ZINCKSOFT

Give The Dream New Life.

# JSU.Timer

*The complete Module API Reference*

# Quick Introduction

The JSU Timer Module (**JSU.Timer / JSUTimer**) gives you the possibility to create timers that execute user actions when a specific time has passed. It mimics the functionality from modern languages like Delphi, Java or C#, making your code more readable and less difficult.

## Including the module in your page

Before including this module in your page, first include the core module:

```
<script type="text/javascript" src="jsu.core.js"></script>

<script type="text/javascript" src="jsu.timer.js"></script>
```

# JSU.Timer API Reference

## A. Constructor

***Syntax:***

```
JSU.Timer(task [, interval [, runOnce]]])
```

***Arguments:***

- ***task***: a function to be executed every time the interval period passes;
- ***interval***: the time in miliseconds to wait between the executions of task (how often the task will be executed) in case of a persistent timer, and the time to wait before executing the task in case of a run-once timer;
- ***runOnce***: a boolean indicating whether it's a persistent timer or a run-once timer; a run-once timer will stop itself after it executes the task.

***Returns:*** A JSU.Timer object.

***Example:***

```
var t = new JSU.Timer(function() {
        alert("hello");
    }, 1000);
```

The code above creates a new JSU.Timer instance. This is a persistent timer, and gets a task that shows the user a "hello" message every one second.

## B. Properties

### 1. __task__

This is a private property, do not use it directly and use the getter/setter methods provided. This property holds the function to be executed.

### 2. __timer__

This is a private property, do not use it directly and use the getter/setter methods provided. This property holds an ID of JSU.Timer's internal JavaScript timer.

### 3. __runonce__

This is a private property, do not use it directly and use the getter/setter methods provided. This property holds a boolean indicating whether the timer is persistent or not.

### 4. __interval__

This is a private property, do not use it directly and use the getter/setter methods provided. This property holds a number representing either the interval at which the task will be executed in case of a persistent timer, or the delay time before executing the task in case of a run-once timer.

### 5. __running__

This is a private property, do not use it directly and use the getter/setter methods provided. This property holds a boolean indicating whether the timer is running or not.

## C. Methods

### 1. start

Starts the timer. If the timer is already running, nothing will happen.

***Syntax:***

```
start([delay])
```

***Arguments:***

- ***delay***: the delay time in miliseconds to wait before actually starting the timer; this argument is optional and defaults to 0.

***Returns:*** The current JSU.Timer object, allowing you to chain methods.

***Example:***

```
t.start(); // starts the timer right away
t.start(5000); // starts the timer after 5 seconds
```

### 2. stop

Stops the timer.

***Syntax:***

```
stop()
```

***Returns:*** The current JSU.Timer object, allowing you to chain methods.

***Example:***

```
t.stop(); // stops the timer right away
```

### 3. restart

Restarts the timer.

***Syntax:***

```
restart()
```

***Returns:*** The current JSU.Timer object, allowing you to chain methods.

***Example:***

```
t.restart(); // restarts the timer right away
```

### 4. pause

Pauses the timer.

***Syntax:***

```
pause([resumeAfter])
```

***Arguments:***

- ***resumeAfter***: the delay time in miliseconds to wait before resuming the timer; this argument is optional and defaults to 0, meaning it won't automatically resume the timer.

***Returns:*** The current JSU.Timer object, allowing you to chain methods.

***Example:***

```
t.pause(); // pauses the timer until calling the start or restart method
t.pause(5000); // pauses the timer and automatically resumes it after 5 sec.
```

### 5. setRunOnce

Sets whether the timer is a persistent or a run-once timer. You can set this even if the timer is running, because it will take effect only after a restart.

***Syntax:***

```
setRunOnce(b)
```

***Arguments:***

- ***b***: a boolean indicating the timer's type (true meaning that it's a run-once timer.

***Returns:*** The current JSU.Timer object, allowing you to chain methods.

***Example:***

```
t.setRunOnce(false); // sets the timer as a persistent timer
```

### 6. setInterval / setTimeout

Sets the interval at which the task is executed, or the time before the task is executed once. The **setInterval** is used for persistent timers, while the **setTimeout** is used for run-once timers. Notice that using **setInterval** on a run-once timer is equivalent to using **setTimeout**, but beware that using **setTimeout** on a persistent timer will convert it into a run-once timer.

***Syntax:***

```
setInterval(ms)
setTimeout(ms)
```

***Arguments:***

- ***ms***: the interval/timeout period in miliseconds.

***Returns:*** The current JSU.Timer object, allowing you to chain methods.

***Example:***

```
t.setInterval(1000); // executes the task every one second
t.setTimeout(5000); // executes the task only once after 5 sec.
```

### 7. setTask

Sets the timer's task to execute. Notice that changing the task of a running timer will have an immediate effect, making it use the new task instead of the old one, so be careful with this method.

***Syntax:***

```
setTask(task)
```

***Task syntax:***

```
function()
```

***Arguments:***

- ***task***: the new task that will be assigned to the timer.

***Returns:*** The current JSU.Timer object, allowing you to chain methods.

***Example:***

```
var tk1 = function() { alert("a"); },
    tk2 = function() { alert("b"); };

var t = new JSU.Timer(tk1, 1000);
t.start();
t.setTask(tk2); // changes the timer's task while running
```

### 8. runsOnce

Checks whether the timer is a run-once or a persistent timer.

***Syntax:***

```
runsOnce()
```

***Returns:*** A boolean, true for run-once timers and false for persistent ones.

### 9. isRunning

Checks whether the timer is currently running or not.

***Syntax:***

```
isRunning()
```

***Returns:*** A boolean.