OOPSTimer

Inherits From: Object

Declared In: OOPSTimer.h

Class Description

OOPSTimer encapsulates the DPS functionality for setting a ªtimed entryº. This involves a mechanism that's called repeatedly at a given time interval. A time interval determines the number of seconds between calls to the timed entry. Whenever an application based on the Application Kit attempts to retrieve events from the event queue, it also checks (depending on *priority*) to determine whether the timed entries are due to be called.

Target objects and action methods provide the mechanism by which timer's interact with other objects in an application. A target is an object that a OOPSTimer has effect over. The target class defines an action method to enable its instances to respond to timed event; an action method takes only one argument: the **id** of the timer that invokes it. When it receives an action message, a target can use the **id** to send a message requesting additional information from the timer about its status. It has also the ability to handle a stop action which will be called if the timer stops by itself. The OOPSTimer is not a control but behaves almost like one, but it has not the ability to continue sending the action up through the target chain.

To distinguish different timer object's there is a tag which can be set and asked for.

The timer can be controlled by controls with the action messages **startTimer:** , **stopTimer:** , **takeRepeatTimeIntervalFrom:** , **takeNumberOfRepeatsFrom:**.

> **Note:** The time interval set is not guarantied, many things can slow the timed entry calls down but the OOPSTimer is doing its best. So please do not use on the timer as a clock.

Instance Variables

float **repeatTimeInterval**;
int **numberOfRepeats**;
int **numberOfRepeatsLeft**;
int **priority**;
int **tag**;
int **_auto**;
id **_target**;
SEL **_action**;
SEL **_stopAction;**
DPSTimedEntry **_timer**;

repeatTimeInterval	The time elapsing between repeats.
numberOfRepeats	Number of repeats until timer stops.
numberOfRepeatsLeft	Number of repeats leftuntil timer stops.
priority	The priority used , measured against application's priority threshold.

tag	An integer used to identify the timer.
_auto	Flag for autostart timer when awake. Private.
_paused	Flag for paused timer. Private.
_target	The object that is sent the timer's action. Private.
_action	The message that the timer sends to its target. Private.
_stopAction	The message that the timer sends to its target when it reaches the last repeat. Private.
_timer	The timer's DPSTimedEntry. A unique identifier. Private.

Method Types

Initializing and freeing a String	± init ± initWithRepeatTimeInterval: ± free -awake
Configuring a OOPSTimer	± setPriority: ± setRepeatTimeInterval: ± setNumberOfRepeats:
Querying attributes and state	± priority ± repeatTimeInterval ± numberOfRepeats

	± numberOfRepeatsLeft ± isRunning - autoStart
Manipulating the OOPSTimer	- startTimer: - pauseTimer: - stopTimer: - setAutoStart:
Target and action	- setAction: - action - setStopAction: - stopAction - setTarget: - target
Interacting with Controls	- takeRepeatTimeIntervalFrom: - takeNumberOfRepeatsFrom:
Assigning a tag	- setTag: - tag
Archiving	± read: ± write:

Instance Methods

action
- (SEL)**action**

Returns the timer's action method.

See also: - **setAction:**, - **target**, - **stopAction**

autoStart
- (BOOL)**autoStart**

Returns a BOOL indicating whether the autostart mechanism is on or off.

See also: - **setAutoStart**

awake
- **awake**

Starts the timer if autoStart is on. You should never invoke this method directly.

free
- **free**

Deallocates the timer and kills the timed entry.

init
- **init**

Initializes a new timer. The OOPSTimer is set to repeat every 1.0 second and do this forever with priority

NX_BASETRESHOLD. Starting the OOPSTimer must be handled explicitly with **startTimer:**. Returns **self**.

See also: **-initWithRepeatTimeInterval:. -initWithRepeatTimeInterval:target:action:**

initWithRepeatTimeInterval:
- initWithRepeatTimeInterval:(float)someTime

Initializes a new timer and sets the timer's repeatTimeInterval to someTime. Starting must be done explicitly with **startTimer:**. Returns **self**.

See also: **- init, -initWithRepeatTimeInterval:target:action:**

initWithRepeatTimeInterval:target:action:
- initWithRepeatTimeInterval:(float)someTime target:aTarget action:(SEL)aSelector

Same as **initWithRepeatTimeInterval** but with the ability to set the target and action. Returns **self**.

See also: **- init, -initWithRepeatTimeInterval:**

isRunning
- (BOOL)**isRunning**

Returns YES if the timer is running, eg. has not reached the number of repeats assigned. If the timer is paused or stoped it returns NO.

See also: **- numberOfRepeatsLeft, - pauseTimer:, - stopTimer:**

numberOfRepeats

- (int)numberOfRepeats

Returns the number of repeats until the timer stops by itself. If the timer is set to continue forever the method will return zero (0). If the timer is started this is the number of repeats the timer will commit.

See also: - setNumberOfRepeats:, -numberOfRepeatsLeft

numberOfRepeatsLeft

- (int)numberOfRepeatsLeft

Returns the number of repeats left until the timer stops by itself. If the timer is set to continue forever the method will return the number of repeats done but negative.

See also: - setNumberOfRepeats:, -numberOfRepeats

pauseTimer:

- pauseTimer:sender

Restarts the timer if it is paused or pauses it if it is running. When restarting it continues at numberOfRepeatsLeft. Returns **self**.

See also: - startTimer:, - stopTimer:

priority

- (int)priority

Returns the priority set to the timer.

See also: - **setPriority:**

read:

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

Reads the String from the typed stream *stream*. Returns **self**.

See also: - **write:**

repeatTimeInterval

- (float)**repeatTimeInterval**;

Returns a float representing the time interval in which the timer activates itself and its target.

See also: - **setRepeatTimeInterval:**

setAction:

- **setAction:**(SEL)*aSelector*

Sets the timer's action method to *aSelector.* Returns **self**.

See also: - **action**, - **setTarget:**, **setStopAction:**

setAutoStart:

- **setAutoStart:**(BOOL)*flag*

Sets the timer's autoStart mechanism on or off. The autostart mechanism starts the timer as soon as the object awakes. Returns **self**.

See also: **- autoStart**

setNumberOfRepeats
- (int)setNumberOfRepeats:(int)someNumber

Set the timer's number of repeats until it stops by itself to *someNumber*. If set to 0 or less it continues forever but still counting **numberOfRepeatsLeft** down. This could be handy if it is set to continue forever and there is a need for counting the number of repeats done. The method **numberOfRepeatsLeft** will then return the number of repeats done but negative. Returns **self**.

See also: **- numberOfRepeats:**

setPriority:
- setPriority:(int)aPriority

An application's priority threshold can be set explicitly as an integer from 0 to 31 through a call to **DSPGetEvent()** or **DPSPeekEvent()**. It's against this threshold that *priority* is measured (note that *priority* can be no greater than 30Ðthe additional threshold level, 31, is provided to disallow all inter-event function calls). However, if you're using the Application Kit, you should access the event queue through Application class methods such as **getNextEvent:**. Although some of these methods let you set the priority threshold explicitly, you typically invoke the methods that set it automatically. Such methods use only three priority levels:

Constant	**Meaning**
NX_BASETHRESHOLD	Normal execution
NX_RUNMODALTHRESHOLD	An attention panel is being run
NX_MODALRESPTHRESHOLD	A modal event loop is being run

When applicable, you should use these constants as the value for *priority*. For example, if you want *handler* to be called during normal execution, but not if an attention panel or a modal loop is running, then you would set *priority* to

NX_BASETHRESHOLD. Returns **self**.

See also: - priority

setRepeatTimeInterval:
- setRepeatTimeInterval:(float)someTime;

Set the timer's repeattime interval to *someTime* . Returns **self**.

See also: - repeatTimeInterval

setTag:
- setTag:(int)*anInt*

Sets the timer's tag to *anInt*. The tag can be used to identify the timer in an Application that contains multiple timer's. Returns **self**.

See also: - tag

setTarget:
- setTarget:*anObject*

Sets the timer's target to *anObject*. This is the object that is sent the timer's action method. Returns **self**.

See also: - target, **- setAction:**

startTimer:
- startTimer:sender

Starts the timer. Returns **self**.

See also: **- stopTimer:, pauseTimer:**

stopAction

- (SEL)**stopAction**

Returns the timer's action method which is invoked when the timer has reached no more repeats left.

See also: **- setStopAction:**, **- target**, **- action**

stopTimer

- stopTimer:sender

Stops the timer. Returns **self**.

See also: **- startTimer:, pauseTimer:**

tag

- (int)**tag**

Returns the timer's tag. The tag can be used to identify the timer in an Application that contains multiple timer's.

See also: **- setTag:**

takeNumberOfRepeatsFrom:

- (int)**takeNumberOfRepeatsFrom:sender**

Interacts with controls for automatic change of the timer's number of repeats until it stops by itself. Takes the senders

intValue and uses it to set its number of repeats. Returns **self**.

See also: **- takeRepeatTimeIntervalFrom:, setNumberOfRepeats**

takeRepeatTimeIntervalFrom:
- (int)**takeRepeatTimeIntervalFrom:sender**

Interacts with controls for automatic change of the timer's repeatTimeInterval. Takes the senders floatValue and uses it to set its repeat time interval. Returns **self**.

See also: **- takeNumberOfRepeatsFrom:, setRepeatTimeInterval:**

target
- target

Returns the timer's target, the object that is sent the timer's action method.

See also: **- setTarget:**, **- action, - stopAction,**

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

Writes the String to the typed stream *stream*. Returns **self**.

See also: **- read:**