Timer

Inherits From: Object

Declared In: Timer.h

Class Description

The Timer class is a flexible object that generates an action on its target at specified intervals. It is also a value-holder, which is capable of wrapping to its initial value at specific intervals. Functionality to synchronize a Timer at given intervals is also included.

Instance Variables

id **target**;
sel **action**;
DPSTimedEntry **entry**;
double **period**;
int **priority**;

BOOL **visibleDebug**;
BOOL **wrap**;
BOOL **sync**;
int **syncValue**;
float **value**;
float **startValue**;
float **wrapValue**;
float **incrementBy**;

target	The object to which the action method will be sent.
action	The message that the Timer sends to it's target.
entry	The timed entry that's currently active.
period	The length of time between actions.
priority	The priority at which the timed entry is run.

visibleDebug	Determines whether errors will be reported visibly.
wrap	Determines whether the Timer will wrap.
sync	Determines whether the Timer will be synchronized to the system clock.
syncValue	The number of entries between synchronizations.
value	The numerical value of the Timer.
startValue	The value the Timer starts to count at.
wrapValue	The value over which the Timer will wrap.
incrementBy	The amount added to the *value* every *period* seconds.

Adopted Protocols

IBObject - getInspectorClassName

Method Types

Initializing and freeing the Timer - init
- free

Controlling the Timer's behaviour - start:
- stop:
- pause:
- resume:

Target/action methods - target

	- setTarget: ± action ± setAction:
Managing the entry	- entry - period - setPeriod: - priority - setPriority:
Setting up the Timer	- visibleDebug - setVisibleDebug: - sync - setSync: - syncValue - setSyncValue: - synchronize:

	- startValue - setStartValue: - wrap - setWrap: - wrapValue - setWrapValue: - floatValue - setFloatValue: - incrementBy - setIncrementBy:
Archiving	- read: - write:
Setting up the Timer	- visibleDebug - setVisibleDebug:

Instance Methods

action

- (SEL)**action**

Returns the selector that is sent to *target* (with *self* as the argument) every *period* seconds.

entry

- (DPSTimedEntry)**entry**

Returns the Timed Entry number that the Timer is currently using. Don't mess with this unless you know what you're doing.

floatValue

- (float)**floatValue**

Returns the value of the timer's *value* instance variable. This value is incremented by *incrementBy* every *period* seconds.

free

- **free**

Frees the Timer instance, stopping it if necessary.

incrementBy

- (float)**incrementBy**

Returns the value with which the Timer's instance variable *value* will be incremented every *period* seconds.

init

- **init**

Initializes the Timer instance to a ready state.

pause:
- **pause:***sender*

Stops the timer until a **resume:** method is called.

period
- (double)**period**

Returns the number of seconds between incrementations.

priority
- (int)**priority**

Returns an integer indicating the priority level at which the Timer will be run.

read:

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

Reads an instance of the timer from *stream*.

resume:

- **read:***sender*

Resumes the timer, if it was stopped with **pause:**.

setAction:

- **setAction:**(SEL)*anAction*

Sets the action that will be sent to *target* when the timer fires. *self* is sent as the action's argument.

setFloatValue:

- **setFloatValue:**(float)*aValue*

Sets the Timer's *value* to be *aValue*. The default is 0.0.

setIncrementBy:

- **setIncrementBy:**(float)*aValue*

Sets the Timer's *incrementBy* instance variable to be *aValue*. This value is the amount that is added to *value* when the timer fires. The default is 1.0.

setPeriod:

- **setPeriod:**(double)*aPeriod*

Sets the Timer's *period* instance variable to be *aPeriod*. This is the actual number of seconds between timer actions. *period* is constraind to [0.05, MAXDOUBLE]. The default is 1.0.

setPriority:

- **setPriority:**(float)*aPriority*

Sets the Timer's *priority* instance variable to be *aPriority*. *priority* is constrained to [0,30]. Priorities in this case are on the same scale as the NX_BASETHRESHOLD, NX_RUNMODALTHRESHOLD, NX_MODALRESPTHRESHOLD constants. The default is NX_BASETHRESHOLD.

setStartValue:

- **setStartValue:**(float)*aValue*

Sets the Timer's *startValue* instance variable to be *aValue*. Note that *value* is set to *startValue* only on a call to **start:**. The default is 0.0.

setSync:

- **setSync:**(BOOL)*yn*

Sets the Timer's *synce* instance variable to be *yn*. If yn is YES, the Timer will synchronize from time to time. The default is NO.

setSyncValue:

- **setSyncValue:**(int)*aValue*

Sets the Timer's *syncValue* instance variable to be *aValue*. Every *syncValue* entries, **synchronize:** is called. The default is 60.0.

setTarget:

- **setTarget:***anObject*

Sets the Timer's *target*.

setVisibleDebug:

- **setVisibleDebug:**(BOOL)*yn*

If YES, error messages will be shown in an alert panel, as well as on the console. Default is NO.

setWrap:

- **setWrap:**(BOOL)*yn*

If YES, *value* will revert to *startValue* whenever *value* is incremented to exceed *wrapValue*. Default is YES.

setWrapValue:

- **setWrapValue:**(float)*aValue*

Sets *wrapValue*, the instance variable that bounds *value*. Default is 60.0.

start:

- **start:***sender*

Sets *value* to *startValue*, and starts the timer.

startValue
- (float)**startValue**

Returns *startValue*.

sync
- (BOOL)**sync**

Returns *sync*.

synchronize
- **synchronize**

Trues *value* to a calculated exact value based on *period* and the clock.

syncValue
- (int)**syncValue**

Returns *syncValue*.

stop:
- **stop:***sender*

Stops the timer. *value* is not reset.

target
- **target**

Returns the Timer's target.

wrap
- (BOOL)**wrap**

Returns *wrap*, which determines whether *value* will become *startValue* after *wrapValue* is exceeded. Default is YES.

visibleDebug

- (BOOL)**visibleDebug**

Returns *visibleDebug*.

wrapValue

- (float)**wrapValue**

Returns *wrapValue*.

write:

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

Writes the timer to *stream*.