentThread:

- (void)**sleepCurrentThread:**(int)*msec*

Performs a context switch and puts the current thread to sleep for msec milliseconds. Use this as an alternative to **usleep()** which is not thread-safe.

See also: - detachNewThreadSelector:toTarget:withObject:, **- exitCurrentThread**

slowCallbackAndWaitTarget:perform:

- **slowCallbackAndWaitTarget:***aTarget*
  perform:(SEL)*aSelector*

This method performs a slow callback to the main thread of execution. The given method will be invoked by the main
thread of execution as if it were called by

```
[aTarget perform:aSelector]
```

Any thread may invoke this method. This method waits until the called-back method finishes execution and returns the value that the called-back method returned (typecast to an `id`).

See also: - slowCallbackAndWaitTarget:perform:with:

slowCallbackAndWaitTarget:perform:with:

- **slowCallbackAndWaitTarget:***aTarget*

perform:(SEL)*aSelector*
with:*anObject*

This method performs a slow callback to the main thread of execution. The given method will be invoked by the main
thread of execution as if it were called by

```
[aTarget perform:aSelector with:anObject]
```

The argument *anObject* can be any parameter, but it must be typecast to an `id`. Any thread may invoke this method. This method waits until the called-back method finishes execution and returns the value that the called-back method returned (typecast to an `id`).

See also: ± callbackAndWaitTarget:perform:with:, **- slowCallbackTarget:perform:with:**

slowCallbackTarget:perform:
- (void)**slowCallbackTarget:***aTarget*
perform:(SEL)*aSelector*

This method performs a slow callback to the main thread of execution. The given method will be invoked by the main
thread of execution as if it were called by

```
[aTarget perform:aSelector]
```

Any thread may invoke this method. This method is non-blocking: the caller does not wait for the method to return.

See also: ± callbackTarget:perform:with:, **- slowCallbackAndWaitTarget:perform:with:**

slowCallbackTarget:perform:with:
- (void)**slowCallbackTarget:***aTarget*
perform:(SEL)*aSelector*
with:*anObject*

This method performs a slow callback to the main thread of execution. The given method will be invoked by the main
thread of execution as if it were called by

```
[aTarget perform:aSelector with:anObject]
```

The argument *anObject* can be any parameter, but it must be typecast to an `id`. Any thread may invoke this method. This method is non-blocking: the caller does not wait for the method to return.

See also: ± callbackTarget:perform:with:, **- slowCallbackAndWaitTarget:perform:with:**